LR(1) Parser

LR → left-right
1 → prediction of lenght 1

- reduces the possibility of conflicts
- works with elements of the form $[A \to \alpha . \beta, u]$ (u: prediction (lookahead))

LR(1) steps:

- build canonical collection of states (similar with the one for other parsers, but this time we keep track of the prediction)
- build LR(1) table
- parsing using the table

Example

$S \to AA$ (1)
$A \to aA$ (2)
$A \to b$ (3)

Augment the grammar →

$S' \to S$
$S \to AA$ (1)
$A \to aA$ (2)
$A \to b$ (3)

Canonical collection:

$S' \to .S$ will always have \$ as prediction (lookahead)

I_0: $[S' \to .S, \$]$, $[S \to .AA, \$]$, $[A \to .aA, a/b]$, $[A \to .b, a/b]$

First($\varepsilon\$$) = First(\$) = \$
First($A\$$) = First(A) = {a, b}

I_1 (S): $[S' \to S., \$]$ — inherited from previous state
I_2 (A): $[S \to A.A, \$]$, $[A \to .aA, \$]$, $[A \to .b, \$]$ — inherited; First($\varepsilon\$$) = \$
I_3 (a): $[A \to a.A, a/b]$, $[A \to .aA, a/b]$, $[A \to .b, a/b]$
I_4 (b): $[A \to b., a/b]$
I_5 (A): $[S \to AA., \$]$
I_6 (a): $[A \to a.A, \$]$, $[A \to .aA, \$]$, $[A \to .b, \$]$
I_7 (b): $[A \to b., \$]$
I_8 (A): $[A \to aA., a/b]$
I_9 (A): $[A \to aA., \$]$

LR1 TABLE

State	Action			Goto	
	a	b	$	S	A
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		

s → shift
r → reduce
acc → accept

→ reduces are fill in only on the columns with corresponding prediction

ex: [A → b ⊙ a | b] s4

reduce — fill in on column a and b

Parsing

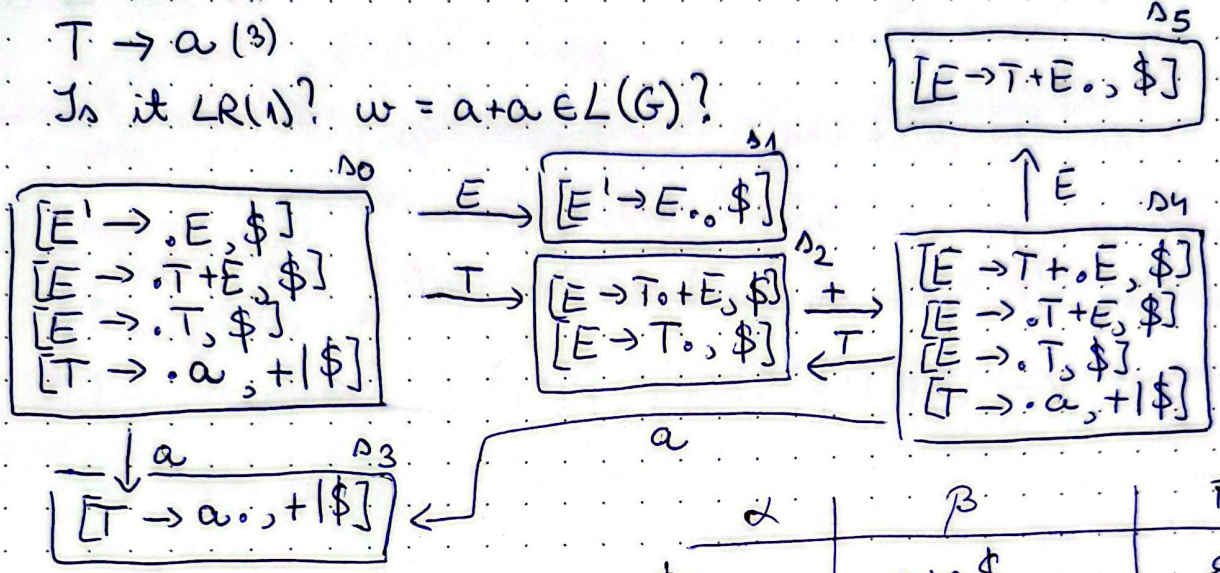
w = abb

α	β	π
$0	abb$	ε
$0a3	bb$	ε
$0a3b4	b$	ε
$0a3A8	b$	3
$0A2	b$	23
$0A2b7	$	23
$0A2A5	$	323
$051	$	1323
$acc		

$S \xrightarrow{1} AA \xrightarrow{3} Ab \xrightarrow{2} aAb \xrightarrow{3} abb$

Ex 2 LR(1)

$E' \to E$
$E \to T+E$ (1)
$E \to T$ (2)
$T \to a$ (3)
Is it LR(1)? $w = a+a \in L(G)$?



