Syntactic Analysis (Parsing)

José Miguel Rivero

rivero@cs.upc.edu

Barcelona School of Informatics (FIB)

Universitat Politècnica de Catalunya BarcelonaTech (UPC)

Summary

- Methods of Linear Parsing
  - Top-down Parsers (LL(1))
  - Bottom-up Parsers (LR(1))
- Types of Top-down Parsers
  - Table Driven Parsers (iterative)
  - Recursive Predictive Parsers
- Example of Recursive Parser (ANTLR style)
- Recursive Predictive Parser Generation
- Bottom-up Parsers
  - Introduction
  - Example of Bottom-up Parsing
  - SLR(1) Table Construction

Methods of Linear Parsing

The list of tokens will be traversed *left-to-right*. Decisions to proceed take into account one token of lookahead.

- Top-down parsers (LL(1))
  - Build the AST from the root to the leaves (top-down)
  - Follow a left-most derivation in forward direction
  - More intuitive: can be *manually* written
  - **Cannot use left-recursion, and need left-factoring**
- Bottom-up parsers (LR1))
  - Build the AST from the leaves to the root (bottom-up)
  - Follow a right-most derivation in *backward* direction
  - Less intuitive than top-down parsers
  - Slightly more powerful

FIB

Types of Top-down Parsers

- Table Driven Parsers (iterative)
  - Parsing algorithm is fixed, driven by a decision table
  - Table M is built from the grammar G.
    Empty boxes correspond to syntax errors

M	a_1	$\ldots$	a	$\ldots$	a_n	\$
A_1						
$\vdots$						
A			$A \to \alpha_k$			
$\vdots$						
A_m						

FIB

Types of Top-down Parsers

- Table Driven Parsers (iterative)
  - Parsing algorithm is fixed, driven by a decision table
  - Table M is built from the grammar G.
    Empty boxes correspond to syntax errors
- Recursive Predictive Parsers
  - Parsing algorithm is formed by a set of mutually recursive functions
  - Each rule $A \rightarrow \alpha$ generates the code of its function

```
void A(void) {
    // Code generated from α
}
```

  - `Gencode` describes how to translate a rule to the associated function

Example of Recursive Parser (ANTLR)

Simple grammar in ANTLR:

```
instruction_list : ( instruction )*
                 ;
instruction : IDENT ASSIG expr
            | IF expr THEN instruction_list
            ;
expr : (IDENT | NUM) ( PLUS (IDENT | NUM) )*
     ;
```

Example of Recursive Parser (ANTLR)

Simple grammar in ANTLR:

```
instruction_list :  ( instruction )*
                 ;
instruction :  IDENT  ASSIG  expr
            |  IF  expr  THEN  instruction_list
            ;
expr :  expr_simple  ( PLUS  expr_simple )*
     ;
expr_simple :  IDENT
            |  NUM
            ;
```

Example of Recursive Parser (ANTLR)

- Production rule

  expr : (IDENT | NUM) (PLUS (IDENT | NUM))* ;

- Parser *by hand*

```
void expr ( ) {
    if ( token == IDENT || token == NUM ) {
        token = nextToken( );
        while ( token == PLUS ) {
            token = nextToken( );
            if ( token == IDENT || token == NUM ) {
                token = nextToken( );
            } else syntaxError()
        }
    } else syntaxError()
}
```

Example of Recursive Parser (ANTLR)

- Production rule

```
instruction_list :  ( instruction ) *
                 ;
```

- Parser

```
void instruction_list ( ) {
    while ( token == IDENT || token == IF ) {
        instruction( );
    }
}
```

Example of Recursive Parser (ANTLR)

- Production rule

```
instruction :  IDENT  ASSIG  expr
            |  IF  expr  THEN  instruction_list
            ;
```

- Parser

```
void instruction ( ) {
    if ( token == IDENT ) {
        MATCH(IDENT);  MATCH(ASSIG);  expr( );
    } else if ( token == IF ) {
        MATCH(IF);  expr( );  MATCH(THEN);  instruction_list( );
    } else syntaxError()
}
```

Example of Recursive Parser (ANTLR)

- Production rule

```
expr :  expr_simple  ( PLUS  expr_simple )*  ;
```

- Parser

```
void  expr ( ) {
        expr_simple( );
        while  ( token == PLUS ) {
            MATCH(PLUS);
            expr_simple( );
        }
}
```

