WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Compiling Techniques - ECOTE

part 7- LR parser

DSc Ilona Bluemke

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!
EUROPEAN UNION
EUROPEAN
SOCIAL FUND
Project is co-financed by European Union within European Social Fund

Bottom –UP parsers

LR parsers

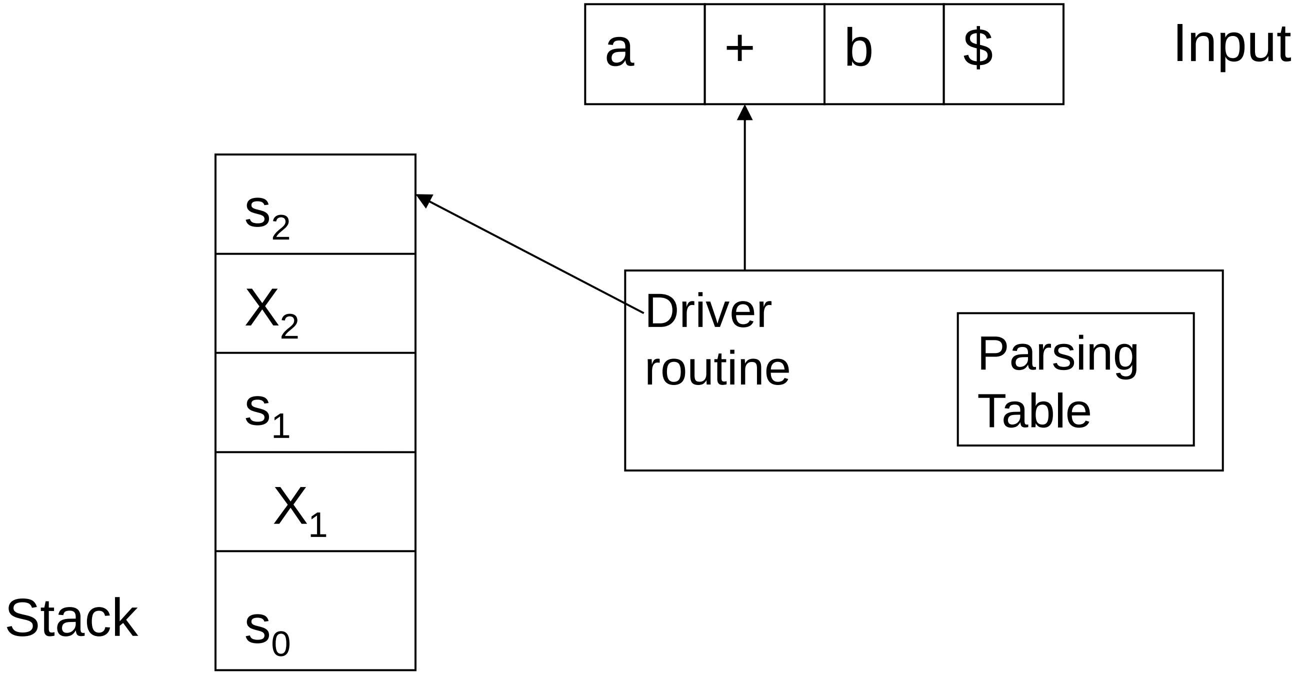
left to right input scanning and

rightmost derivation

- LR parsing method more general than RD, LL, precedence methods
- Good error detection
- LR parsers generators available (parsing table generators)

LR methods:

1. SLR (simple LR)
2. canonical LR (most powerful)
3. LALR – look ahead LR



Stack – $s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_m\ s_m$

where:

- s_i – state (summarising the information contained on stack below),
- X_i – grammar symbol

Parsing table:

- parsing action function **ACTION**
- goto function **GOTO**

Driving routine:

- Determines $\mathbf{s_m}$ (stack top symbol) and $\mathbf{a_i}$ (current input symbol)
- Consults **ACTION[$\mathbf{s_m}$, $\mathbf{a_i}$]**

  ACTION[s_m, a_i] can have values:
  - shift s
  - reduce $A \rightarrow \beta$
  - accept
  - error

configuration of a LR parser:

