BCSE307L – COMPILER DESIGN

TEXT BOOK:

1. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, "Compilers: Principles, Techniques and Tools", Second Edition, Pearson Education Limited, 2014.

Module:2	SYNTAX ANALYSIS	8 hours
Role of Parser- Parse Tree - Elimination of Ambiguity – Top Down Parsing - Recursive Descent Parsing - LL (1) Grammars – Shift Reduce Parsers- Operator Precedence Parsing - LR Parsers, Construction of SLR Parser Tables and Parsing- CLR Parsing- LALR Parsing.		

Bottom Up Parsing

- Reduction
- Handle Pruning
- Shift-Reduce Parsing
- Operator Precedence
- LR Parser
  - SLR (Simple LR)
  - CLR (Canonical LR)
  - LALR (Lookahead LR)

LR Parsing : Simple LR (SLR)

LR(k)

L – left to right scanning of the input

R – construction a rightmost derivation in reverse

K – number of input symbols of lookahead that are used in marking parsing decisions.

- K =0 or k = 1, LR parser with k ≤ 1

LR Parsing

- LR parsers can be constructed to recognize all programming language constructs for which context-free grammars
- LR-parsing method is the most general non-backtracking shift-reduce parsing method
- LR parser can detect a syntactic error as soon as it is possible to do so on a left to right scan of the input
- The class of grammars that can be parsed using LR methods is a proper superset of the class of grammars

LR Parser

- Augmented Grammar
- Finding Canonical LR(0) items
- Finding First and Follow
- Parser Table
- Stack Implementation

1) Augmented Grammar

If G is a grammar with start Symbol S, then G' ,the augmented grammar for G, is G with a new start symbol S' and production S' → S

Example

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E)$

$F \rightarrow id$

1) Augmented Grammar

E' → E

E → E+T --- (1)

E → T --- (2)

T → T*F --- (3)

T → F ---(4)

F → (E) ---(5)

F → id ---(6)

2) Finding Canonical LR(0) items

Items and the LR(0) Automaton

- Closure of Item sets
- Function GOTO

Items and the LR(0) Automaton

- LR parser makes shit-reduce decisions by maintaining states to keep track of where we are in a parse
- States represent sets of “items”
- An LR(0) item of s grammar G is a production of G with a dot at some position of the body.
- Production A → XYZ yields the four items

$$
\begin{aligned}
A &\rightarrow \cdot XYZ \\
A &\rightarrow X \cdot YZ \\
A &\rightarrow XY \cdot Z \\
A &\rightarrow XYZ \cdot
\end{aligned}
$$

The production $A \rightarrow \epsilon$ generates only one item, $A \rightarrow \cdot$

Ex: $A \rightarrow aBb$ Possible LR(0) Items:

$A \rightarrow \bullet\, aBb$

$A \rightarrow a \bullet Bb$

$A \rightarrow aB \bullet b$

$A \rightarrow aBb \bullet$

Construction of Parse Tree

1.Construction of set of LR(0) items

2.Construction of PT using LR(0)

Using closure

Goto function

- Collection of set of LR(0) items-canonical collection of LR(0)

An **LR(0) item** of a grammar G is a production of G a dot at the some position of the right side.

Closure of Item Sets

If I is a set of items for a grammar G, then $\text{CLOSURE}(I)$ is the set of items constructed from I by the two rules:

1. Initially, add every item in I to $\text{CLOSURE}(I)$.

2. If $A \to \alpha \cdot B\beta$ is in $\text{CLOSURE}(I)$ and $B \to \gamma$ is a production, then add the item $B \to \cdot\gamma$ to $\text{CLOSURE}(I)$, if it is not already there. Apply this rule until no more new items can be added to $\text{CLOSURE}(I)$.

Function GOTO

The second useful function is $\text{GOTO}(I, X)$ where I is a set of items and X is a grammar symbol. $\text{GOTO}(I, X)$ is defined to be the closure of the set of all items $[A \to \alpha X \cdot \beta]$ such that $[A \to \alpha \cdot X \beta]$ is in I.

