Compilers
Parsing *bottom-up*

LEIC

FEUP-FCUP

2022

This lecture

Parsers LR

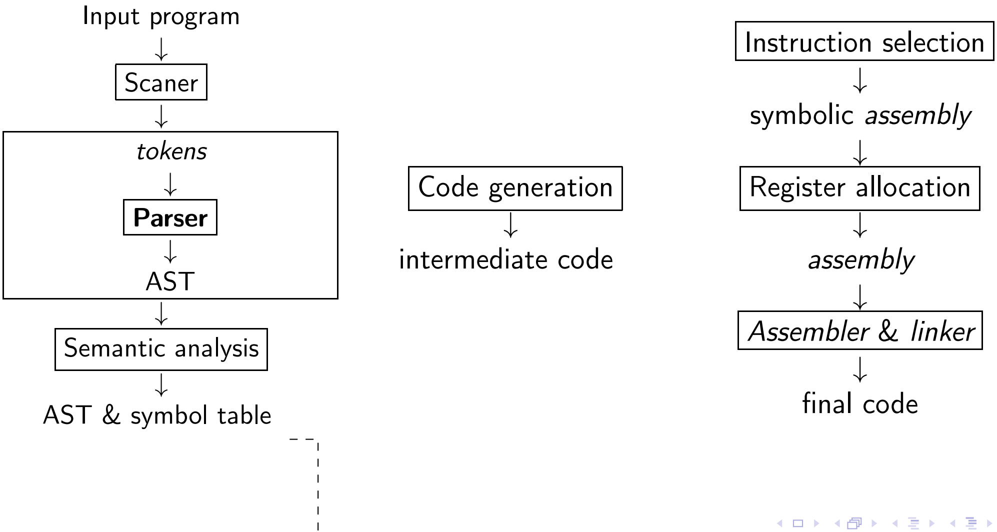
$LR(0)$

Parsing $SLR(1)$

Conflicts

Extras

Compiler


Input program
Scaner
tokens
Parser
AST
Semantic analysis
AST & symbol table
Code generation
intermediate code
Instruction selection
symbolic assembly
Register allocation
assembly
Assembler & linker
final code

This lecture

Parsers LR

$LR(0)$

Parsing $SLR(1)$

Conflicts

Extras

Parsing *bottom-up*

- Last lecture *LL*(1)
  (*Left-right parse, Leftmost derivation, 1-symbol lookahead*)
- Problem: rewrite a grammar to be *LL*(1)
- Parsing *LR* (*Left-right parse, Rightmost derivation*)
- Features:
  - Parsers *LR* are more general
  - Easier to rewrite a grammar to be *LR* than *LL*
  - *LR* parsing deals easier with ambiguity: define operators *priorities* and *associativity* without the need to rewrite the grammar

Parsing LR

Stack automata:

- Terminal symbols: (input)
- A stack of symbols (terminals and not terminals)
- Initially the stack is empty
- In each step choose::

Shift push the next symbol to the stack;

Reduce Choose a rule of the form $X \to \gamma$ such that γ are on the top of the stack; remove those symbols and push X.

Parsing LR

Stack automata:

- Terminal symbols: (input)
- A stack of symbols (terminals and not terminals)
- Initially the stack is empty
- In each step choose::

  Shift push the next symbol to the stack;

  Reduce Choose a rule of the form $X \to \gamma$ such that γ are on the top of the stack; remove those symbols and push X.

Add:

- An initial symbol S' and an end symbol \$;
- a new rule $S' \to S\,\$$

Example 1

$$
\begin{aligned}
S' &\to S\,\$ \\
S &\to (\,S\,)\,S \quad | \quad \varepsilon
\end{aligned}
$$

LR parsing step by step:

pilha	*input*	ação
ε	$()\$$	shift
$($	$)\$$	reduce $S \to \varepsilon$
$(S$	$)\$$	shift
(S)	$\$$	reduce $S \to \varepsilon$
$(S)S$	$\$$	reduce $S \to (S)S$
S	$\$$	accept