Recursive Predictive Parsers Generation

- Firstly check that the grammar is LL(1), building the table $M[A, a]$ without conflicts.
- $Genrule(\ A \rightarrow \alpha\ )$ generates the code of a function A associated to the production rule
- $Gencode(\ e\ )$ generates the code that recognizes in the input an expression e

$Genrule(\ A \rightarrow \alpha\ ) \equiv$

```
void A ( void ) {
        /* Gencode( α ) */
}
```

Recursive Predictive Parsers Generation

$Gencode(\ e_1 \,|\, e_2 \,|\, \ldots \,|\, e_n\ ) \equiv$

`if (token` $\in first(e_1)$ `) {`

$\quad /* \ Gencode(\ e_1\ ) \ */$

`} else if (token` $\in first(e_2)$ `) {`

$\quad /* \ Gencode(\ e_2\ ) \ */$

$\ldots$

`} else if (token` $\in first(e_n)$ `) {`

$\quad /* \ Gencode(\ e_n\ ) \ */$

`} else` $syntaxError()$; `// if` $\nexists\, i : 1 \leq i \leq n : nullable?(e_i)$

FIB

Recursive Predictive Parsers Generation

$Gencode(\ e_1 \,|\, e_2 \,|\, \ldots \,|\, e_n\ ) \equiv$

`if (token` $\in first(e_1)$ `) {`

$\quad /* \ Gencode(\ e_1\ ) \ */$

`} else if (token` $\in first(e_2)$ `) {`

$\quad /* \ Gencode(\ e_2\ ) \ */$

$\ldots$

`} else if (token` $\in first(e_n)$ `) {`

$\quad /* \ Gencode(\ e_n\ ) \ */$

`}` `// if` $\exists\, i : 1 \leq i \leq n : nullable?(e_i)$

FIB

Recursive Predictive Parsers Generation

$Gencode(\ e_1\ e_2\ \ldots\ e_n\ ) \equiv$

$/*\ Gencode(\ e_1\ )\ */$

$/*\ Gencode(\ e_2\ )\ */$

$\ldots$

$/*\ Gencode(\ e_n\ )\ */$

Recursive Predictive Parsers Generation

$Gencode(\ e_1^*\ ) \equiv$

`while` (`token` $\in first(e_1)$) {

$/*\ Gencode(\ e_1\ )\ */$

}

$Gencode(\ e_1^+\ ) \equiv$

`do` {

$/*\ Gencode(\ e_1\ )\ */$

} `while` (`token` $\in first(e_1)$);

$Gencode(\ e_1?\ ) \equiv$

`if` (`token` $\in first(e_1)$ {

$/*\ Gencode(\ e_1\ )\ */$

}

$Gencode(\ \epsilon\ ) \equiv$

; // do nothing

Recursive Predictive Parsers Generation

$Gencode(A) \equiv$ `// for a non-terminal` A

$A();$

$Gencode(a) \equiv$ `// for a terminal` a

$MATCH(a);$

`Where` $MATCH(a)$ `is defined as follows:`

```
if (token == a) {
        token = nextToken();
} else syntaxError();
```

Bottom-up LR(1) Parsers

- Characteristics
- Example of Bottom-up Parsing
- Shift-Reduce Parsing Algorithm
- Viable Prefixes. LR(0) DFA
- **action** and **goto** Tables Construction
- Shift/reduce and reduce/reduce conflicts
- Types of Bottom-up Parsing
  - SLR(1)
  - LR(1)
  - LALR(1)

Example of Bottom-up Parsing

$$
\begin{array}{rcl}
E & \rightarrow & E + T \\
 & | & T \\
T & \rightarrow & T * F \\
 & | & F \\
F & \rightarrow & (E) \\
 & | & \texttt{id}
\end{array}
$$

$w = \texttt{id}_1 + \texttt{id}_2 * \texttt{id}_3$

Example of Bottom-up Parsing

$$
\begin{array}{rcl}
E & \rightarrow & E + T \\
 & | & T \\
T & \rightarrow & T * F \\
 & | & F \\
F & \rightarrow & (E) \\
 & | & \texttt{id}
\end{array}
$$