Canonical Collection of sets of LR(0)

void $items(G')$ {
    $C = \{\text{CLOSURE}(\{[S' \rightarrow \cdot S]\})\}$;
    repeat
        for (each set of items I in C)
            for (each grammar symbol X)
                if ($\text{GOTO}(I, X)$ is not empty and not in C)
                    add $\text{GOTO}(I, X)$ to C;
    until no new sets of items are added to C on a round;
}

Figure 4.33: Computation of the canonical collection of sets of LR(0) items

Canonical Collections of LR(0) items Example:

First consider the set of items I_0:

$$
\begin{aligned}
E' &\rightarrow \cdot E \\
E &\rightarrow \cdot E + T \\
E &\rightarrow \cdot T \\
T &\rightarrow \cdot T * F \\
T &\rightarrow \cdot F \\
F &\rightarrow \cdot (E) \\
F &\rightarrow \cdot \mathbf{id}
\end{aligned}
$$

I_1:

$$
\begin{aligned}
E' &\rightarrow E \cdot \\
E &\rightarrow E \cdot + T
\end{aligned}
$$

I_2:

$$
\begin{aligned}
E &\rightarrow T \cdot \\
T &\rightarrow T \cdot * F
\end{aligned}
$$

I_0

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$
$E \rightarrow \cdot T$
$T \rightarrow \cdot T * F$
$T \rightarrow \cdot F$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot \mathbf{id}$

I_1

$E' \rightarrow E \cdot$
$E \rightarrow E \cdot + T$

I_2

$E \rightarrow T \cdot$
$T \rightarrow T \cdot * F$

I_3

$T \rightarrow F \cdot$

I_4

$F \rightarrow (\cdot E)$

$E \rightarrow \cdot E + T$
$E \rightarrow \cdot T$
$T \rightarrow \cdot T * F$
$T \rightarrow \cdot F$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot \mathbf{id}$

I_5

$F \rightarrow \mathbf{id} \cdot$

I_6

$E \rightarrow E + \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (\ E \ )$

