31/10/19

COMPILER DESIGN

[4th PDF - 242-248]

⊛ LR parsing:

If some part of the string is matching with some part of production then it is represented by.

$A \rightarrow X Y \cdot Z$

match found in input string

$\alpha B \beta$

$XY \ldots$

- Any production rule with · notation is called an item. (LR(0) item).

⊛ Closure of set of items: (I) → Closure (I)

Grammer:

$E \rightarrow \cdot E + T \mid \cdot T$

$T \rightarrow \cdot T * F \mid \cdot F$

$F \rightarrow \cdot id$

1. $I = \{ E \rightarrow \cdot E + T,\ E \rightarrow \cdot T,\ T \rightarrow \cdot T * F,\ T \rightarrow \cdot F,\ F \rightarrow \cdot id \}$.

2. CLOSURE (I) ← I

9 → SLR automata / LR(0) automata construction.
→ complete the following SLR automaton.

3. $A \rightarrow \alpha . B\beta$ (item) & production rules

$B \rightarrow .\gamma$.

add, $B \rightarrow .\gamma$ to CLOSURE(I).

* Canonical LR(0): collection / exhaustive set of all the closures.

Each of these sets are considered as one state. All these states are known as finite automata or SLR automata or LR(0) automata.

* GOTO → function that is used to build LR(0) / SLR automata. It takes 2 inputs. Transition from one item set to another. GOTO(I, x)

* Augment Grammer:

$E' \rightarrow E$

↑ special starting symbol.

If this is added to an existing grammer then it becomes an augment grammer. This is used to stop the parsing.

* closure of Item Sets:

Augment grammer:

$I = E' \rightarrow .E$, $E \rightarrow .E + T$, $E \rightarrow .T$

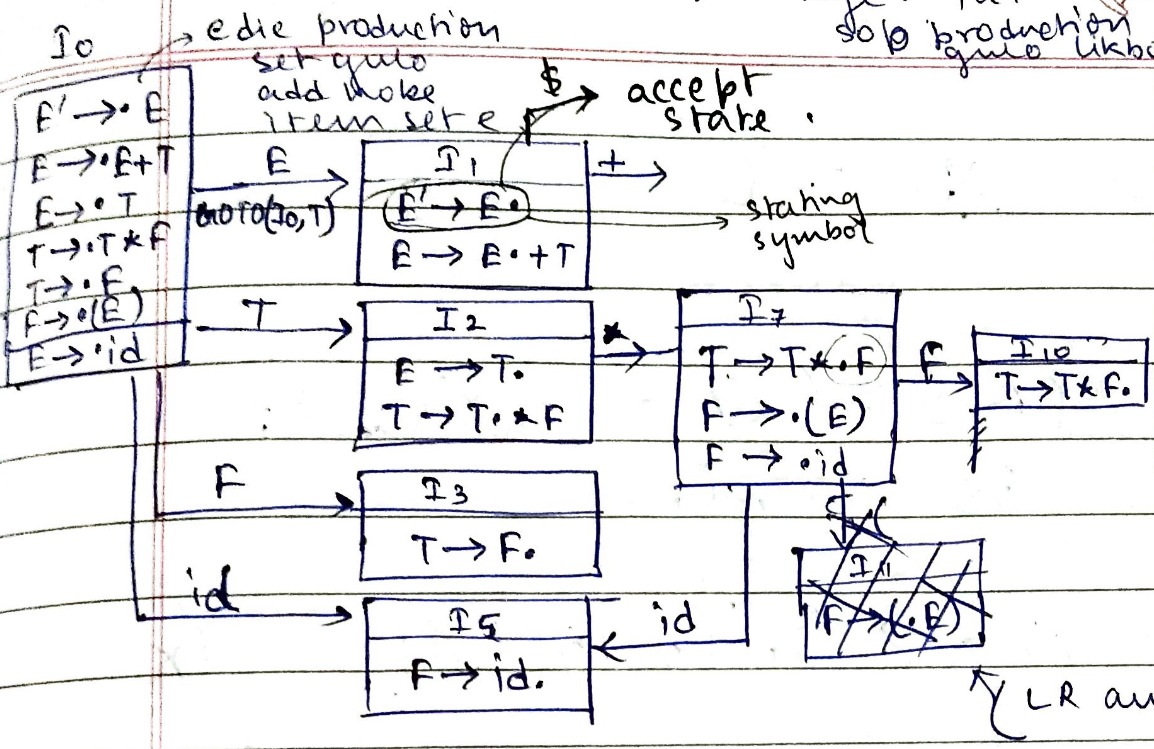
$T \rightarrow .T * F$, $T \rightarrow .F$, $F \rightarrow .(E)$, $F \rightarrow .id$