α	β	π
$0	a+a$	ε
$0a3	+a$	ε
$0T2	+a$	3
$0T2+4	a$	3
$0T2+4a3	$	3
$0T2+4T2	$	33
$0T2+4E5	$	233
$0E1	$	1233
$ acc	$	1233

State	Action			Goto	
	a	+	$	E	T
0	s3			1	2
1			acc		
2		s4	r2		
3		r3	r3		
4	s3			5	2
5			r1		

Attribute grammars

$AG = (G, A, R)$ where G is a CFG

$A = \{A(x) \mid x \in N \cup \Sigma\}$ is a finite set of attributes

$R = \{R(p) \mid p \in P\}$ is a finite set of rules to compute attributes

Ex 1. Given the following CFG create an attribute grammar that will compute the decimal value of a binary number. Show how attributes are evaluated / computed for 101.

P: $N \to BN$ $\quad G = (\{B, N\}, \{0, 1\}, P, N)$

$N \to B$

$B \to 0$

$B \to 1$

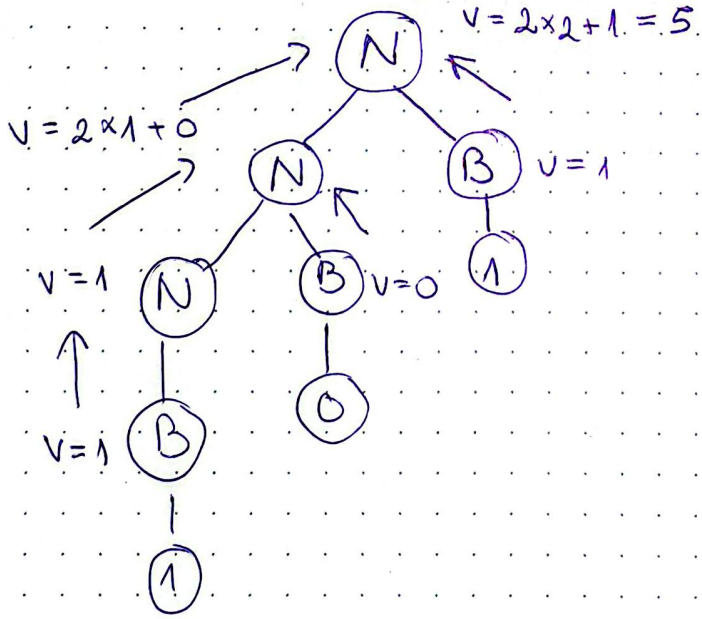
Solution

$N_1 \to N_2 B \quad \{N_1.v = 2 \times N_2.v + B.v\}$

$N \to B \quad \{N.v = B.v\}$

$B \to 0 \quad \{B.v = 0\}$

$B \to 1 \quad \{B.v = 1\}$



2. Build an AG that will compute the numerical value of an arithmetic expression. Apply your grammar for (3+5)*2-6.

Solution

$exp_3 \rightarrow exp_2 + term$ { $exp_3.val = exp_2.val + term.val$ }

$exp_1 \rightarrow exp_2 - term$ { $exp_1.val = exp_2.val - term.val$ }

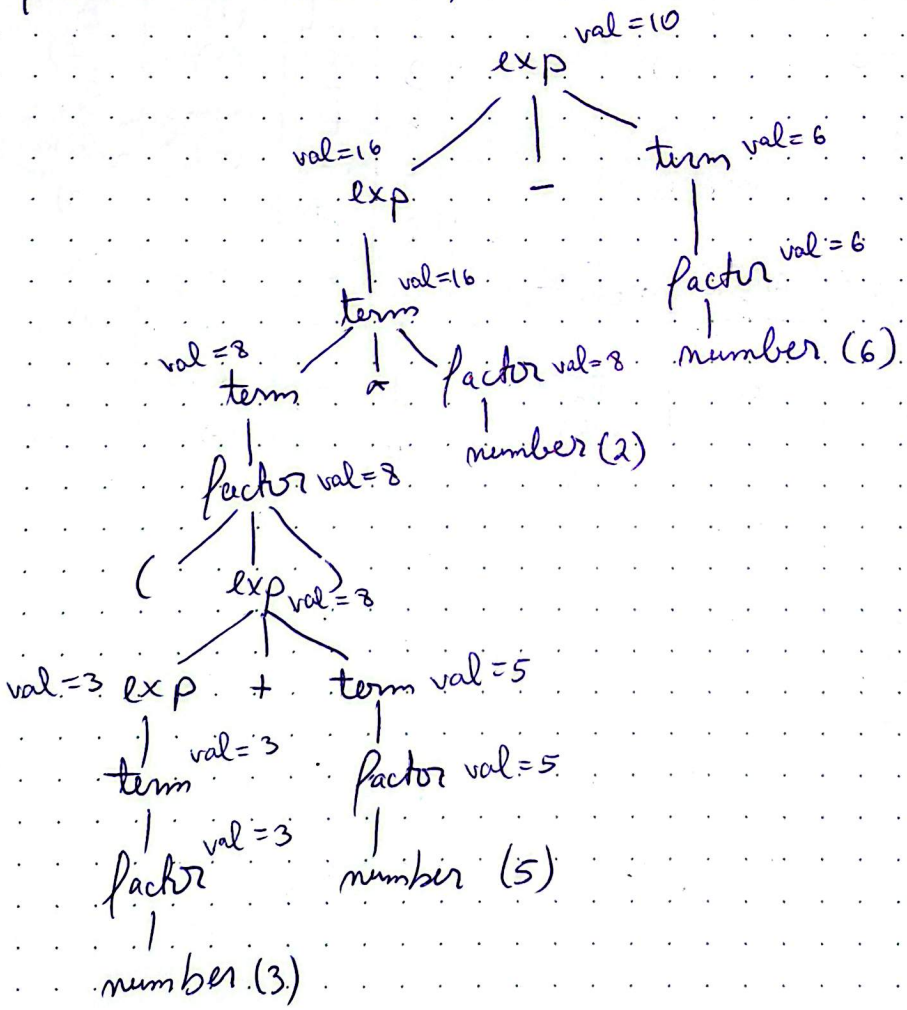
$exp \rightarrow term$ { $exp.val = term.val$ }

$term_1 \rightarrow term_2 * factor$ { $term_1.val = term_2.val * factor.val$ }

$term \rightarrow factor$ { $term.val \rightarrow factor.val$ }

$factor \rightarrow (exp)$ { $factor.val \rightarrow exp.val$ }

$factor \rightarrow number$ { $factr.val = number$ }



Intermediary code