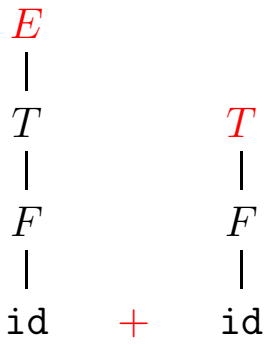
$shift$ $\texttt{id}_1$

$\boxed{\texttt{id}_1}$ $+$ $\texttt{id}_2$ $*$ $\texttt{id}_3$ $\$$

$E \Rightarrow_{rd} E+T \Rightarrow_{rd} E+T*F \Rightarrow_{rd} E+T*id_3$
$\Rightarrow_{rd} E+F*id_3 \Rightarrow_{rd} E+id_2*id_3 \Rightarrow_{rd} T+id_2*id_3$
$\Rightarrow_{rd} F+id_2*id_3 \Rightarrow_{rd} id_1+id_2*id_3$

Example of Bottom-up Parsing

$$
\begin{array}{lll}
E \rightarrow & E + T \\
\mid & T \\
T \rightarrow & T * F \\
\mid & F \\
F \rightarrow & (\, E \,) \\
\mid & \texttt{id}
\end{array}
$$



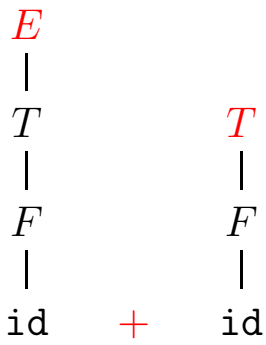
$shift$ $*$

$\texttt{id}_1 \quad + \quad \texttt{id}_2 \quad \boxed{*} \quad \texttt{id}_3 \quad \$$

$E \Rightarrow_{rd} E+T \Rightarrow_{rd} E+T*F \Rightarrow_{rd} \boxed{E+T} * id_3$
$\Rightarrow_{rd} E+F*id_3 \Rightarrow_{rd} E+id_2*id_3 \Rightarrow_{rd} T+id_2*id_3$
$\Rightarrow_{rd} F+id_2*id_3 \Rightarrow_{rd} id_1+id_2*id_3$

Example of Bottom-up Parsing

$$
\begin{array}{lll}
E \rightarrow & E + T \\
\mid & T \\
T \rightarrow & T * F \\
\mid & F \\
F \rightarrow & (\, E \,) \\
\mid & \texttt{id}
\end{array}
$$



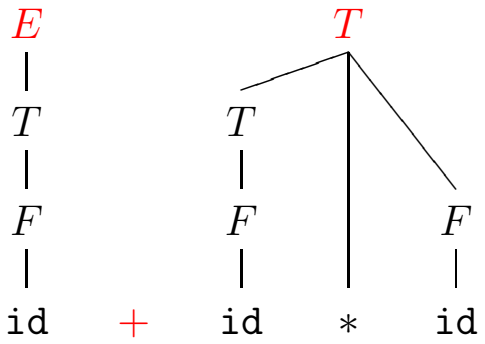
$shift$ $\texttt{id}_3$

$\texttt{id}_1 \quad + \quad \texttt{id}_2 \quad * \quad \boxed{\texttt{id}_3} \quad \$$

$E \Rightarrow_{rd} E+T \Rightarrow_{rd} E+T*F \Rightarrow_{rd} \boxed{E+T*} id_3$
$\Rightarrow_{rd} E+F*id_3 \Rightarrow_{rd} E+id_2*id_3 \Rightarrow_{rd} T+id_2*id_3$
$\Rightarrow_{rd} F+id_2*id_3 \Rightarrow_{rd} id_1+id_2*id_3$

Example of Bottom-up Parsing

$$
\begin{array}{rcl}
E & \rightarrow & E + T \\
 & | & T \\
T & \rightarrow & T * F \\
 & | & F \\
F & \rightarrow & (\, E \,) \\
 & | & \texttt{id}
\end{array}
$$