$F \rightarrow \cdot \mathbf{id}$

I_8

$E \rightarrow E \cdot + T$

$F \rightarrow (\ E \cdot \ )$

I_7

$T \rightarrow T * \cdot F$

$F \rightarrow \cdot (\ E \ )$

$F \rightarrow \cdot \mathbf{id}$

$$I_9$$

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

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

$$I_{11}$$

$$F \rightarrow (E) \cdot$$

$$I_{10}$$

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

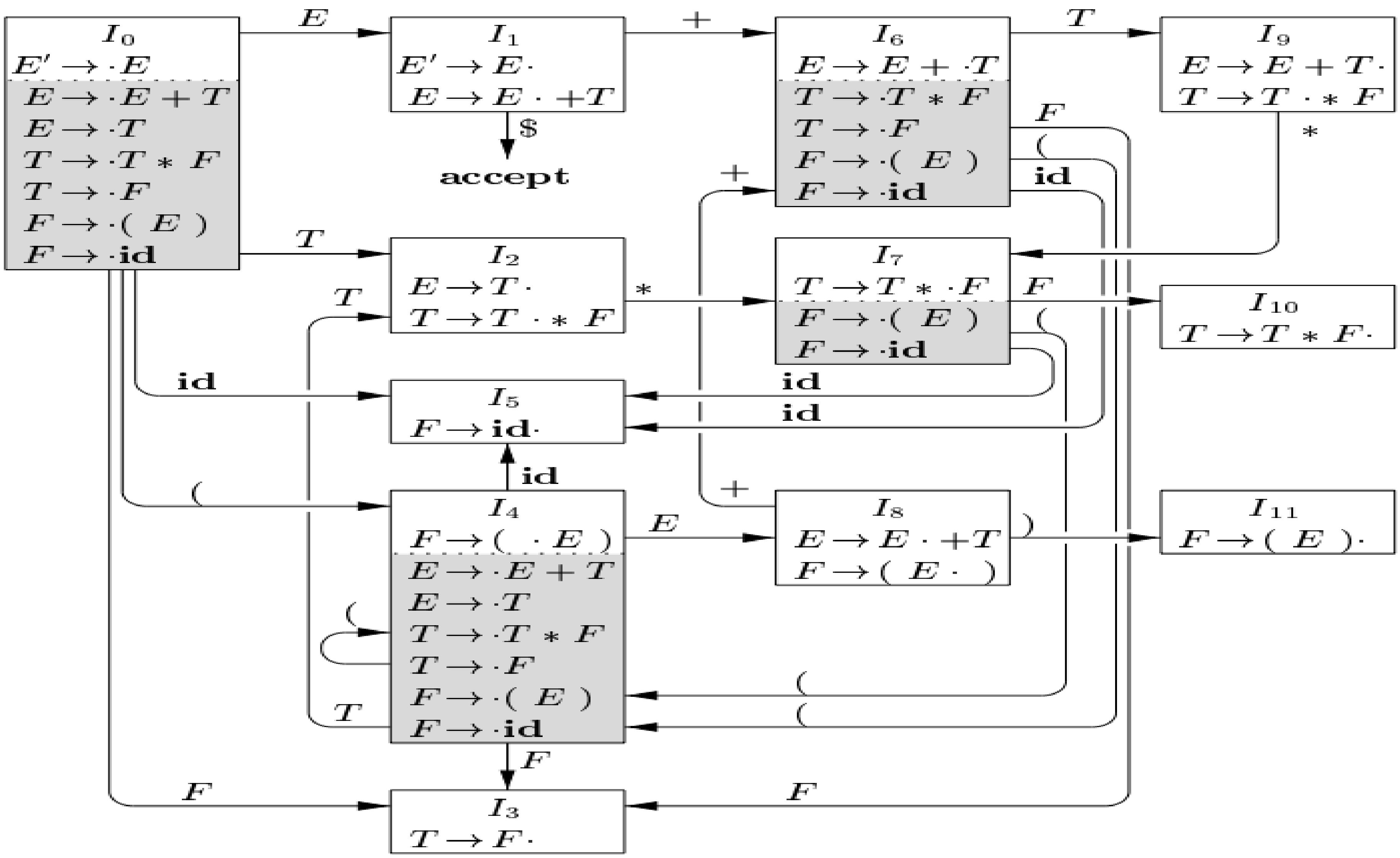


Figure 4.31: LR(0) automaton for the expression grammar (4.1)

3) FIRST and FOLLOW

FIRST(X)

Rules:

1. X is a Terminal, FIRST (X) = { X }
2. X is a Non-terminal, $X \rightarrow Y_1, Y_2, Y_3 \ldots . Y_k$, $K \geq 1$,
   FIRST(X) = FIRST(Y_1)
3. $X \rightarrow \varepsilon$ is a production, FIRST(X) = { ε }

FOLLOW (X)

Rules:

1. S is a Start Symbol, FOLLOW(S) = \$
2. If a production A → α B β , FOLLOW(B) = FIRST(β) – ε
3. If a production A → α B or A → α B β i.e, FIRST(β) = ε

   FOLLOW (B) = FOLLOW(A)