$(s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_m\ s_m\ ,\ a_i\ a_{i+1}\ \ldots\ a_n\$\ )$

the configuration resulting after each of four types of move are as follows:

1. if ACTION[s_m, a_i] = shift **s** the parser executes a shift move entering the configuration:

   $(s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_m\ s_m\ a_i s,\ a_{i+1}\ \ldots\ a_n\$\ )$

   a_{i+1} becomes the new current input symbol

2. if ACTION[s_m, a_i] = reduce $A \to \beta$ the parser executes a reduce move entering the configuration:

(s_0 X_1 s_1 X_2 s_2 ... X_{m-r} s_{m-r}As, a_i a_{i+1} ... a_n\$)

where **s = GOTO[s_{m-r} ,A]**

r is the length of **β** . The parsers first pops **2r** symbols off the stack (r state symbols, r grammar symbols), exposing state **s_{m-r}** , then pushes **A** and **GOTO[s_{m-r} ,A]** The current input symbol is not changed. The sequence of grammar symbols X_{m-r+1} X_m popped off the stack will always match β.

3. if **ACTION[s_m, a_i] = accept**, parsing is completed
4. **ACTION[s_m, a_i] = error**, parser calls error recovery routine

Initially parser is in configuration:

(s_0 , a_1a_2 ... a_n\$)

Code for actions:

- **s_i** – shift and **i** next state
- **r_j** – reduce by production numbered **j**
- acc – accept
- blank – error

Example:

1) $S \rightarrow S+A$
2) $S \rightarrow A$
3) $A \rightarrow A * B$
4) $A \rightarrow B$
5) $B \rightarrow (S)$
6) $B \rightarrow a$

ACTION

function

State	a	+	*	(	)	$
0	s5			s4		
1		s6				acc
2		r2	s7		r2	r2
3		r4	r4		r4	r4
4	s5			s4		
5		r6	r6		r6	r6
6	s5			s4		
7	s5			s4		
8		s6			s11	
9		r1	s7		r1	r1
10		r3	r3		r3	r3
11		r5	r5		r5	r5

GOTO function

state	S	A	B
0	1	2	3
1			
2			
3			
4	8	2	3
5			
6		9	3
7			10
8			
9			
10			
11			

Parser movements

	stack	input
1.	0	a*a+a$
2.	0a5	*a+a$
3.	0B3	*a+a$
4.	0A2	*a+a$
5.	0A2*7	a+a$
6.	0A2*7a5	+a$
7.	0A2*7B10	+a$
8.	0A2	+a$
9.	0S1	+a$
10.	0S1+6	a$

Parser movements

	stack	input
11.	0S1+6a5	$
12.	0S1+6B3	$
13.	0S1+6A9	$
14.	0S1	$

LR grammar

A grammar for which we can construct a parsing table in which every entry is uniquely defined

LR(k) – k input symbols can be used to help make shift/reduce decisions

The canonical collection of LR(0) items

- Construction of a **DFA** from the grammar, then turning it into parsing table
- DFA recognises **viable prefixes** – prefixes of the right-sentential forms that do not contain any symbols to the right of the handle

LR(0) item – item

Item it is a production of G with a dot at some position of the right side.

Production $A \rightarrow XYZ$ generates 4 items:

1. $A \rightarrow .XYZ$, 2. $A \rightarrow X.YZ$,

3. $A \rightarrow XY.Z$, 4. $A \rightarrow XYZ.$

- Items can be represented as a pairs of integers (production number, position of the dot)
- An item indicates how much of a production we have seen at a given point in the parsing process.

Example

1. $A \rightarrow .XYZ$

We are expecting to see a string derivable from XYZ next on the input

2. $A \rightarrow X.YZ$

We have just seen a string derivable from X and we expect to see a string derivable from YZ

- We group items into sets. The items can be viewed as states of NFA recognising viable prefixes.

- One collection of sets of items – **canonical collection LR(0)** provides a basis for constructing simple LR parsers (**SLR**).

G is a grammar with S starting symbol

G' is augmented grammar of G – is G with new starting symbol **S'** and production **S'→S**.

The purpose of this new symbol is to indicate when the parser should stop (when reduction S'→S).