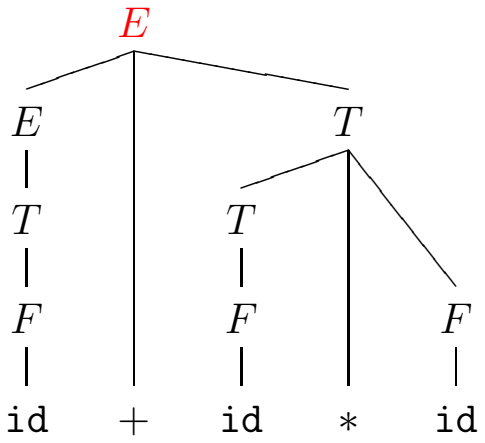


$reduce\ \ with\ \ E \rightarrow E + T$

$$
\begin{aligned}
E \ &\Rightarrow_{rd} \boxed{E + T} \ \Rightarrow_{rd} \ E + T * F \ \Rightarrow_{rd} \ E + T * id_3 \\
&\Rightarrow_{rd} \ E + F * id_3 \ \Rightarrow_{rd} \ E + id_2 * id_3 \ \Rightarrow_{rd} \ T + id_2 * id_3 \\
&\Rightarrow_{rd} \ F + id_2 * id_3 \ \Rightarrow_{rd} \ id_1 + id_2 * id_3
\end{aligned}
$$

Example of Bottom-up Parsing

$$
\begin{array}{rcl}
E & \rightarrow & E + T \\
 & | & T \\
T & \rightarrow & T * F \\
 & | & F \\
F & \rightarrow & (\, E \,) \\
 & | & \texttt{id}
\end{array}
$$



$accept$

$$
\begin{aligned}
&\boxed{E} \ \Rightarrow_{rd} \ E + T \ \Rightarrow_{rd} \ E + T * F \ \Rightarrow_{rd} \ E + T * id_3 \\
&\Rightarrow_{rd} \ E + F * id_3 \ \Rightarrow_{rd} \ E + id_2 * id_3 \ \Rightarrow_{rd} \ T + id_2 * id_3 \\
&\Rightarrow_{rd} \ F + id_2 * id_3 \ \Rightarrow_{rd} \ id_1 + id_2 * id_3
\end{aligned}
$$

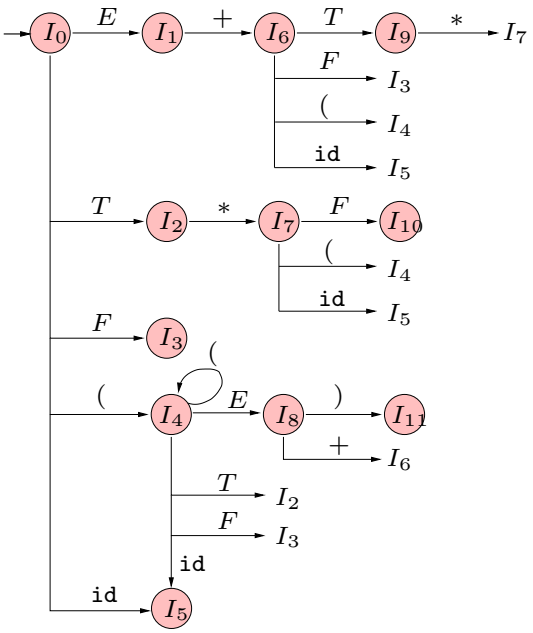
Example of Bottom-up Parsing (cont.)