Reading back the sequence of actions we have the derivation:

$$S \Rightarrow (S)S \Rightarrow (S) \Rightarrow ()$$

Examplo 2

Expressions:

$$E' \to E\,\$$$
$$E \to E\texttt{+n} \quad | \quad \texttt{n}$$

pilha	*input*	ação
ε	$\texttt{n+n\$}$	shift
$\texttt{n}$	$\texttt{+n\$}$	reduce $E \to \texttt{n}$
E	$\texttt{+n\$}$	shift
$E+$	$\texttt{n\$}$	shift
$E + \texttt{n}$	$\$$	reduce $E \to E + \texttt{n}$
E	$\$$	accept

The corresponding derivation:

$$E \Rightarrow E\texttt{+n} \Rightarrow \texttt{n+n}$$

LR Automata

The LR automata chooses the next action:

- using the stack configuration (not only the top of the stack)
- the next input symbols (*look-ahead*)

LR Automata

The LR automata chooses the next action:

- using the stack configuration (not only the top of the stack)
- the next input symbols (*look-ahead*)

- Let us use integers to represent the states of the automata (stack configurations)
- The automata changes state when we push or pop symbols into (or from) the stack
- These actions are described by the LR *parsing* table

LR *parsing* table

- Lines: states (integers)
- Columns:symbols (terminals or non-terminals)
- Entries: actions

shift q push a token and go to state q

reduce k let $X \to \alpha_1 \dots \alpha_n$ be rule number k:

1. pop $\alpha_n, \dots, \alpha_1$ (i.e. the right hand side symbols)
2. go back to the state on the top of the new stack and push X;
3. lookup in the table for the state "*go q*" for the entry X in this state;
4. go to q

go q go to q (after a reduce *reduce*)

accept accepts the input

Example

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

$$
\begin{array}{ll}
(0) & T' \to T\,\$ \\
(1) & T \to R \\
(2) & T \to aTc \\
(3) & R \to \varepsilon \\
(4) & R \to bR
\end{array}
$$

Example

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

$$
\begin{array}{ll}
(0) & T' \to T\,\$ \\
(1) & T \to R \\
(2) & T \to aTc \\
(3) & R \to \varepsilon \\
(4) & R \to bR
\end{array}
$$

Some transitions:

- in state 0, with next symbol a, *shift* and changes to state 3;

Example

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

Some transitions:

- in state 0, with next symbol a, *shift* and changes to state 3;
- in state 3, with next symbol c, *reduce* using rule 3 ($R \to \varepsilon$);

Example

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

Some transitions:

- in state 0, with next symbol a, *shift* and changes to state 3;
- in state 3, with next symbol c, *reduce* using rule 3 ($R \to \varepsilon$);
- in state 0, after *reduce* $T \to \ldots$ go to state 1;

Example

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

$$\begin{array}{ll} (0) & T' \to T\,\$ \\ (1) & T \to R \\ (2) & T \to aTc \\ (3) & R \to \varepsilon \\ (4) & R \to bR \end{array}$$

Some transitions:

- in state 0, with next symbol a, *shift* and changes to state 3;
- in state 3, with next symbol c, *reduce* using rule 3 ($R \to \varepsilon$);
- in state 0, after *reduce* $T \to \ldots$ go to state 1;
- in state 1, with next symbol \$ (*accept*) the input.

Parsing

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

(0) $T' \rightarrow T\,\$$
(1) $T \rightarrow R$
(2) $T \rightarrow aTc$
(3) $R \rightarrow \varepsilon$
(4) $R \rightarrow bR$

state	stack	input	action
0	ε	aabbbcc$	

Parsing

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

state	stack	input	action
0	ε	aabbbcc$	shift 3
3	a	abbbcc$	

$$
\begin{array}{ll}
(0) & T' \rightarrow T\,\$ \\
(1) & T \rightarrow R \\
(2) & T \rightarrow aTc \\
(3) & R \rightarrow \varepsilon \\
(4) & R \rightarrow bR
\end{array}
$$

