Notes on LR Parsing

Geoffrey Matthews

Department of Computer Science
Western Washington University

November 19, 2018

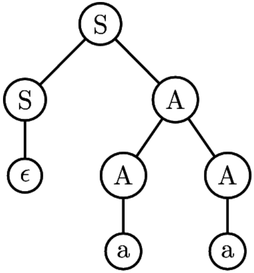
Readings

- http://www.cs.rochester.edu/~nelson/courses/csc_173/grammars/cfg.html
- http://en.wikipedia.org/wiki/Context-free_grammar
- http://en.wikipedia.org/wiki/Context-free_language
- http://en.wikipedia.org/wiki/Parsing
- http://en.wikipedia.org/wiki/Pushdown_automata
- http://en.wikipedia.org/wiki/LR_parser
- https://parasol.tamu.edu/~rwerger/Courses/434/lec12-sum.pdf

Bottom up parsing of CFGs

- We start with the input and attempt to build the parse tree.
- If we begin with the input and attempt to build the tree above it, we are doing **bottom-up** parsing.
- Equivalently, we try to constuct a rightmost derivation from right to left, scanning the input left to right.

$$\begin{aligned} S &\to SA \mid \epsilon \\ A &\to AA \mid a \end{aligned}$$



$$S \overset{S\to SA}{\Longrightarrow} SA \overset{A\to AA}{\Longrightarrow} SAA \overset{A\to a}{\Longrightarrow} SAa \overset{A\to a}{\Longrightarrow} Saa \overset{S\to\epsilon}{\Longrightarrow} aa$$

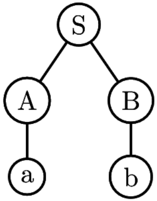
$LR(k)$ grammars

- $LR(k)$ means we find a rightmost derivation by scanning the input left to right, and have to lookahead at most k symbols.

LR parsing: Shift and Reduce

Shift:	move character from input to stack
Reduce:	if stack holds RHS of a rule, replace with LHS

$$\begin{aligned} S &\to AB \\ A &\to a \\ B &\to b \end{aligned}$$



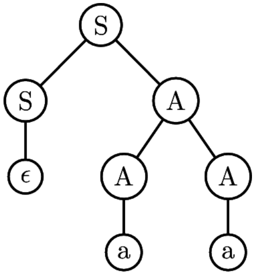
Stack	Input	Rule
\$	ab\$	shift
\$a	b\$	A→a
\$A	b\$	shift
\$Ab	\$	B→b
\$AB	\$	S→AB
\$S	\$	accept

$$S \overset{S\to AB}{\Longrightarrow} AB \overset{B\to b}{\Longrightarrow} Ab \overset{A\to a}{\Longrightarrow} ab$$

- Note: At all times, stack+input=derivation string

LR parsing: Shift and Reduce

$$\begin{aligned} S &\rightarrow SA \mid \epsilon \\ A &\rightarrow AA \mid a \end{aligned}$$

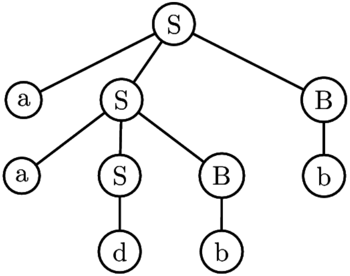


Stack	Input	Rule
\$	aa\$	$S \rightarrow \epsilon$
\$S	aa\$	shift
\$Sa	a\$	$A \rightarrow a$
\$SA	a\$	shift
\$SAa	\$	$A \rightarrow a$
\$SAA	\$	$A \rightarrow AA$
\$SA	\$	$S \rightarrow SA$
\$S	\$	accept

$$S \overset{S\rightarrow SA}{\Longrightarrow} SA \overset{A\rightarrow AA}{\Longrightarrow} SAA \overset{A\rightarrow a}{\Longrightarrow} SAa \overset{A\rightarrow a}{\Longrightarrow} Saa \overset{S\rightarrow \epsilon}{\Longrightarrow} aa$$

Another *LR* parse

$$
\begin{aligned}
S &\rightarrow aSB \mid d \\
B &\rightarrow b
\end{aligned}
$$