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow \texttt{id}$

$\texttt{id}_1 + \texttt{id}_2 * \texttt{id}_3$

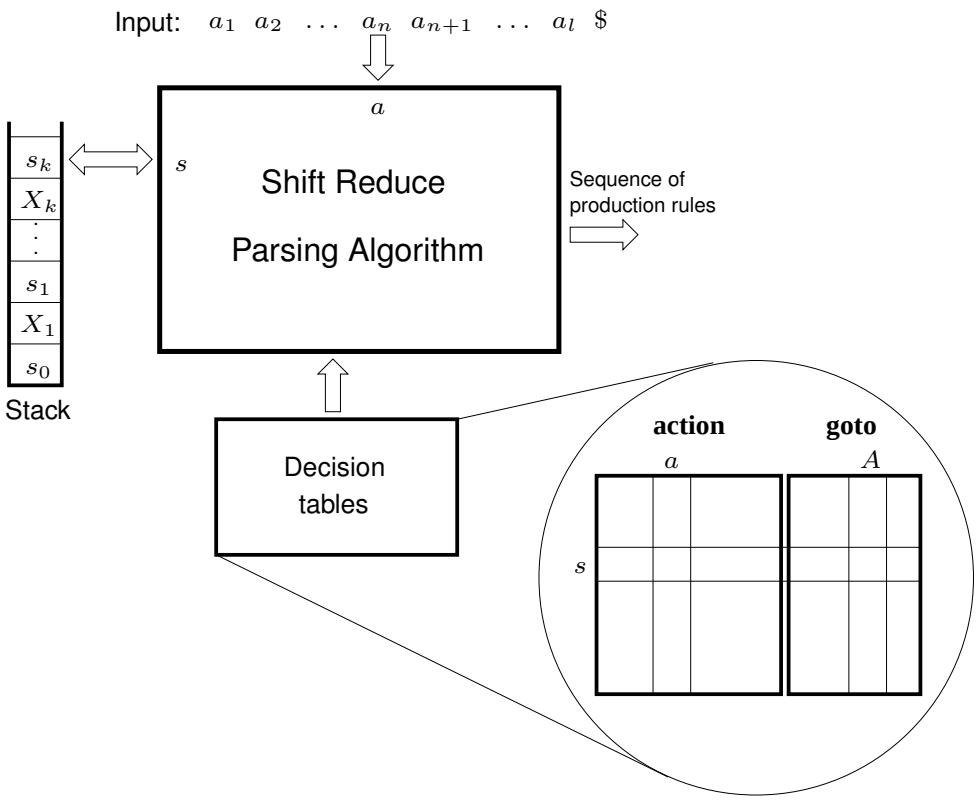
Step	Stack	Input	Action
(1)		$\boxed{id_1} + id_2 * id_3$ \$	shift $\texttt{id}_1$
(2)	id_1	$\boxed{+} id_2 * id_3$ \$	reduce with 6. $F \rightarrow id$
(3)	F	$\boxed{+} id_2 * id_3$ \$	reduce with 4. $T \rightarrow F$
(4)	T	$\boxed{+} id_2 * id_3$ \$	reduce with 2. $E \rightarrow T$
(5)	E	$\boxed{+} id_2 * id_3$ \$	shift $+$
(6)	$E +$	$\boxed{id_2} * id_3$ \$	shift $\texttt{id}_2$
(7)	$E + id_2$	$\boxed{*} id_3$ \$	reduce with 6. $F \rightarrow id$
(8)	$E + F$	$\boxed{*} id_3$ \$	reduce with 4. $T \rightarrow F$
(9)	$E + T$	$\boxed{*} id_3$ \$	shift $*$
(10)	$E + T *$	$\boxed{id_3}$ \$	shift $\texttt{id}_3$
(11)	$E + T * id_3$	$\boxed{\$}$	reduce with 6. $F \rightarrow id$
(12)	$E + T * F$	$\boxed{\$}$	reduce with 3. $T \rightarrow T * F$
(13)	$E + T$	$\boxed{\$}$	reduce with 1. $E \rightarrow E + T$
(14)	E	$\boxed{\$}$	accept

Viable Prefixes. LR(0) DFA



Step	Stack	Input	Action
(10)	$0\ E\ 1 + 6\ T\ 9 * 7$	$\boxed{id_3}$ \$	shift $\texttt{id}_3$ (**s5**)