Closure (I)

If I is a set of items for a grammar G, then the set of items Closure (I) is constructed from I by the rules:

1. Every item in **I** is in **Closure (I)**
2. If $\mathbf{A \rightarrow \alpha . B \beta}$ is in Closure (I) and $\mathbf{B \rightarrow \gamma}$ is a production , then add the item $\mathbf{B \rightarrow . \gamma}$ if it is not already there.
3. Repeat step 2 until no more items can be added

Example:

1) $S \rightarrow S+A$
2) $S \rightarrow A$
3) $A \rightarrow A * B$
4) $A \rightarrow B$
5) $B \rightarrow (S)$
6) $B \rightarrow a$

Example

Set of items {[S'→.S]}, Closure (I) contains:

S'→.S,

S →.A, S →.S+A

A →.B, A →.A*B

B →.a, B →.(S)

goto(I, X)

I – set of items

X - grammar symbol

goto (I, X) – the closure of the set of all items **[A→αX.β]** such that **[A→α.Xβ]** is in **I**.

Example

$I=\{[S' \to S.], [S \to S.+A]\}$

Then **goto(I, +)** consists of :

$S \to S+.A$,

$A \to .B$,

$A \to .A*B$,

$B \to .a$,

$B \to .(S)$

The sets of items construction

procedure ITEMS(G')
begin
  C:= {CLOSURE({S'→.S})};
  repeat
    for each set of items **I** in **C** and each grammar symbol such that **goto(I,X)** is not empty and is not in C
      do add goto(I,X) to C
  until no more items can be added to C
end

I_0 :

- $S' \to .S$,
- $S \to .A$,
- $S \to .S+A$
- $A \to .B$
- $A \to .A*B$
- $B \to .a$
- $B \to .(S)$

$I_1 = goto\ (I_0, S)$:

- $S' \to S.$
- $S \to S.+A$

$I_2 = goto\ (I_0, A)$:

- $S \to A.$
- $A \to A.*B$

$I_3 = goto\ (I_0, B)$:

- $A \to B.$

I_4 = goto (I_0, „(”):

- $B \rightarrow (.S)$
- $S \rightarrow .A$
- $S \rightarrow .S+A$
- $A \rightarrow .B$
- $A \rightarrow .A*B$
- $B \rightarrow .a$
- $B \rightarrow .(S)$

I_5 = goto (I_0, a):

- $B \rightarrow a.$

I_6 = goto (I_1, +):

- $S \rightarrow S+.A$
- $A \rightarrow .B$
- $A \rightarrow .A*B$
- $B \rightarrow .a$
- $B \rightarrow .(S)$

I_7 = goto (I_2, *):

- $A \rightarrow A*.B$
- $B \rightarrow .a$
- $B \rightarrow .(S)$

I_8 = goto (I_4, S):

- B → (S**.**)
- S →S**.**+A

I_9 = goto (I_6, A):

- S →S+A**.**
- A →A**.***B

I_{10} = goto (I_7, B):