- 3 address code
  - quadruples <op><arg1><arg2><result>
  - triples <op><arg1><arg2>

① Write the 3 address code using both quadruples and triples for:

```
if not a and b then
    sum := sum + a
else
    sum := sum + b
end if
sum := sum - 1
```

- quadruples

No	op	arg1	arg2	result
0	not	a		T1
1	and	T1	b	T2
2	if	T2		(6)
3	+	sum	b	T3
4	:=	T3		sum
5	goto			(8)
6	+	sum	a	T4
7	:=	T4		sum
8	-	sum	1	T5
9	:=	T5		sum

temporary variables to store results

jump to line 6 if if statement is evaluated as true

line 3 and 4 are if false

this is a convention, to have the false branch below if evaluation and jump only if true

will jump out of if-else block

- triples

No	op	arg1	arg2
0	not	a	
1	and	(0)	b
2	if	(1)	(6)
3	+	sum	b
4	:=	sum	(3)
5	goto		(8)
6	+	sum	a
7	:=	sum	(6)
8	-	sum	1
9	:=	sum	(8)

jump

jump

→ don't have a column for result
→ to use a result we refer it as the line where its computation was made
ex 1 and (0) b
↑ result from line 0

② Using either quadruples or triples provide the 3-address code for:

a)
```
while (i<10)
    c = c+1
endwhile
```

b)
```
repeat
    i = i+1
until (i>=10)
```

c)
```
for (i=0; i<10; i=i+1)
    x = x+i
```

d)
```
i = 1
while (i<=n) and (i<>b)
    a := b+2*c
    i := i+1
endwhile
b := 0
```

a)

No	op	arg1	arg2	result
0	<	i	10	T1
1	if	T1		(3)
2	goto			(6)
3	+	i	1	T2
4	:=	T2		i
5	goto			(0)
6				

or

No	op	arg1	arg2	result
0	>=	i	10	T1
1	if	T1		(5)
2	+	i	1	T2
3	:=	T2		i
4	goto			(0)
5				

b)

No	op	arg1	arg2	result
0	+	i	1	T1
1	:=	T1		i
2	<	i	10	T2
3	if	T2		(0)

c)