Shift / Reduce Parsing Algorithm



Shift / Reduce Parsing Algorithm

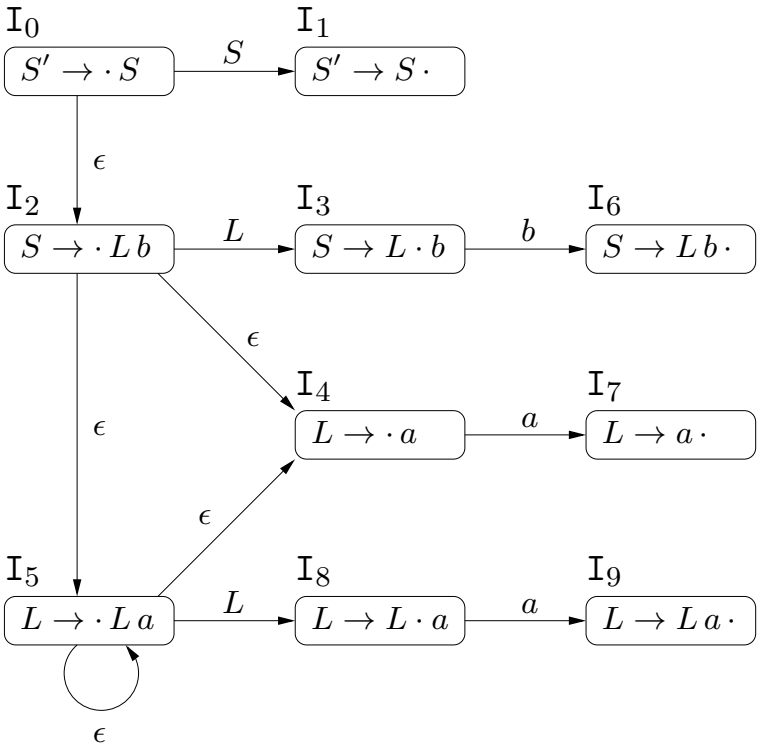
$Stk := EmptyStack();\ PushStack(Stk, 0);$ // *Initial state 0*
$a := FirstToken();$
`loop`
  $s := TopStack(Stk);$ // *Current state*
  `if` $action[s, a] =$ `si` `then` // *Shift and go to state i*
    $PushStack(Stk, a);\ PushStack(Stk, i);$
    $a := NextToken();$
  `else if` $action[s, a] =$ `rj` `then` // *Reduce with rule j)* $A \to \alpha$
    `for` $i := 1$ `to` $|\alpha|$ `do`
      $PopStack(Stk);\ PopStack(Stk);$ // *Pop states and symbols*
    $s' := TopStack(Stk);\ s' := goto[s', A];$ // *New state s'*
    $PushStack(Stk, A);\ PushStack(Stk, s');$ // *Push symbol A and s'*
    emit production rule $A \to \alpha$
  `else if` $action[s, a] =$ `acc` `then` // *Accept*
    accept
  `else` *throw syntax error*
`endloop`

LR(0) Items

- An LR(0) item has the form $A \to \alpha \cdot \beta$
  - at this moment α is on [top of] the stack
  - it is expected [at the beginning of the rest of the input] something derivable from β
- For example, at state I_7 of the previous automata:
  - we have $T *$ on top of the stack, and we are expecting something that can be a factor F in order to get a term T of the form $T * F$.
    So item $T \to T * \cdot F \in I_7$
  - we are also directly expecting an identifier `id` to get that factor F, or a left parenthesis $($ to get a factor of the form (E).
    So also items $F \to \cdot \texttt{id}$ and $F \to \cdot (E) \in I_7$

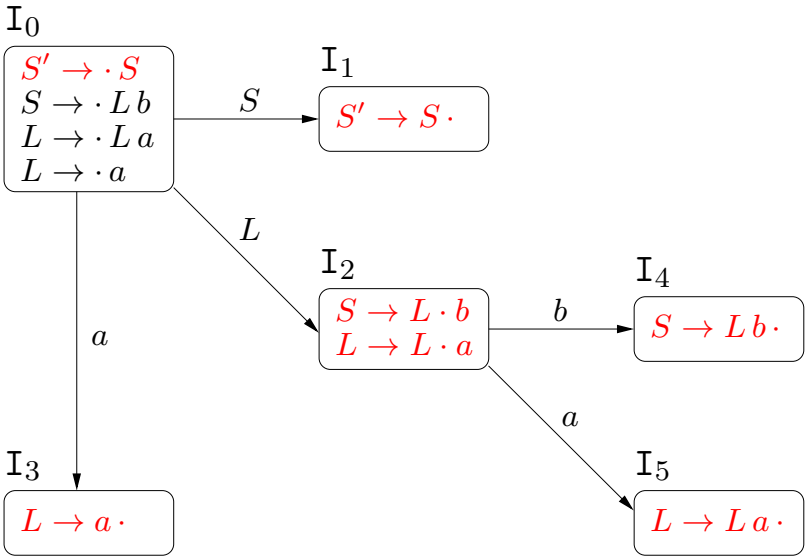
LR(0) NFA

0) $S' \to S$
1) $S \to L\ b$
2) $L \to L\ a$
3) $L \to a$



LR(0) DFA

0) $S' \to S$
1) $S \to L\ b$
2) $L \to L\ a$
3) $L \to a$



LR(0) Tables Construction

- if $A \to \alpha \cdot a\,\beta \in I_i$ and $\texttt{DTran}[I_i,\ a] = I_j$ then