FIRST and FOLLOW

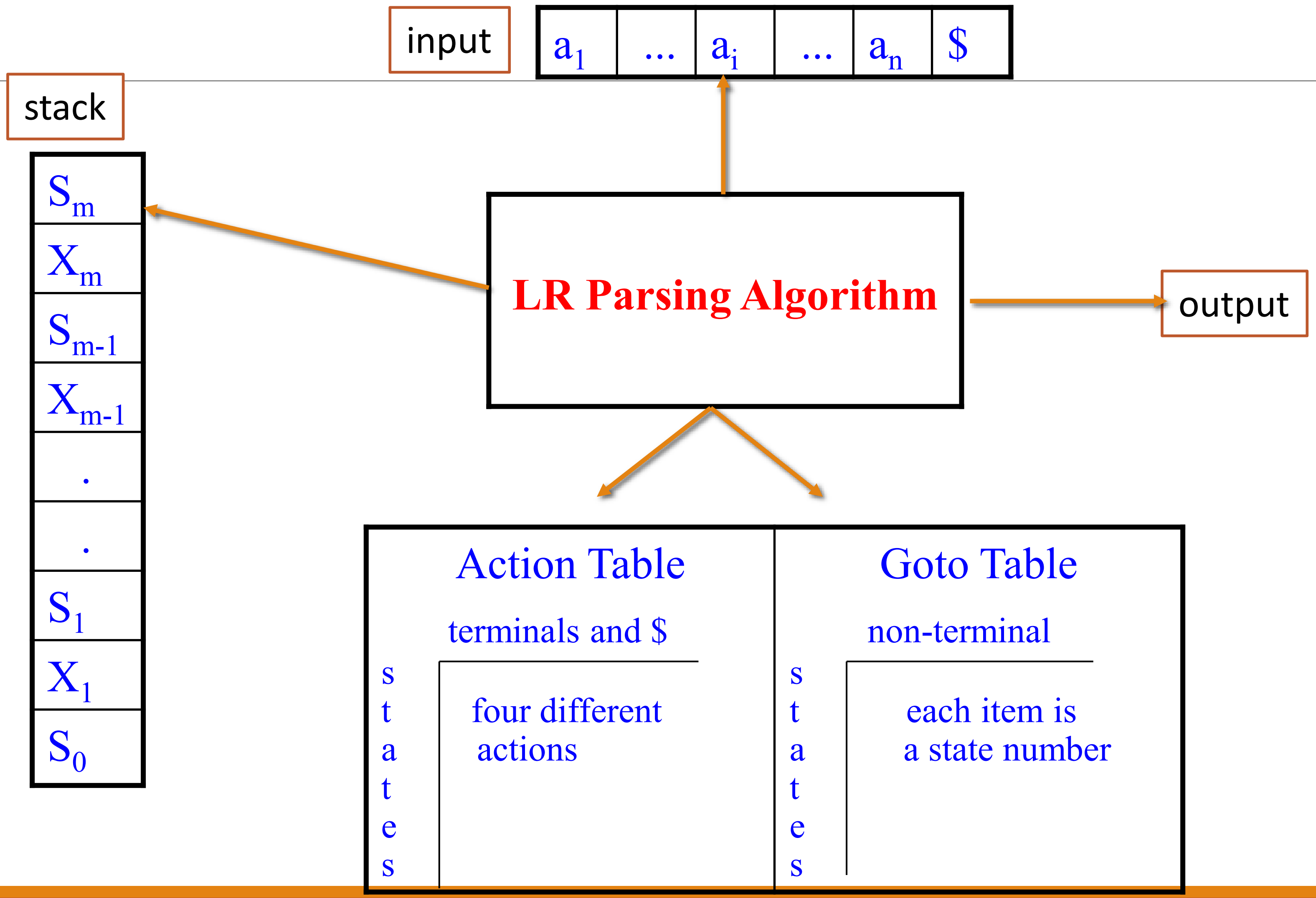
Non-terminal	FIRST	FOLLOW
E	(, id	+,) , $
T	(, id	+, * ,) , $
F	(, id	+, *,) , $

4) LR Parsing

LR Parsing Algorithm

- Structure of the LR Parsing Table
- LR Parser Configurations
- Behavior of the LR Parser

LR Parsing Algorithm



Structure of the LR Parsing Table

The parsing table consists of two parts: a parsing-action function ACTION and a goto function GOTO.

1. The ACTION function takes as arguments a state i and a terminal a (or \$, the input endmarker). The value of ACTION$[i, a]$ can have one of four forms:

   (a) Shift j, where j is a state. The action taken by the parser effectively shifts input a to the stack, but uses state j to represent a.

   (b) Reduce $A \to \beta$. The action of the parser effectively reduces β on the top of the stack to head A.

   (c) Accept. The parser accepts the input and finishes parsing.

   (d) Error. The parser discovers an error in its input and takes some corrective action.

2. We extend the GOTO function, defined on sets of items, to states: if GOTO$[I_i, A] = I_j$, then GOTO also maps a state i and a nonterminal A to state j.

LR Parser Configurations

A Configuration of an LR parser is a pair

$$(s_0 s_1 \cdots s_m,\ a_i a_{i+1} \cdots a_n \$)$$

This configuration represents the right –sentential form

$$X_1 X_2 \cdots X_m a_i a_{i+1} \cdots a_n$$

Behavior of the LR Parser

The entry $\text{ACTION}[s_m, a_i]$ in the parsing action table. The configurations resulting after each of the four types of move are as follows

1. If $\text{ACTION}[s_m, a_i] = \text{shift } s$, the parser executes a shift move; it shifts the next state s onto the stack, entering the configuration

$$(s_0 s_1 \cdots s_m s,\ a_{i+1} \cdots a_n \$)$$

2. If $\text{ACTION}[s_m, a_i] = \text{reduce } A \rightarrow \beta$, then the parser executes a reduce move, entering the configuration

$$(s_0 s_1 \cdots s_{m-r} s,\ a_i a_{i+1} \cdots a_n \$)$$

where r is the length of β, and $s = \text{GOTO}[s_{m-r}, A]$.