Parsing

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

state	stack	input	action
0	ε	aabbbcc\$	shift 3
3	a	abbbcc\$	shift 3
3	aa	bbbcc\$	

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

Parsing

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

state	stack	input	action
0	ε	aabbbcc$	shift 3
3	a	abbbcc$	shift 3
3	aa	bbbcc$	shift 4
4	aab	bbcc$	

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

Parsing

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

state	stack	input	action
0	ε	aabbbcc$	shift 3
3	a	abbbcc$	shift 3
3	aa	bbbcc$	shift 4
4	aab	bbcc$	shift 4
4	aabb	bcc$	

Parsing

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

state	stack	input	action
0	ε	aabbbcc\$	shift 3
3	a	abbbcc\$	shift 3
3	aa	bbbcc\$	shift 4
4	aab	bbcc\$	shift 4
4	aabb	bcc\$	shift 4
4	aabbb	cc\$	

(0) $T' \rightarrow T\,\$$
(1) $T \rightarrow R$
(2) $T \rightarrow aTc$
(3) $R \rightarrow \varepsilon$
(4) $R \rightarrow bR$

Parsing

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

state	stack	input	action
0	ε	aabbbcc$	shift 3
3	a	abbbcc$	shift 3
3	aa	bbbcc$	shift 4
4	aab	bbcc$	shift 4
4	aabb	bcc$	shift 4
4	aabbb	cc$	reduce $R \to \varepsilon$; go 6
6	aabbbR	cc$	

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

Parsing

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

state	stack	input	action
0	ε	aabbbcc$	shift 3
3	a	abbbcc$	shift 3
3	aa	bbbcc$	shift 4
4	aab	bbcc$	shift 4
4	aabb	bcc$	shift 4
4	aabbb	cc$	reduce $R \to \varepsilon$; go 6
6	aabbbR	cc$	reduce $R \to bR$; go 6
6	aabbR	cc$	

(0) $T' \to T\,\$$

(1) $T \to R$

(2) $T \to aTc$

(3) $R \to \varepsilon$

(4) $R \to bR$

Parsing

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

state	stack	input	action
0	ε	aabbbcc$	shift 3
3	a	abbbcc$	shift 3
3	aa	bbbcc$	shift 4
4	aab	bbcc$	shift 4
4	aabb	bcc$	shift 4
4	aabbb	cc$	reduce $R \rightarrow \varepsilon$; go 6
6	aabbbR	cc$	reduce $R \rightarrow bR$; go 6
6	aabbR	cc$	reduce $R \rightarrow bR$; go 6
6	aabR	cc$	

(0) $T' \rightarrow T\,\$$
(1) $T \rightarrow R$
(2) $T \rightarrow aTc$
(3) $R \rightarrow \varepsilon$
(4) $R \rightarrow bR$

Parsing

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

state	stack	input	action
0	ε	aabbbcc$	shift 3
3	a	abbbcc$	shift 3
3	aa	bbbcc$	shift 4
4	aab	bbcc$	shift 4
4	aabb	bcc$	shift 4
4	aabbb	cc$	reduce $R \to \varepsilon$; go 6
6	aabbbR	cc$	reduce $R \to bR$; go 6
6	aabbR	cc$	reduce $R \to bR$; go 6
6	aabR	cc$	reduce $R \to bR$; go 2
2	aaR	cc$	

Parsing

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

state	stack	input	action
0	ε	aabbbcc$	shift 3
3	a	abbbcc$	shift 3
3	aa	bbbcc$	shift 4
4	aab	bbcc$	shift 4
4	aabb	bcc$	shift 4
4	aabbb	cc$	reduce $R \to \varepsilon$; go 6
6	aabbbR	cc$	reduce $R \to bR$; go 6
6	aabbR	cc$	reduce $R \to bR$; go 6
6	aabR	cc$	reduce $R \to bR$; go 2
2	aaR	cc$	reduce $T \to R$; go 5
5	aaT	cc$	