  $action[i,\ a] \supseteq \{\ \texttt{shift}\ a\ \texttt{and go to}\ j\ (\texttt{s}_j)\ \}$

- if $A \to \alpha \cdot \in I_i$ and $n)\ A \to \alpha \in G$ then

  $\forall\, a \in \Sigma \cup \{\$\}$:

  $action[i,\ a] \supseteq \{\ \texttt{reduce with rule}\ n\ (\texttt{r}_n)\ \}$

- if $A \to \alpha \cdot A\,\beta \in I_i$ and $\texttt{DTran}[I_i,\ A] = I_j$ then

  $goto[i,\ A] = j$

LR(0) Tables Construction

0) $S' \to S$
1) $S \to L\ b$
2) $L \to L\ a$
3) $L \to a$

state	action			goto	
	a	b	\$	S	L
0	$s3$			1	2
1			acc		
2	$s5$	$s4$			
3	$r3$	$r3$	$r3$		
4	$r1$	$r1$	$r1$		
5	$r2$	$r2$	$r2$		

LR(0) Tables Construction. Example

0) $E' \to E$
1) $E \to E + T$
2) $E \to T$
3) $T \to T * F$
4) $T \to F$
5) $F \to (\ E\ )$
6) $F \to$ `id`

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

SLR(1) Tables Construction

- if $A \to \alpha \cdot a\,\beta \in I_i$ and $\texttt{DTran}[I_i,\ a] = I_j$ then

  $action[i,\ a] \supseteq \{\ \texttt{shift}\ a\ \texttt{and go to}\ j\ (\texttt{s}_j)\ \}$

- if $A \to \alpha \cdot \in I_i$ and $n)\ A \to \alpha \in G$ then

  $\forall\, a \in follow(A)$:

  $action[i,\ a] \supseteq \{\ \texttt{reduce with rule}\ n\ (\texttt{r}_n)\ \}$

- if $A \to \alpha \cdot A\,\beta \in I_i$ and $\texttt{DTran}[I_i,\ A] = I_j$ then

  $goto[i,\ A] = j$

SLR(1) Tables Construction

0) $S' \to S$
1) $S \to L\ b$
2) $L \to L\ a$
3) $L \to a$

state	*action*			*goto*	
	a	b	\$	S	L
0	$s3$			1	2
1			acc		
2	$s5$	$s4$			
3	$r3$	$r3$			
4			$r1$		
5	$r2$	$r2$			

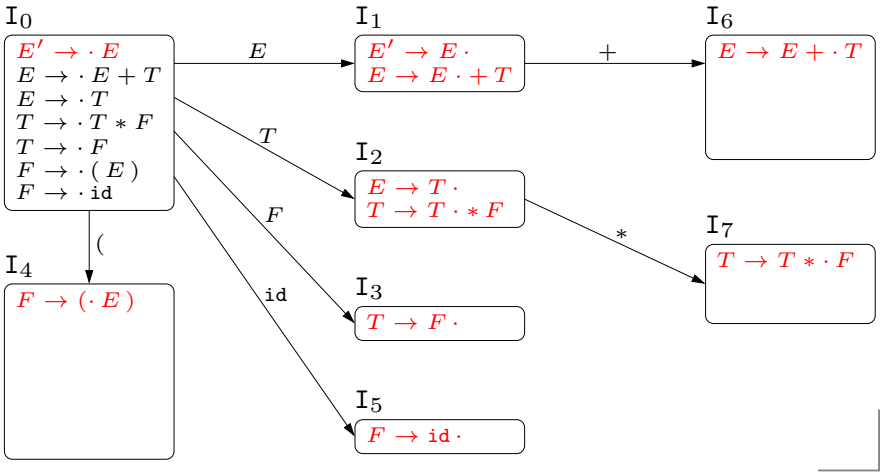
$follow(S) = \{\ \$\ \}$
$follow(L) = \{\ a,\ b\ \}$