Stack	Input	Rule
\$	aadbb\$	shift
\$a	adbb\$	shift
\$aa	dbb\$	shift
\$aad	bb\$	$S \rightarrow d$
\$aaS	bb\$	shift
\$aaSb	b\$	$B \rightarrow b$
\$aaSB	b\$	$S \rightarrow aSB$
\$aS	b\$	shift
\$aSb	\$	$B \rightarrow b$
\$aSB	\$	$S \rightarrow aSB$
\$S	\$	accept

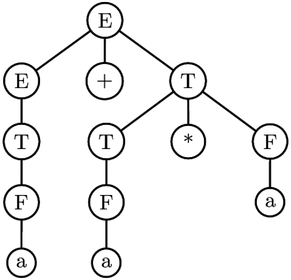
$$S \overset{S\rightarrow aSB}{\Longrightarrow} aSB \overset{B\rightarrow b}{\Longrightarrow} aSb \overset{S\rightarrow aSB}{\Longrightarrow} aaSBb \overset{B\rightarrow b}{\Longrightarrow} aaSbb \overset{S\rightarrow d}{\Longrightarrow} aadbb$$

LR parsing arithmetic

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid a \end{aligned}$$

$$\begin{aligned} E &\Rightarrow E + T \\ &\Rightarrow E + T * F \\ &\Rightarrow E + T * a \\ &\Rightarrow E + F * a \\ &\Rightarrow E + a * a \\ &\Rightarrow T + a * a \\ &\Rightarrow F + a * a \\ &\Rightarrow a + a * a \end{aligned}$$

Stack	Input	Rule
$	a+a*a$	shift
$a	+a*a$	F→a
$F	+a*a$	T→F
$T	+a*a$	E→T
$E	+a*a$	shift
$E+	a*a$	shift
$E+a	*a$	F→a
$E+F	*a$	T→F
$E+T	*a$	shift
$E+T*	a$	shift
$E+T*a	$	F→a
$E+T*F	$	T→T*F
$E+T	$	E→E+T
$E	$	accept



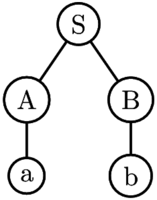
LR(1) parsing

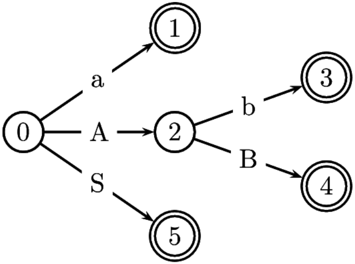
- The trick is to know when to shift and when to reduce.
- Hopefully by looking at **only one** symbol of the input.
- Everything on the stack has already been examined.
- We can use the entire stack to determine actions.
- We do this by using a DFA to keep track of stack state.
- We note each time a RHS appears on top of the stack.
- If a RHS is on top of the stack, a reduction is *possible*.
  - We can then choose whether to shift or reduce.
  - Otherwise you must shift.

$LR(1)$ parsing

$$\begin{aligned} S &\to AB \\ A &\to a \\ B &\to b \end{aligned}$$

Stack	Input	Rule
\$	ab\$	shift
\$a	b\$	A→a
\$A	b\$	shift
\$Ab	\$	B→b
\$AB	\$	S→AB
\$S	\$	accept



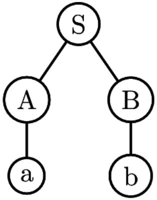


$$S \overset{S\to AB}{\Longrightarrow} AB \overset{B\to b}{\Longrightarrow} Ab \overset{A\to a}{\Longrightarrow} ab$$

We will store the state of the DFA on the stack, too.

$LR(1)$ parsing

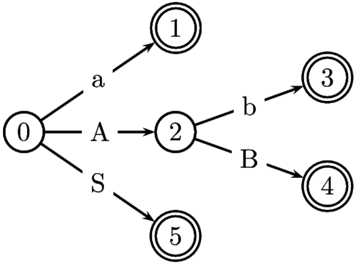
$$\begin{aligned} S &\to AB \\ A &\to a \\ B &\to b \end{aligned}$$



Stack	Input	Rule
0	ab$	shift
0 a 1	b$	A→a
0 A 2	b$	shift
0 A 2 b 3	$	B→b
0 A 2 B 4	$	S→AB
0 S 5	$	accept

$$S \xRightarrow{S\to AB} AB \xRightarrow{B\to b} Ab \xRightarrow{A\to a} ab$$

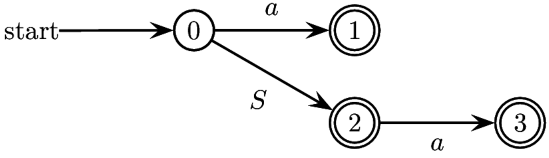
	a	b	A	B	S	$
0	1		2		5	
1		$A \to a$				
2		3		4		
3						$B \to b$
4						$S \to AB$
5						accept



Left recursion: $S \rightarrow Sa \mid a$

Stack	Input
0	a a a a $
0 a 1	a a a $
0 S 2	a a a $
0 S 2 a 3	a a $
0 S 2	a a $
0 S 2 a 3	a $
0 S 2	a $
0 S 2 a 3	$
0 S 2	$

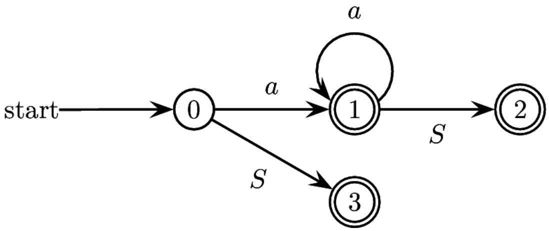
	a	$	S
0	1		2
1	$S \rightarrow a$		
2	3	accept	
3	$S \rightarrow Sa$	$S \rightarrow Sa$	



Right recursion: $S \to aS \mid a$

Stack	Input
0	a a a a $
0 a 1	a a a $
0 a 1 a 1	a a $
0 a 1 a 1 a 1	a $
0 a 1 a 1 a 1 a 1	$
0 a 1 a 1 a 1 S 2	$
0 a 1 a 1 S 2	$
0 a 1 S 2	$
0 S 3	$

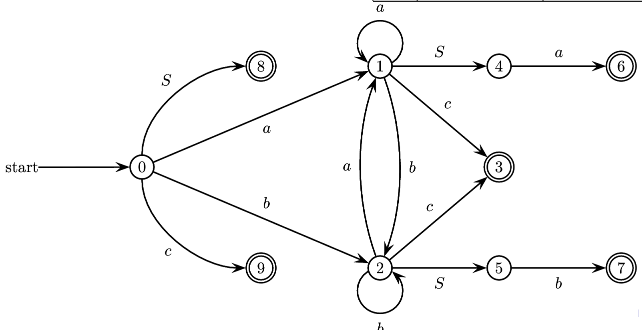
	a	$	S
0	1		3
1	1	$S \to a$	2
2		$S \to aS$	
3		accept	



Middle recursion: $S \rightarrow aSa \mid bSb \mid c$

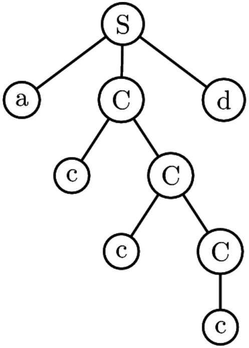
Stack	Input
0	a b c b a $
0 a 1	b c b a $
0 a 1 b 2	c b a $
0 a 1 b 2 c 3	b a $
0 a 1 b 2 S 5	b a $
0 a 1 b 2 S 5 b 7	a $
0 a 1 S 4	a $
0 a 1 S 4 a 6	$
0 S 8	$

	a	b	c	$	S
0	1	2	9		8
1	1	2	3		4
2	1	2	3		5
3	$S \rightarrow c$	$S \rightarrow c$			
4	6				
5		7			
6	$S \rightarrow aSa$	$S \rightarrow aSa$		$S \rightarrow aSa$	
7	$S \rightarrow bSb$	$S \rightarrow bSb$		$S \rightarrow bSb$	
8				accept	
9				$S \rightarrow c$	