$E \to \alpha . B\beta$
$B \to \gamma$ → $B \to . \gamma$

[.F → terminal er age . tai solo production guno likbo]

GOTO(I, X) (I: item set, X: grammar symbol)

$A \to \alpha . X\beta$
↓
$A \to \alpha X . \beta$
(and add this production rule to new item set)



⊛ Parsing string id*id in SLR automata.

Item set	STACK	SYMBOL	INPUT	ACTION	Priority is given to reduction
(1)	0	$	id * id $	shift to I5	(shift).
(2)	05	$ id	* id $	reduce by F → id	if we cannot shift we reduce
(3)	03 (corresponding state of F)	$ F	* id $	reduce T → F	
(4)	02	$ T	* id $	shift to I7	
(5)	027	$ T *	id $	shift to I5	
(6)	0275	$ T * id	$	reduce F → id	
(7)	02710	$ T * F	$	reduce T → T*F	
(8)	02	$ T	$	reduce E → T	
(9)	01	$ E	$	accept	

• For parser generation we have automated tools such as YACC. (Yet Another compiler compiler)

FIRST(X) = { }

(i) FIRST(X) = {X} if X is terminal.

(ii) $X \rightarrow Y_1 Y_2 Y_3 \cdots Y_{i-1} Y_i \cdots Y_K$

FIRST(Y_i) → FIRST(X) if FIRST(Y_1), FIRST(Y_2), ... FIRST(Y_{i-1}) all of them contains 'ϵ'

'ϵ' is in FIRST(X) if all of FIRST(Y_x) $x \in [1, K]$ then 'ϵ' is in FIRST(X).

(iii) 'ϵ' is in FIRST(X) if $X \rightarrow \epsilon$

FOLLOW (X) :-

(i) '$' is in FOLLOW(x) if 'X' is starting symbol.

(ii) $A \rightarrow \alpha B \beta$

$\beta \longrightarrow$ strings.

FIRST(β) is in FOLLOW (B) except 'ϵ'.

(iii) $A \rightarrow \alpha B$

$A \rightarrow \alpha B \beta$ with ϵ in FIRST(β).

FOLLOW (A) is in FOLLOW (B)

$E \to TE'$

$E' \to +TE' \mid \epsilon$

$T \to FT'$

$T' \to *FT' \mid \epsilon$

$F \to (E) \mid id.$

$X \to Y_1 Y_2$

$FIRST(C) = \{C\}$

$A \to \alpha B$

$A \to \alpha B \beta$

$FIRST(E) = FIRST(T) = FIRST(F)$

$= \{C, id\}.$

$FIRST(E') = \{+, \epsilon\}.$

$FIRST(T') = \{*, \epsilon\}.$

$FOLLOW(E) = \{), \$\}$

$FOLLOW(E') = \{), \$\}$

$FOLLOW(T) = \{+,), \$\}$

$FOLLOW(T') = \{+,), \$\}$

$FOLLOW(F) = \{*, +,), \$\}.$

LL(1)

(M)

	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T						
T'						
F						

$M[A, a]$ — Terminal.

↓ Nonterminal.

Predictive Parsing Table.

For every $A \to \alpha$,

(i) Add $A \to \alpha$ to $M[A, a]$ for $a \in FIRST(\alpha)$

(ii) If 'ϵ' is in $FIRST(\alpha)$ then add '$A \to \alpha$' to $M[A, b]$ for every $b \in FOLLOW(A)$,

if '$\$$' is in $FOLLOW(A)$, add $A \to \alpha$ to $M[A, \$]$.