Behavior of the LR Parser

3. If $\text{ACTION}[s_m, a_i]$ = accept, parsing is completed.

4. If $\text{ACTION}[s_m, a_i]$ = error, the parser has discovered an error and calls an error recovery routine.

LR Parsing Algorithm

Algorithm 4.44: LR-parsing algorithm.

INPUT: An input string w and an LR-parsing table with functions ACTION and GOTO for a grammar G.

OUTPUT: If w is in $L(G)$, the reduction steps of a bottom-up parse for w; otherwise, an error indication.

METHOD: Initially, the parser has s_0 on its stack, where s_0 is the initial state, and $w\$$ in the input buffer. The parser then executes the program in Fig. 4.36.

LR Parsing Algorithm

```
let a be the first symbol of w$;
while(1) { /* repeat forever */
        let s be the state on top of the stack;
        if ( ACTION[s, a] = shift t ) {
                push t onto the stack;
                let a be the next input symbol;
        } else if ( ACTION[s, a] = reduce A → β ) {
                pop |β| symbols off the stack;
                let state t now be on top of the stack;
                push GOTO[t, A] onto the stack;
                output the production A → β;
        } else if ( ACTION[s, a] = accept ) break; /* parsing is done */
        else call error-recovery routine;
}
```

ACTION and GOTO functions of an LR-parsing table for the expression grammar.

(1)	$E \to E + T$	(4)	$T \to F$
(2)	$E \to T$	(5)	$F \to (E)$
(3)	$T \to T * F$	(6)	$F \to \mathbf{id}$

The codes for the actions are:

1. si means shift and stack state i,
2. rj means reduce by the production numbered j,
3. acc means accept,
4. blank means error.

Constructing SLR-Parsing Table

Algorithm 4.46 : Constructing an SLR-parsing table.

INPUT: An augmented grammar G'.

OUTPUT: The SLR-parsing table functions ACTION and GOTO for G'.

METHOD:

1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(0) items for G'.

2. State i is constructed from I_i. The parsing actions for state i are determined as follows:

   (a) If $[A \rightarrow \alpha \cdot a \beta]$ is in I_i and GOTO$(I_i, a) = I_j$, then set ACTION$[i, a]$ to "shift j." Here a must be a terminal.

   (b) If $[A \rightarrow \alpha \cdot]$ is in I_i, then set ACTION$[i, a]$ to "reduce $A \rightarrow \alpha$" for all a in FOLLOW(A); here A may not be S'.

   (c) If $[S' \rightarrow S \cdot]$ is in I_i, then set ACTION$[i, \$]$ to "accept."

   If any conflicting actions result from the above rules, we say the grammar is not SLR(1). The algorithm fails to produce a parser in this case.

Constructing SLR-Parsing Table

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If $\text{GOTO}(I_i, A) = I_j$, then $\text{GOTO}[i, A] = j$.

4. All entries not defined by rules (2) and (3) are made "error."

5. The initial state of the parser is the one constructed from the set of items containing $[S' \to \cdot S]$.

Parsing Table

STATE	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

5) Stack Implementation

	STACK	SYMBOLS	INPUT	ACTION
(1)	0		$\mathbf{id} * \mathbf{id} + \mathbf{id}\,\$$	shift
(2)	0 5	$\mathbf{id}$	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(3)	0 3	F	$* \mathbf{id} + \mathbf{id}\,\$$	reduce by $T \to F$
(4)	0 2	T	$* \mathbf{id} + \mathbf{id}\,\$$	shift
(5)	0 2 7	$T *$	$\mathbf{id} + \mathbf{id}\,\$$	shift
(6)	0 2 7 5	$T * \mathbf{id}$	$+ \mathbf{id}\,\$$	reduce by $F \to \mathbf{id}$
(7)	0 2 7 10	$T * F$	$+ \mathbf{id}\,\$$	reduce by $T \to T * F$
(8)	0 2	T	$+ \mathbf{id}\,\$$	reduce by $E \to T$
(9)	0 1	E	$+ \mathbf{id}\,\$$	shift
(10)	0 1 6	$E +$	$\mathbf{id}\,\$$	shift
(11)	0 1 6 5	$E + \mathbf{id}$	$\$$	reduce by $F \to \mathbf{id}$
(12)	0 1 6 3	$E + F$	$\$$	reduce by $T \to F$
(13)	0 1 6 9	$E + T$	$\$$	reduce by $E \to E + T$
(14)	0 1	E	$\$$	accept