LR(1) parsing, a more complex example

$$\begin{aligned} S &\rightarrow aCd \mid bCD \\ C &\rightarrow cC \mid c \\ D &\rightarrow d \end{aligned}$$

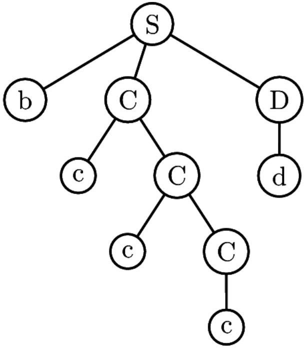


Stack	Input	Rule
\$	acccd\$	shift
\$a	cccd\$	shift
\$ac	ccd\$	shift
\$acc	cd\$	shift
\$accc	d\$	C→c
\$accC	d\$	C→cC
\$acC	d\$	C→cC
\$aC	d\$	shift
\$aCd	\$	S →aCd
\$S	\$	accept

$$S \Rightarrow aCd \Rightarrow acCd \Rightarrow accCd \Rightarrow acccd$$

LR(1) parsing

$$\begin{aligned} S &\rightarrow aCd \mid bCD \\ C &\rightarrow cC \mid c \\ D &\rightarrow d \end{aligned}$$



Stack	Input	Rule
\$	bcccd\$	shift
\$b	cccd\$	shift
\$bc	ccd\$	shift
\$bcc	cd\$	shift
\$bccc	d\$	C →c
\$bccC	d\$	C →cC
\$bcC	d\$	C →cC
\$bC	d\$	shift
\$bCd	\$	D→d
\$bCD	\$	S→bCD
\$S	\$	accept

$$S \Rightarrow bCD \Rightarrow bCd \Rightarrow bcCd \Rightarrow bccCd \Rightarrow bcccd$$

LR(1) parsing

$$\begin{array}{rcl} S & \rightarrow & aCd \mid bCD \\ C & \rightarrow & cC \mid c \\ D & \rightarrow & d \end{array}$$

- $S \Rightarrow aCd \Rightarrow acCd \Rightarrow accCd \Rightarrow acccd$
- $S \Rightarrow bCD \Rightarrow bCd \Rightarrow bcCd \Rightarrow bccCd \Rightarrow bcccd$
- At any point, the derivation string must look like one of these:

  $aCd \quad ac^+Cd \quad ac^+d \quad bCD \quad bCd \quad bc^+Cd \quad bc^+d$

- Whenever we see one of these, we have to know which rule to apply at what point in the shifting of the string.

LR(1) parsing

$$\begin{aligned} S &\rightarrow aCd \mid bCD \\ C &\rightarrow cC \mid c \\ D &\rightarrow d \end{aligned}$$

Stack	Input	Rule	Peek
$\$aCd$	$\$$	S→aCd	
$\$ac^+C$	$d\$$	C →cC	
$\$ac^+$	$d\$$	C →c	d
$\$bCD$	$\$$	S→bCD	
$\$bCd$	$\$$	D→d	
$\$bc^+C$	$d\$$	C →cC	
$\$bc^+$	$d\$$	C →c	d

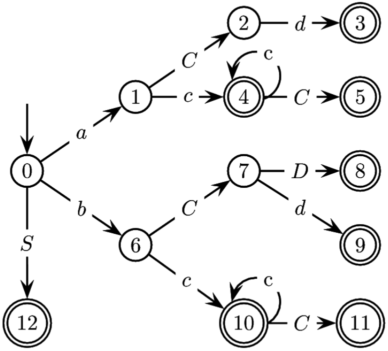
Stack	Input	Rule
$	acccd$	shift
$a	cccd$	shift
$ac	ccd$	shift
$acc	cd$	shift
$accc	d$	C→c
$accC	d$	C→cC
$acC	d$	C→cC
$aC	d$	shift
$aCd	$	S →aCd
$S	$	accept

Stack	Input	Rule
$	bcccd$	shift
$b	cccd$	shift
$bc	ccd$	shift
$bcc	cd$	shift
$bccc	d$	C →c
$bccC	d$	C →cC
$bcC	d$	C →cC
$bC	d$	shift
$bCd	$	D→d
$bCD	$	S→bCD
$S	$	accept