No	op	arg1	arg2	result
0	=	0		i
1	<	i	10	T1
2	if	T1		(4)
3	goto			(9)
4	+	x	i	T2
5	:=	T2		x
6	+	i	1	T3
7	:=	T3		i
8	goto			(1)
9				

Push-down automata (PDA)

7 tuples $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

Q - finite set of states

Σ - alphabet (finite set of input)

$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to P(Q \times \Gamma^*)$ tranition function

Γ - stack alphabet (finite set of stack symbols)

$q_0 \in Q$ initial state

$Z_0 \in \Gamma$ initial stack symbol

$F \subseteq Q$ set of final states

- config: $(q, x, \alpha) \in Q \times \Sigma^* \times \Gamma^*$

  PDA is in state q, input contains x, head of stack is α

- init config: (q_0, w, Z_0)

Ex: Design PDA for accepting:

1. $L = \{0^n 1^{2n} \mid n >= 0\}$
2. $L = \{0^{2n} 1^n \mid n \geq 0\}$
3. $L = \{0^n 1^m 2^n \mid m, n >= 1\}$
4. $L = \{0^n 1^m \mid m, n >= 0,\ m > n\}$
5. $L = \{$ all sequences of matching paranthesis $\}$
6. $L = \{ww^R \mid w \in \{a,b\}^+\}$ R - means reverse

1. $L = \{0^n 1^{2n} \mid n \geq 0\}$

Idea: to have two 1s for each 0s we'll push two symbols each time we read 0 and pop one symbol for reading 1

- Enumeration representation

what I add to the stack

$\delta(q_0, 0, Z_0) = (q_0, XXZ_0)$

state | input to read | head of stack | state where to go

$\delta(q_0, 0, X) = (q_0, XXX)$

} what happens when reading 0s

$\delta(q_0, 1, X) = (q_1, \varepsilon)$

ε means I will pop one symbol from stack = the current head

$\delta(q_1, 1, X) = (q_1, \varepsilon)$

} what happens when reading 1s

$\delta(q_1, \varepsilon, Z_0) = (q_f, Z_0)$ ← Accept criteria when input is empty (ε) and head of stack is Z_0

don't push or pop anything

$\delta(q_0, \varepsilon, Z_0) = (q_f, Z_0)$ ← Accept the empty string ($n=0$)

$Q = \{q_0, q_1, q_f\}$

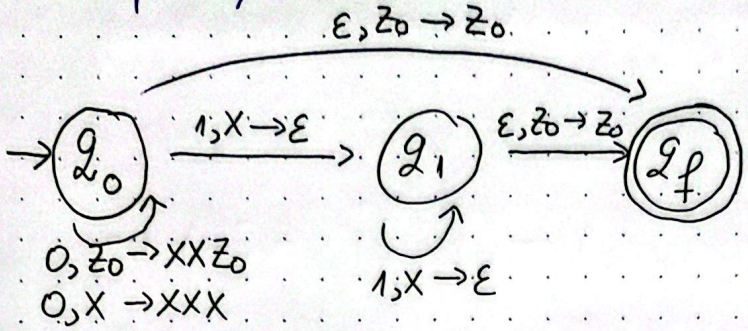
$\Sigma = \{0, 1\}$

$\Gamma = \{Z_0, X\}$

$F = \{q_f\}$

Z_0 = initial stack symbol

• Graph representation



② $L = \{0^{2n}1^n \mid n \geq 0\}$

Idea: for every two 0s push one symbol to stack, for each 1 pop one symbol

$Q = \{q_0, q_1, q_2, q_f\}$, $\Gamma = \{A, z_0\}$ $F = \{q_f\}$

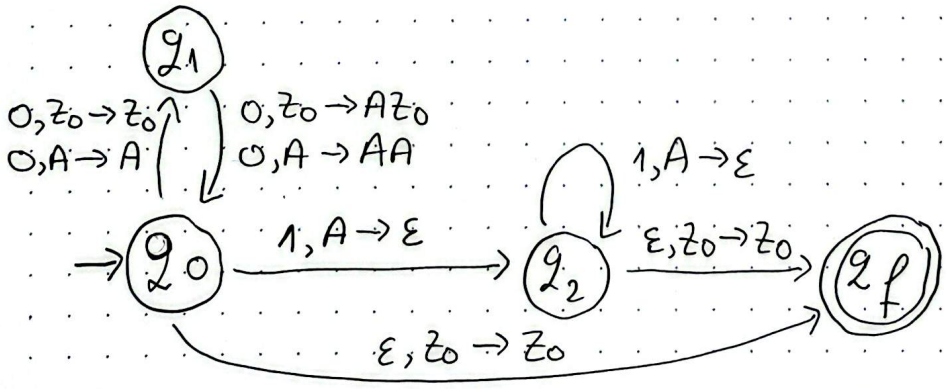
initial state: q_0, initial stack symbol: z_0