Parsing

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

state	stack	input	action
0	ε	aabbbcc\$	shift 3
3	a	abbbcc\$	shift 3
3	aa	bbbcc\$	shift 4
4	aab	bbcc\$	shift 4
4	aabb	bcc\$	shift 4
4	aabbb	cc\$	reduce $R \to \varepsilon$; go 6
6	aabbbR	cc\$	reduce $R \to bR$; go 6
6	aabbR	cc\$	reduce $R \to bR$; go 6
6	aabR	cc\$	reduce $R \to bR$; go 2
2	aaR	cc\$	reduce $T \to R$; go 5
5	aaT	cc\$	shift 7
7	aaTc	c\$	

Parsing

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

state	stack	input	action
0	ε	aabbbcc\$	shift 3
3	a	abbbcc\$	shift 3
3	aa	bbbcc\$	shift 4
4	aab	bbcc\$	shift 4
4	aabb	bcc\$	shift 4
4	aabbb	cc\$	reduce $R \to \varepsilon$; go 6
6	aabbbR	cc\$	reduce $R \to bR$; go 6
6	aabbR	cc\$	reduce $R \to bR$; go 6
6	aabR	cc\$	reduce $R \to bR$; go 2
2	aaR	cc\$	reduce $T \to R$; go 5
5	aaT	cc\$	shift 7
7	aaTc	c\$	reduce $T \to aTc$; go 5
5	aT	c\$	

Parsing

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

state	stack	input	action
0	ε	aabbbcc\$	shift 3
3	a	abbbcc\$	shift 3
3	aa	bbbcc\$	shift 4
4	aab	bbcc\$	shift 4
4	aabb	bcc\$	shift 4
4	aabbb	cc\$	reduce $R \to \varepsilon$; go 6
6	aabbbR	cc\$	reduce $R \to bR$; go 6
6	aabbR	cc\$	reduce $R \to bR$; go 6
6	aabR	cc\$	reduce $R \to bR$; go 2
2	aaR	cc\$	reduce $T \to R$; go 5
5	aaT	cc\$	shift 7
7	aaTc	c\$	reduce $T \to aTc$; go 5
5	aT	c\$	shift 7
7	aTc	\$	

Parsing

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

state	stack	input	action
0	ε	aabbbcc$	shift 3
3	a	abbbcc$	shift 3
3	aa	bbbcc$	shift 4
4	aab	bbcc$	shift 4
4	aabb	bcc$	shift 4
4	aabbb	cc$	reduce $R \to \varepsilon$; go 6
6	aabbbR	cc$	reduce $R \to bR$; go 6
6	aabbR	cc$	reduce $R \to bR$; go 6
6	aabR	cc$	reduce $R \to bR$; go 2
2	aaR	cc$	reduce $T \to R$; go 5
5	aaT	cc$	shift 7
7	aaTc	c$	reduce $T \to aTc$; go 5
5	aT	c$	shift 7
7	aTc	$	reduce $T \to aTc$; go 1
1	T	$	

Parsing

	a	b	c	$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

(0) $T' \to T\,\$$
(1) $T \to R$
(2) $T \to aTc$
(3) $R \to \varepsilon$
(4) $R \to bR$

state	stack	input	action
0	ε	aabbbcc$	shift 3
3	a	abbbcc$	shift 3
3	aa	bbbcc$	shift 4
4	aab	bbcc$	shift 4
4	aabb	bcc$	shift 4
4	aabbb	cc$	reduce $R \to \varepsilon$; go 6
6	aabbbR	cc$	reduce $R \to bR$; go 6
6	aabbR	cc$	reduce $R \to bR$; go 6
6	aabR	cc$	reduce $R \to bR$; go 2
2	aaR	cc$	reduce $T \to R$; go 5
5	aaT	cc$	shift 7
7	aaTc	c$	reduce $T \to aTc$; go 5
5	aT	c$	shift 7
7	aTc	$	reduce $T \to aTc$; go 1
1	T	$	**accept**

LR *parsing* algorithm

```
stack= empty; push(0, stack); next=getToken()
loop
   case table[top(stack),next] of
     shift s: push(s, stack); next=getToken()

     reduce p: let X = left-hand-side of production p
                   n = length(right-hand-side of production p)
               pop n elements of stack;
               lookup table[top(stack),X] and find "go s";
               push(s,stack)

     accept:   terminate with sucess

     empty:    report error
```

LR *parsing* algorithm (cont.)