Figure 4.38: Moves of an LR parser on $\mathbf{id} * \mathbf{id} + \mathbf{id}$

Viable Prefixes

The prefixes of right sentential forms that can appear on the stack of shift-reduce parser are called viable prefixes.

Item $A \rightarrow \beta_1 \beta_2$ is valid prefix $\alpha\ \beta_1$ if there is a derivations

$$S' \underset{rm}{\Rightarrow} \alpha\ Aw \underset{rm}{\Rightarrow} \alpha\ \beta_1 \beta_2\ w$$

More Powerful LR-Parsers

Canonical-LR (CLR)

- Makes full use of the lookahed symbols
- Large set of items called LR(1) items

Lookahead-LR (LALR)

- Based on LR(0) set of items and fewer states based on the LR(1) items

Canonical LR(1) Items

Formally, we say LR(1) item $[A \rightarrow \alpha{\cdot}\beta, a]$ is *valid* for a viable prefix γ if there is a derivation $S \underset{rm}{\overset{*}{\Rightarrow}} \delta A w \underset{rm}{\Rightarrow} \delta\alpha\beta w$, where

1. $\gamma = \delta\alpha$, and
2. Either a is the first symbol of w, or w is ϵ and a is \$.

Canonical LR(1) Items

```
SetOfItems CLOSURE(I) {
    repeat
        for ( each item [A → α·Bβ, a] in I )
            for ( each production B → γ in G′ )
                for ( each terminal b in FIRST(βa) )
                    add [B → ·γ, b] to set I;
    until no more items are added to I;
    return I;
}

SetOfItems GOTO(I, X) {
    initialize J to be the empty set;
    for ( each item [A → α·Xβ, a] in I )
        add item [A → αX·β, a] to set J;
    return CLOSURE(J);
}

void items(G′) {
    initialize C to {CLOSURE({[S′ → ·S, $]})};
    repeat
        for ( each set of items I in C )
            for ( each grammar symbol X )
                if ( GOTO(I, X) is not empty and not in C )
                    add GOTO(I, X) to C;
    until no new sets of items are added to C;
}
```

CLR – Example 2

Example 4.54: Consider the following augmented grammar.

$$
\begin{array}{rcl}
S' & \rightarrow & S \\
S & \rightarrow & C\ C \\
C & \rightarrow & c\ C \ \mid \ d
\end{array}
\tag{4.55}
$$

CLR – Example 2

$$
\begin{array}{ll}
I_0: & S \to \cdot S, \$ \\
 & S \to \cdot CC,\ \$ \\
 & C \to \cdot cC,\ c/d \\
 & C \to \cdot d,\ c/d
\end{array}
$$

The brackets have been omitted for notational convenience, and we use the notation $[C \to \cdot cC,\ c/d]$ as a shorthand for the two items $[C \to \cdot cC,\ c]$ and $[C \to \cdot cC,\ d]$.

Now we compute $\text{GOTO}(I_0, X)$ for the various values of X. For $X = S$ we must close the item $[S' \to S\cdot,\ \$]$. No additional closure is possible, since the dot is at the right end. Thus we have the next set of items

$$
I_1: \quad S' \to S\cdot,\ \$
$$

For $X = C$ we close $[S \to C\cdot C,\ \$]$. We add the C-productions with second component \$ and then can add no more, yielding

$$
\begin{array}{ll}
I_2: & S \to C\cdot C,\ \$ \\
 & C \to \cdot cC,\ \$ \\
 & C \to \cdot d,\ \$
\end{array}
$$

Next, let $X = c$. We must close $\{[C \to c\cdot C,\ c/d]\}$. We add the C-productions with second component c/d, yielding

CLR – Example 2

$$
\begin{aligned}
I_0: \quad & S \to \cdot S, \$ \\
& S \to \cdot CC,\ \$ \\
& C \to \cdot cC,\ c/d \\
& C \to \cdot d,\ c/d
\end{aligned}
$$

The brackets have been omitted for notational convenience, and we use the notation $[C \to \cdot cC,\ c/d]$ as a shorthand for the two items $[C \to \cdot cC,\ c]$ and $[C \to \cdot cC,\ d]$.

Now we compute $\text{GOTO}(I_0, X)$ for the various values of X. For $X = S$ we must close the item $[S' \to S\cdot,\ \$]$. No additional closure is possible, since the dot is at the right end. Thus we have the next set of items

$$
I_1: \quad S' \to S\cdot,\ \$
$$

For $X = C$ we close $[S \to C{\cdot}C,\ \$]$. We add the C-productions with second component \$ and then can add no more, yielding

$$
\begin{aligned}
I_2: \quad & S \to C{\cdot}C,\ \$ \\
& C \to \cdot cC,\ \$ \\
& C \to \cdot d,\ \$
\end{aligned}
$$

Next, let $X = c$. We must close $\{[C \to c{\cdot}C,\ c/d]\}$. We add the C-productions with second component c/d, yielding

CLR – Example 2

$$
\begin{aligned}
I_3: \quad & C \rightarrow c{\cdot}C,\ c/d \\
& C \rightarrow {\cdot}cC,\ c/d \\
& C \rightarrow {\cdot}d,\ c/d
\end{aligned}
$$

Finally, let $X = d$, and we wind up with the set of items

$$I_4: \quad C \rightarrow d{\cdot},\ c/d$$

We have finished considering GOTO on I_0. We get no new sets from I_1, but I_2 has goto's on C, c, and d. For GOTO(I_2, C) we get

$$I_5: \quad S \rightarrow CC{\cdot}, \$$$

no closure being needed. To compute GOTO(I_2, c) we take the closure of $\{[C \rightarrow c{\cdot}C,\ \$]\}$, to obtain

$$
\begin{aligned}
I_6: \quad & C \rightarrow c{\cdot}C,\ \$ \\
& C \rightarrow {\cdot}cC,\ \$ \\
& C \rightarrow {\cdot}d,\ \$
\end{aligned}
$$

CLR – Example 2

Continuing with the GOTO function for I_2, GOTO(I_2, d) is seen to be

$$I_7: \quad C \to d\cdot,\ \$$$

Turning now to I_3, the GOTO's of I_3 on c and d are I_3 and I_4, respectively, and GOTO(I_3, C) is

$$I_8: \quad C \to cC\cdot,\ c/d$$

I_4 and I_5 have no GOTO's, since all items have their dots at the right end. The GOTO's of I_6 on c and d are I_6 and I_7, respectively, and GOTO(I_6, C) is

$$I_9: \quad C \to cC\cdot,\ \$$$

CLR – Example 2

State	Action			Goto	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

Figure 4.42: Canonical parsing table for grammar (4.55)

LALR – Example 2

Consider the following augmented grammar.

$$
\begin{array}{rcl}
S' & \rightarrow & S \\
S & \rightarrow & C\,C \\
C & \rightarrow & c\,C \mid d
\end{array}
$$

LALR – Example 2

Example 4.60 : Again consider grammar (4.55) whose GOTO graph was shown in Fig. 4.41. As we mentioned, there are three pairs of sets of items that can be merged. I_3 and I_6 are replaced by their union:

$$\begin{array}{ll} I_{36}: & C \rightarrow c{\cdot}C,\ c/d/\$ \\ & C \rightarrow {\cdot}cC,\ c/d/\$ \\ & C \rightarrow {\cdot}d,\ c/d/\$ \end{array}$$

I_4 and I_7 are replaced by their union:

$$I_{47}: \quad C \rightarrow d{\cdot},\ c/d/\$$$

and I_8 and I_9 are replaced by their union:

$$I_{89}: \quad C \rightarrow cC{\cdot},\ c/d/\$$$

LALR – Example 2

The LALR action and goto functions for the condensed sets of items are shown in Fig. 4.43.

STATE	ACTION			GOTO	
	c	d	\$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

Figure 4.43: LALR parsing table for the grammar of Example 4.54

Thank You