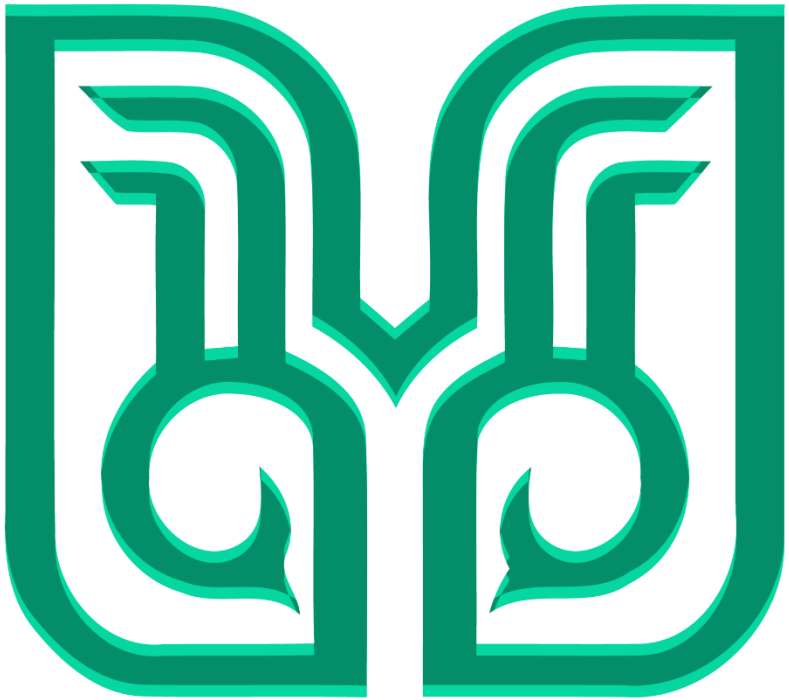


Compiler Design Project

Part II

Parsing with LR(1)

Presented By:

Ardalan Soltanzadeh

400407014

Sepehr Bazmi

400405019

Dr. Fatemeh Yousefi

LR(1)

LR(1) parsing is a bottom-up parsing technique used to analyze and validate the syntax of a programming language or any other context-free language. The "LR" stands for "left-to-right" scanning of the input string and "rightmost derivation" of the parse tree, while the "(1)" indicates that the parser considers one lookahead symbol to make parsing decisions.

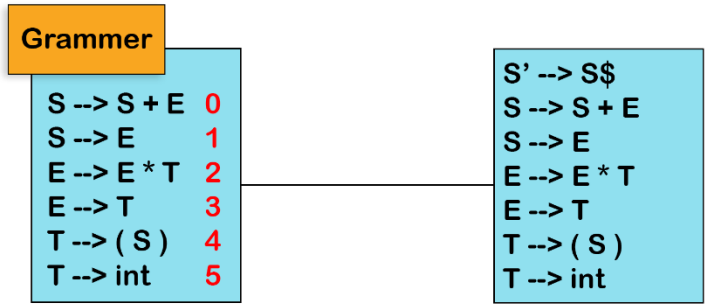
The LR(1) parser builds a deterministic finite automaton (DFA) known as an LR(1) automaton or LR(1) item set. This automaton represents the state transitions that occur during the parsing process. The DFA is constructed based on a set of augmented grammar rules and their corresponding LR(1) items.

An LR(1) item is a production rule with a dot (.) marker that represents the current position in the production. It also includes a lookahead symbol, which is the symbol that immediately follows the dot.

The LR(1) DFA is then used during the parsing process to perform shift and reduce actions. Shift actions correspond to moving to the next state based on the current input symbol, while reduce actions involve applying a production rule and replacing a group of symbols with a non-terminal.

Overall, the LR(1) parser uses the LR(1) DFA to determine the appropriate parsing actions based on the current state and the lookahead symbol, allowing it to build a valid parse tree for the input string.