- In each step the algorithm uses the table to choose the action
- It is sufficient to store the sates in the stack
  - each state represents a configuration of symbols in the stack
  - it is not necessary to store the symbols
- The difficult part is building the parsing table
- We build the table from the grammar (we dont need the *input*)

Building the *parsing* table

- Choose actions using the stack and the next tokens (*look-ahead*):

  $LR(0)$ (0 *look-ahead* symbols)
  $LR(1)$ 1 *look-ahead* symbol
  $LR(k)$ k *look-ahead* symbols

- $LR(0)$ is not enough powerful to parse programming languages
- $LR(k)$ with $k \geq 2$ requires huge tables
- $LR(1)$ is suficent to parse programming languages

This lecture

$LR(0)$ items

- *Items* are with an extra symbol (the position) (dot)
- Automata *states* are sets of items

Example: the grammar on the left has 3 rules corresponding to 9 items (on the right hand side).

$$
\begin{aligned}
S' &\to S\,\$ \\
S &\to (\,S\,)\,S \\
S &\to \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
S' &\to .S\,\$ \\
S' &\to S.\,\$ \\
S' &\to S\,\$. \\
S &\to .(\,S\,)\,S \\
S &\to (.S\,)\,S \\
S &\to (\,S.)\,S \\
S &\to (\,S\,).S \\
S &\to (\,S\,)\,S. \\
S &\to .
\end{aligned}
$$

$LR(0)$ items (cont.)

- Items represent an intermediate state
- Examplo: $S \to (S).S$
  - top of the stack: (S)
  - the automata already recognized (S) and continues with S
- $$A \to \beta.\gamma$$

  means that β on the top of the stack and the automata may continue with γ
- $A \to .\gamma$ is the initial item: start with γ
- $A \to \gamma.$ is a complete item: γ is on the top of the stack and we recognize A (*reduce*)

Transitions

- States are items
- For each item transitions using terminal and non-terminal symbols
- Examples:

$$S \rightarrow .(S)S \xrightarrow{(} S \rightarrow (.S)S$$

$$S \rightarrow (.S)S \xrightarrow{S} S \rightarrow (S.)S$$

- A transition using a terminal symbol occurs after a shift
- A transition using a non-terminal symbol occurs after a reduce

Transitions-ε

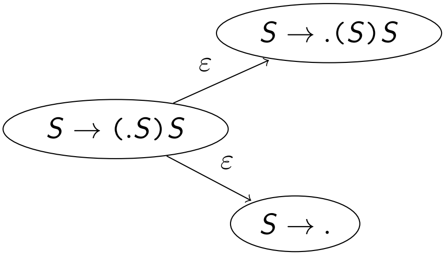
- For each item with next non-terminal symbol B

$$A \to \alpha . B \gamma$$

add transitions-ε for *every* initial items of B

$$B \to .\beta$$

- Exemplo:



Initial state

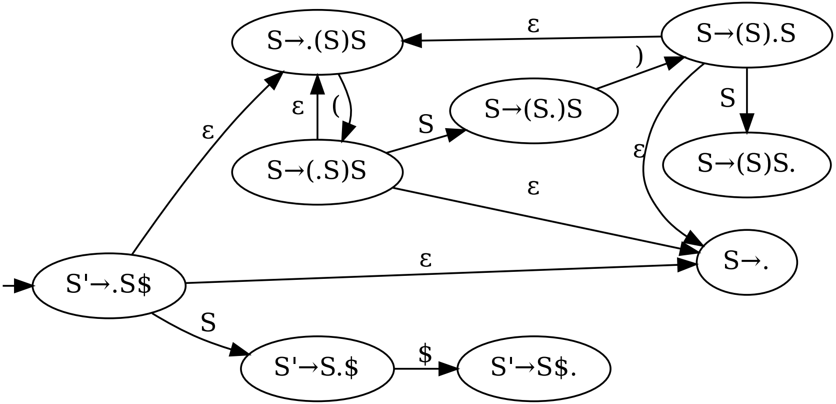
- Initial state: initial item

$$\longrightarrow \boxed{S' \to .S\,\$}$$

(S' is the new non-terminal added to the grammar)