SLR(1) Tables Construction. Example

0) $E' \to E$
1) $E \to E + T$
2) $E \to T$
3) $T \to T * F$
4) $T \to F$
5) $F \to (E)$
6) $F \to \texttt{id}$

$follow(E) = \{ +,), \$ \}$
$follow(T) = \{ +, *,), \$ \}$
$follow(F) = \{ +, *,), \$ \}$

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			

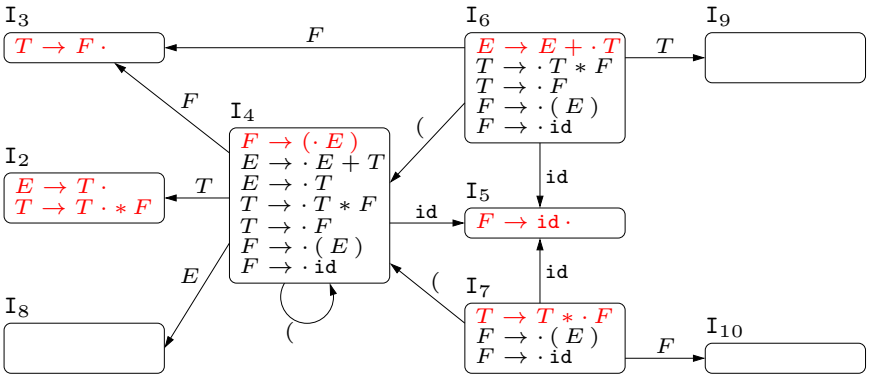


SLR(1) Tables Construction. Example

0) $E' \to E$
1) $E \to E + T$
2) $E \to T$
3) $T \to T * F$
4) $T \to F$
5) $F \to (E)$
6) $F \to \texttt{id}$

$follow(E) = \{ +,), \$ \}$
$follow(T) = \{ +, *,), \$ \}$
$follow(F) = \{ +, *,), \$ \}$

state	action						goto		
	id	+	*	(	)	$	E	T	F
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10

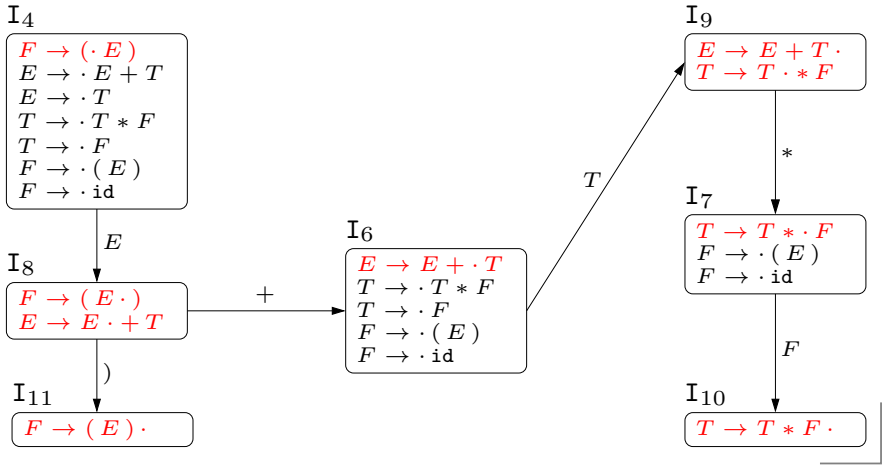


SLR(1) Tables Construction. Example

0) $E' \to E$
1) $E \to E + T$
2) $E \to T$
3) $T \to T * F$
4) $T \to F$
5) $F \to (E)$
6) $F \to \texttt{id}$

$follow(E) = \{ +,), \$ \}$
$follow(T) = \{ +, *,), \$ \}$
$follow(F) = \{ +, *,), \$ \}$

state	action						goto		
	id	+	*	(	)	$	E	T	F
8		s6			s11				
9		r1	**s7**		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			



SLR(1) Tables Construction. Example

0) $E' \to E$
1) $E \to E + T$
2) $E \to T$
3) $T \to T * F$
4) $T \to F$
5) $F \to (E)$
6) $F \to \texttt{id}$

$follow(E) = \{ +,), \$ \}$
$follow(T) = \{ +, *,), \$ \}$
$follow(F) = \{ +, *,), \$ \}$

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			