$\delta(q_0, 0, z_0) = (q_1, z_0)$
$\delta(q_0, 0, A) = (q_1, A)$ } reading first 0 of a pair

$\delta(q_1, 0, z_0) = (q_0, Az_0)$
$\delta(q_1, 0, A) = (q_0, AA)$ } for reading second 0 of a pair

$\delta(q_0, 1, A) = (q_2, \varepsilon)$
$\delta(q_2, 1, A) = (q_2, \varepsilon)$ } for reading 1

$\delta(q_2, \varepsilon, z_0) = (q_f, z_0)$
$\delta(q_0, \varepsilon, z_0) = (q_f, z_0)$ } for accepting



Using a PDA for checking a certain word acceptance example

$w = 001$

$(q_0, 001, Z_0) \vdash (q_1, 01, Z_0) \vdash (q_0, 1, AZ_0) \vdash$

$\vdash (q_2, \varepsilon, Z_0) \vdash (q_f, \varepsilon, Z_0) \Rightarrow w$ is accepted

③ $L = \{0^n 1^m 2^n \mid n, m \geq 1\}$

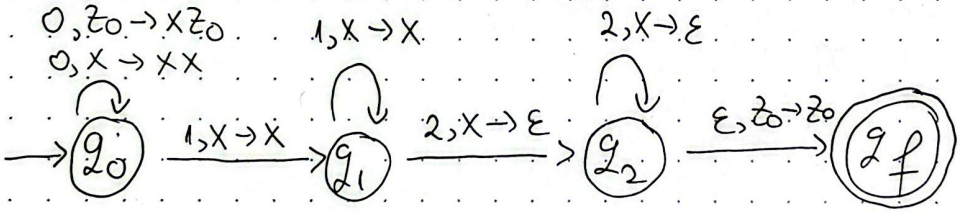
• Idea: for each 0 push a symbol and pop a symbol for each 2

$\delta(q_0, 0, Z_0) = (q_0, XZ_0)$ $\quad \delta(q_1, 2, X) = (q_2, \varepsilon)$

$\delta(q_0, 0, X) = (q_0, XX)$ $\quad \delta(q_2, 2, X) = (q_2, \varepsilon)$

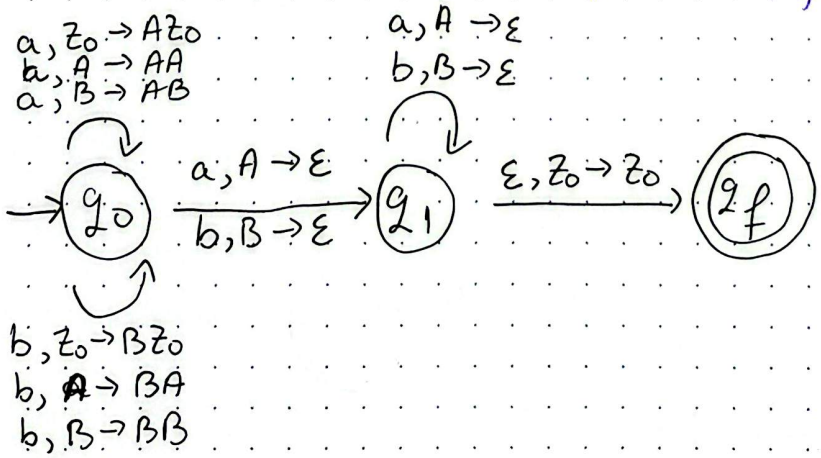
$\delta(q_0, 1, X) = (q_1, X)$ $\quad \delta(q_2, \varepsilon, Z_0) = (q_f, Z_0)$

$\delta(q_1, 1, X) = (q_1, X)$



⑥ $L = \{ww^R \mid w \in \{a,b\}^+\}$ (check if a sequence of a's and b's is palindrome)

$\Gamma = \{A, B, Z_0\}$; $Q = \{q_0, q_1, q_f\}$ $F = \{q_f\}$



- this a non deterministic PDA because from q_0 with the same input and head of stack there are two possible transitions
- this language cannot be represented with a deterministic PDA
- a sequence is accepted by a non-deterministic PDA if there exists at least one path in the graph that will reach each symbol from the input and reach the final state with an empty stack (having only Z_0)

- In a PDA you can push multiple symbols at the same time if you need (as in the examples before), but it is allowed to pop only one symbol at a time.