- This automata does not have final states
- Accept occurs when we *shift* the end of input symbol \$ (corresponding to an action in the *parsing table*)

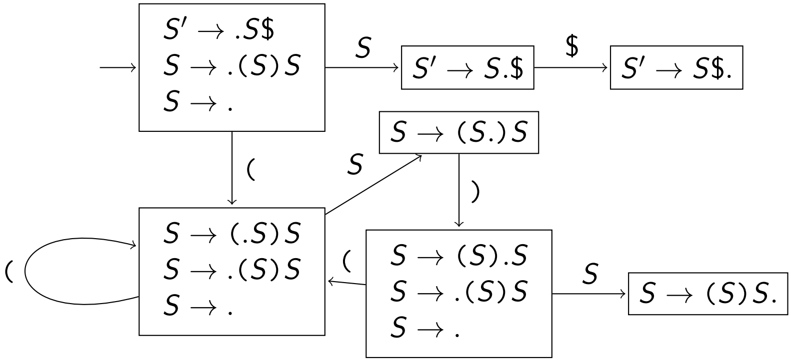
Example



Deterministic LR automata

- Due to transitions-ε the automata is non-deterministic (NFA)
- Let us transform it into a deterministic automata (DFA)
- DFA states are now sets of items

Example: DFA automata



Building the $LR(0)$ table

		tokens	non-terminals
states{	0	$\cdots$	$\cdots$
	1	$\cdots$	$\cdots$
	$\vdots$		
	n	$\cdots$	$\cdots$

- States of the DFA are numbered
- Transitions with a terminal symbol: shift
- Transitions with a non-terminal symbol: go
- For each state with a complete item ($A \to \gamma.$): reduce
- For state $\{S' \to S.\$\}$ with next symbol \$: accept

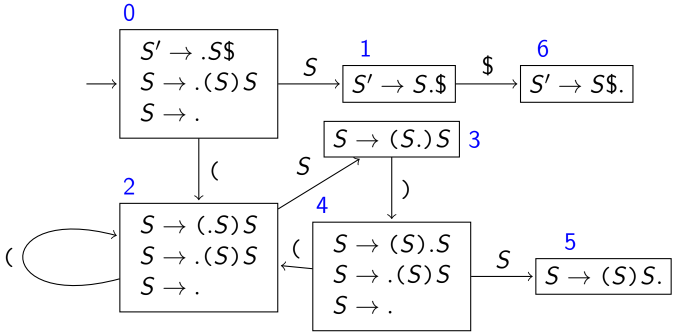
The grammar $LR(0)$ if each entry has at most one action.

Example

	(	)	$	S
0				
1				
2				
3				
4				
5				

Example

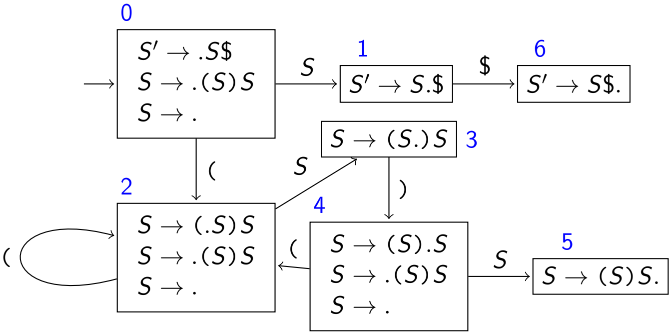
	(	)	\$	S
0	s2			
1				
2	s2			
3		s4		
4	s2			
5				



- Transitions using tokens (shift)