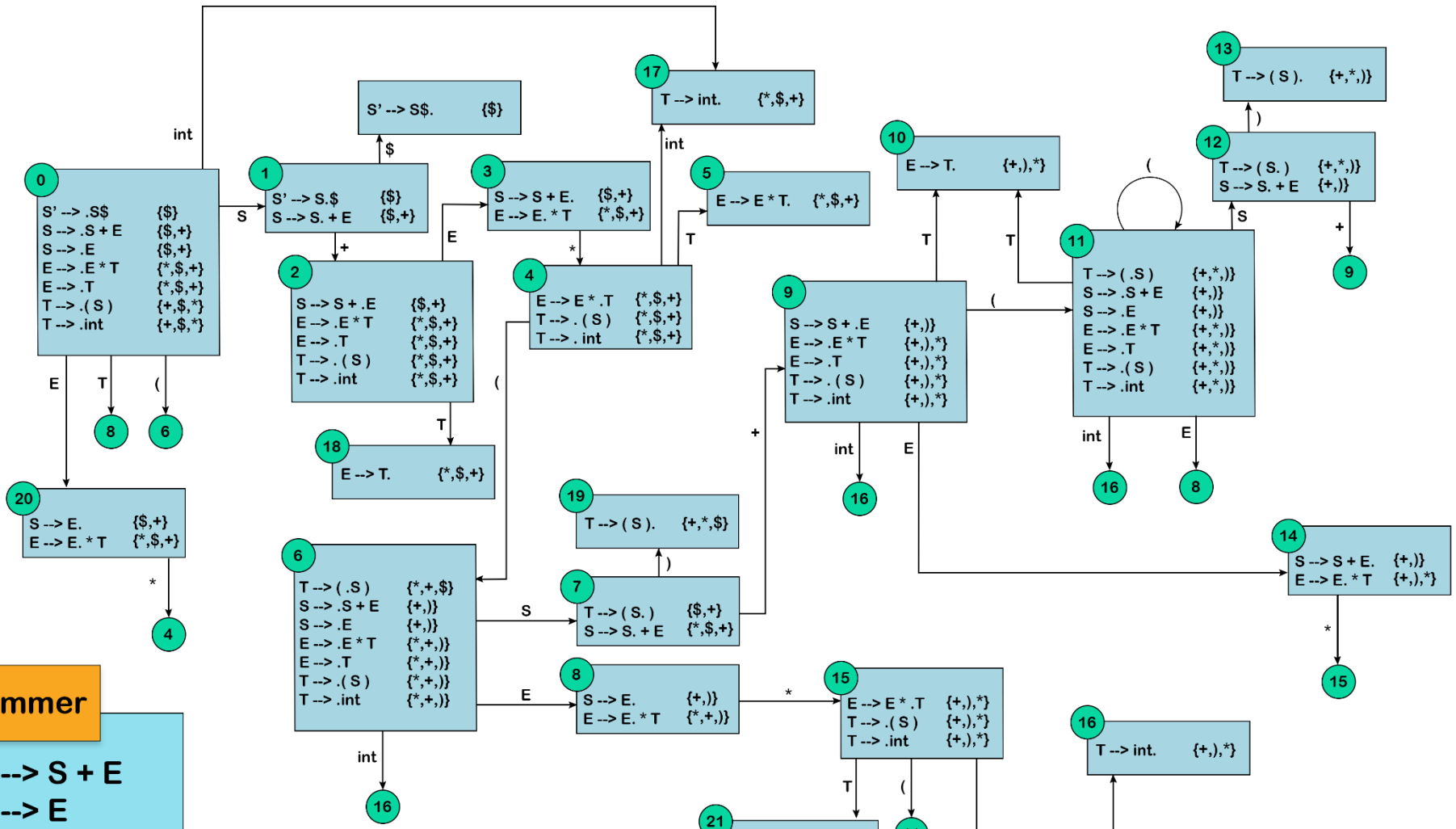
We start by adding S' → S$ to the grammar and create the DFA using the new grammar:



LR(1) DFA consists of a set of states, transitions, and an initial state. Each state represents a set of LR(1) items, and the transitions between states are determined by the symbols that follow the dot in the LR(1) items.

We start the first state with S' → .S$ and start creating our way through the end.

LR(1)



Grammer

S --> S + E
S --> E
E --> E * T
E --> T
T --> (S)
T --> int

Converting an LR(1) DFA to a parsing table is an essential step in building an LR(1) parser. The parsing table provides a concise representation of the DFA's state transitions and the corresponding parsing actions to be taken based on the current state and lookahead symbol :

	+	*	int	(	)	$	S	E	T
0	-	-	S17	S6	-	-	1	20	18
1	S2	-	-	-	-	accept	-	-	-
2	-	-	S17	S6	-	-	-	3	18
3	R0	S4	-	-	-	R0	-	-	-
4	-	-	S17	S6	-	-	-	-	5
5	R2	R2	-	-	-	R2	-	-	-
6	-	-	S16	S11	-	-	7	8	10
7	S9	-	-	-	S19	-	-	-	-
8	R1	S15	-	-	R1	-	-	-	-
9	-	-	S16	S11	-	-	-	14	10
10	R3	R3	-	-	R3	-	-	-	-
11	-	-	S16	S11	-	-	12	8	10
12	S9	-	-	-	S13	-	-	-	-
13	R4	R4	-	-	R4	-	-	-	-
14	R0	S15	-	-	R0	-	-	-	-
15	-	-	S16	S11	-	-	-	-	21
16	R5	R5	-	-	R5	-	-	-	-
17	R5	R5	-	-	-	R5	-	-	-
18	R3	R3	-	-	-	R3	-	-	-
19	R4	R4	-	-	-	R4	-	-	-
20	R1	S4	-	-	-	R1	-	-	-
21	R2	R2	-	-	R2	-	-	-	-

string	stack	action
int + int * int $	0	[0,int] = S17
+ int * int $	0 int 17	[17,+] = R5
+ int * int $	0 T	[0,T] = 18
+ int * int $	0 T 18	[18,+] = R3
+ int * int $	0 E	[0,E] = 20
+ int * int $	0 E 20	[20,+] = R1
+ int * int $	0 S	[0,S] = 1
+ int * int $	0 S1	[1,+] = S2
int * int $	0 S1 + 2	[2,int] = S17
* int $	0 S1 + 2 int 17	[17,*] = R5
* int $	0 S1 + 2 T	[2,T] = 18
* int $	0 S1 + 2 T 18	[18,*] = R3
* int $	0 S1 + 2 E	[2,E] = 3
* int $	0 S1 + 2 E3	[3,*] = S4
int $	0 S1 + 2 E3 * 4	[4,int] = S17
$	0 S1 + 2 E3 * 4 int 17	[17,$} = R5
$	0 S1 + 2 E3 * 4 T	[4,T] = 5
$	0 S1 + 2 E3 * 4 T5	[5,$] = R2
$	0 S1 + 2 E	[2,E] = 3
$	0 S1 + 2 E3	[3,$] = R0
$	0 S	[0,S] = 1
$	0 S1	[1,$] = accept