DFA for LR parsing

$$\begin{array}{lcl} S & \rightarrow & aCd \mid bCD \\ C & \rightarrow & cC \mid c \\ D & \rightarrow & d \end{array}$$

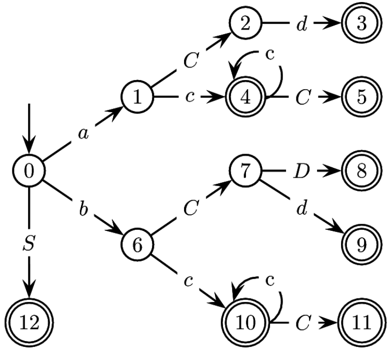
Stack	Input	Rule	Peek
$\$aCd$	\$	S→aCd	
$\$ac^+C$	$d\$$	C →cC	
$\$ac^+$	$d\$$	C →c	d
$\$bCD$	\$	S→bCD	
$\$bCd$	\$	D→d	
$\$bc^+C$	$d\$$	C →cC	
$\$bc^+$	$d\$$	C →c	d



Using the DFA in LR parsing

Stack	Input	Rule	Peek
$\$aCd$	$\$$	S→aCd	
$\$ac^+C$	$d\$$	C →cC	
$\$ac^+$	$d\$$	C →c	d
$\$bCD$	$\$$	S→bCD	
$\$bCd$	$\$$	D→d	
$\$bc^+C$	$d\$$	C →cC	
$\$bc^+$	$d\$$	C →c	d

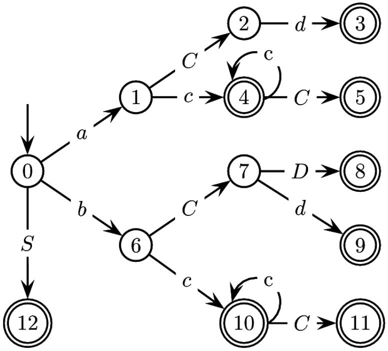
Stack	Input	Rule
0	acccd$	shift
0 a 1	cccd$	shift
0 a 1 c 4	ccd$	shift
0 a 1 c 4 c 4	cd$	shift
0 a 1 c 4 c 4 c 4	d$	C→c
0 a 1 c 4 c 4 C 5	d$	C→cC
0 a 1 c 4 C 5	d$	C→cC
0 a 1 C 2	d$	shift
0 a 1 C 2 d 3	$	S →aCd
0 S 12	$	accept



Using the DFA in LR parsing

Stack	Input	Rule	Peek
$\$aCd$	$	S→aCd	
$\$ac^+C$	$d\$$	C →cC	
$\$ac^+$	$d\$$	C →c	d
$\$bCD$	$	S→bCD	
$\$bCd$	$	D→d	
$\$bc^+C$	$d\$$	C →cC	
$\$bc^+$	$d\$$	C →c	d

Stack	Input	Rule
0	bcccd$	shift
0 b 6	cccd$	shift
0 b 6 c 10	ccd$	shift
0 b 6 c 10 c 10	cd$	shift
0 b 6 c 10 c 10 c 10	d$	C →c
0 b 6 c 10 c 10 C 11	d$	C →cC
0 b 6 c 10 C 11	d$	C →cC
0 b 6 C 7	d$	shift
0 b 6 C 7 d 9	$	D→d
0 b 6 C 7 D 8	$	S→bCD
0 S 12	$	accept



More examples in notes on repo.

LR(k) languages, Knuth's theorem

Theorem

$$\begin{aligned} \mathit{LR}(k)\ \mathit{languages} &= \mathit{LR}(1)\ \mathit{languages} \\ &= \mathit{deterministic\ context\ free\ languages} \end{aligned}$$

LR parsing exercises

Redo all the solved examples. Also, find DFAs and tables for the following languages, and trace some parses:

- $S \to a \mid b \mid c$
- $S \to aSa \mid b$
- $S \to ABC$
  $A \to a$
  $B \to b$
  $C \to c$

Lex and Yacc Style Parsers

- `http://epaperpress.com/lexandyacc/`
- `https://docs.racket-lang.org/parser-tools/index.html`