Example

	(	)	\$	S
0	s2			g1
1				
2	s2			g3
3		s4		
4	s2			g5
5				

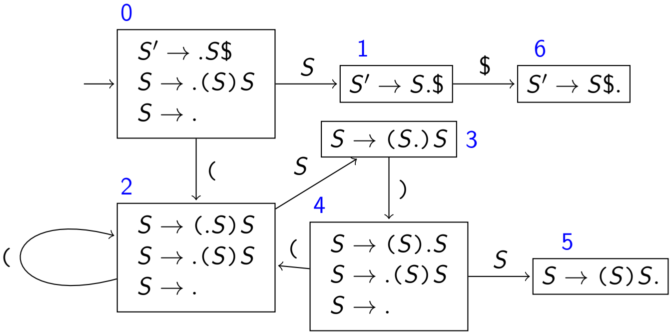


- Transitions using tokens (shift)
- Transições using non-terminal symbols (goto)

Example

	(	)	\$	S
0	s2,r2	r2	r2	g1
1			a	
2	s2,r2	r2	r2	g3
3		s4		
4	s2,r2	r2	r2	g5
5	r1	r1	r1	

0	$S' \to S\,\$$
1	$S \to (S)\,S$
2	$S \to \varepsilon$

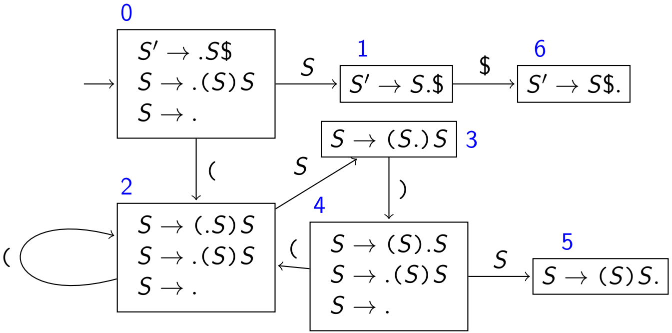


- Transitions using tokens (shift)
- Transições using non-terminal symbols (goto)
- Complete items (reduce) and shift using \$ (accept)

Example

	(	)	\$	S
0	s2,r2	r2	r2	g1
1			a	
2	s2,r2	r2	r2	g3
3		s4		
4	s2,r2	r2	r2	g5
5	r1	r1	r1	

0 $S' \to S\,\$$
1 $S \to (S)\,S$
2 $S \to \varepsilon$



- Transitions using tokens (shift)
- Transições using non-terminal symbols (goto)
- Complete items (reduce) and shift using \$ (accept)
- There are shift/reduce conflits, thus this grammar is not $LR(0)$

Exercise 1

Show that the following grammar is LR(0)

$$A \rightarrow (A) \quad | \quad \texttt{a}$$

building the automata and parsing table.

This lecture

Parsers LR

$LR(0)$

Parsing $SLR(1)$

Conflicts

Extras

$LR(0)$ limitations

- $LR(0)$ uses only information in the stack to choose the reduce actions
- Several times this generates shift-reduce conflits
- We now will use also the next token (*look-ahead*)
- A simple extension of $LR(0)$: $SLR(1)$
  (*Simple Left-right parse, Rightmost derivation 1 symbol look-ahead*)

$SLR(1)$ parsing

- The *parsing* algorithm is the same
- Build the automata as in the $LR(0)$ case
- Building the *parsing* table:
  - add *shift goto* as for $LR(0)$;
  - add *reduce* actions for rules $A \to \gamma.$ only in columns for tokes in FOLLOW(A)

Example

	(	)	$	S
0	s2			g1
1			a	
2	s2			g3
3		s4		
4	s2			g5
5				

- Same transitions for tokens (shift) and non-terminals symbols (goto)

Example

	(	)	$	S
0	s2	r2	r2	g1
1			a	
2	s2	r2	r2	g3
3		s4		
4	s2	r2	r2	g5
5		r1	r1	

0 $S' \to S\,\$$

1 $S \to (S)\,S$

2 $S \to \varepsilon$

$\text{FOLLOW}(S) = \{),\$\}$

- Same transitions for tokens (shift) and non-terminals symbols (goto)
- Complete items (reduce) use *look-ahead*
- There are no conflicts *shift* and *reduce* — the grammar is $SLR(1)$.

Exercise

Build the transition automata and the parsing table and show that the following grammar:

$$
\begin{aligned}
T &\rightarrow R \\
T &\rightarrow aTc \\
R &\rightarrow \varepsilon \\
R &\rightarrow bR
\end{aligned}
$$

is $SLR(1)$.

(the table must not have conflicts)

This lecture

Conflicts

- When a table entry has more that one action we say that there is a conflict
- Conflicts mean that there several possible choices:

  shift/reduce: *shift* and *reduce* in the same state

  reduce/reduce: more than one *reduce* in the same state
- How to solve conflicts:
  - rewrite the grammar (if it is ambiguous)
  - define precedence and priority of *tokens*
  - choose *shift* instead of *reduce* (by default)

Example: "Dangling else"

$$S \to \texttt{if cond then } S \texttt{ else } S$$
$$S \to \texttt{if cond then } S$$
$$S \to \texttt{skip}$$

This grammar is ambiguous. The following sentence:

$$\texttt{if cond then if cond then skip else skip}$$

has two derivation trees:

$$\texttt{if cond then } \{\texttt{if cond then skip else skip}\} \tag{1}$$
$$\texttt{if cond then } \{\texttt{if cond then skip}\} \texttt{ else skip} \tag{2}$$

Example: "Dangling else" (cont.)

Building the SLR(1) automata we get the following state:

$$\boxed{\begin{array}{l} S \rightarrow \texttt{if cond then } S\ . \\ S \rightarrow \texttt{if cond then } S\ .\ \texttt{else } S \end{array}}$$

When the next *token* is `else` we may:

- *shift*
- *reduce* (because FOLLOW(S) $= \{\texttt{else}, \$\}$)

Thus, there is a shift/reduce conflict.

If we choose *shift* we get the usual tree:

$$\texttt{if cond then } \{\texttt{if cond then skip else skip}\}$$

used in programming languages (Pascal, C, Java, etc.)

Exercise 2

For the following grammar:

$$E \to E + E \quad | \quad E * E \quad | \quad \texttt{num}$$

Show that it is ambiguous and has shift-reduce conflicts in SLR(1).

This lecture

Parsers LR

$LR(0)$

Parsing $SLR(1)$

Conflicts

Extras

$LR(1)$ and $LALR(1)$ Parsing

- $SLR(1)$ is not enough powerful to parse programming languages
- $LR(1)$ is a generalization of $SLR(1)$ which solves these problems
- However: $LR(1)$ produces larger automata compared with $SLR(1)$
- Bottom-up parser generators use a simplified version of $LR(1)$: $LALR(1)$ — *Look-ahead* $LR(1)$
- Diferences between $SLR(1)$, $LR(1)$ e $LALR(1)$ are quite technical
- The general principal is the same: use lookahead information

$LR(1)$

- $LR(0)$ states represent only the right-hand side of grammar rules
- $LR(1)$: states also use the *look-ahead* symbol

Items $LR(1)$

- *items* are now pairs:
  - productions with one position in the right-hand side (i.e. $LR(0)$ items);
  - and a terminal symbol (token) (*look-ahead*)

$$(\underbrace{A \rightarrow \alpha.\beta,}_{\text{item } LR(0)} \underbrace{a}_{\text{look-ahead}})$$

- As before: automata states are sets of items
- When the automata is on an item:

$$(A \rightarrow \alpha.\beta,\ a)$$

the sequence α is on the top of the stack and rest is derivable from βa

Items $LR(1)$ (cont.)

Transitions:

$$(A \to \alpha.X\gamma,\ a) \xrightarrow{X} (A \to \alpha X.\gamma,\ a)$$

Transitions-ε:

$$(A \to \alpha.B\gamma,\ a) \xrightarrow{\varepsilon} (B \to .\beta,\ b)$$

When B is a non-terminal and for every rule $B \to \beta$ and $b \in \mathrm{FIRST}(\gamma a)$.

$LR(1)$ Automata

- Actions *shift* and *go* as for $LR(0)$ and $SLR(1)$
- Add *reduce* actions $A \to \alpha$ when the state has a complete item

$$(A \to \alpha., a)$$

and the *next token* is a.