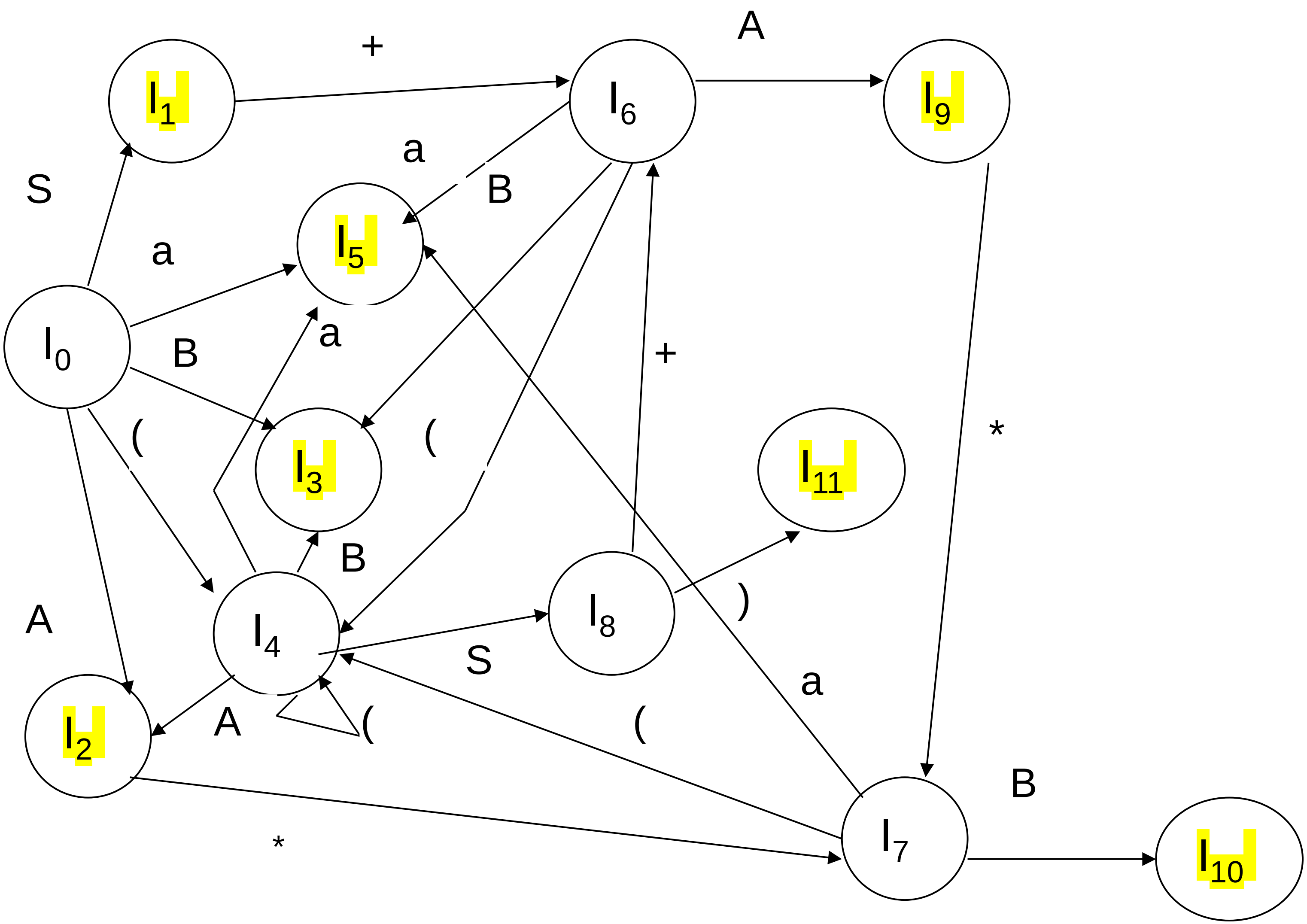
- A →A*B**.**

I_{11} = goto (I_8, „)"):

- B → (S) **.**

WARSAW UNIVERSITY OF TECHNOLOGY DEVELOPMENT PROGRAMME

DFA recognizing viable prefixes



Constructing SLR parsing tables

C – canonical collection of sets of items for augmented grammar G'.

Let **C= { I_0 , I_1 ,... I_n}**

The states of the parser are **0, 1,...n**, state **i** being constructed from **I_i** .

The parsing actions are determined as follows:

1. If **[$A \rightarrow \alpha.a\beta$]** is in **$I_i$** and **goto($I_i$,a)=$I_j$** then set ACTION [i, a] to shift **j**.
2. If **[$A \rightarrow \alpha.$]** is in **I_i** then set ACTION [i, a] to reduce $A \rightarrow \alpha$ for all a in $FOLLOW_1(A)$.

3. If **[S'→S.]** is in I_i then set ACTION[i,$] to accept.

The **GOTO** transitions for state **i** are constructed using the rule:

4. If **goto[I_i , A] = I_j**, **GOTO[i,A]=j**
5. All entries not defined by rules 1-4 are made error
6. The initial state of the parser is the one constructed from the set **[S'→.S]**

ACTION

function

Fill the table

State	a	+	*	(	)	$
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						

GOTO function

Fill the table

state	S	A	B
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			

SLR grammar

- No conflicts in parsing tables (at most one entry in a cell)

WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Compiling Techniques - ECOTE

end of part 7- LR parser

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund