

# Context Free Languages and Pushdown Automata

Geoffrey Matthews

Department of Computer Science  
Western Washington University

February 27, 2018

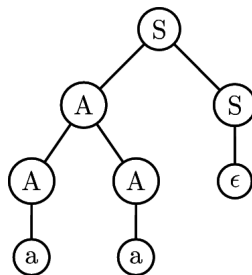
# Readings

- ▶ [http://www.cs.rochester.edu/~nelson/courses/csc\\_173/grammars/cfg.html](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_grammar)
- ▶ [http://en.wikipedia.org/wiki/Context-free\\_language](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/Pushdown_automata)
- ▶ [http://en.wikipedia.org/wiki/LR\\_parser](http://en.wikipedia.org/wiki/LR_parser)
- ▶ <https://parasol.tamu.edu/~rwerger/Courses/434/lec12-sum.pdf>

# Top-down Parsing of CFGs

- ▶ We start with the input and attempt to build the parse tree.
- ▶ If we begin with the start symbol and attempt to build the tree below it, we are doing **top-down** parsing.
- ▶ Equivalently, we try to construct a leftmost derivation from left to right.

$$\begin{array}{lcl} S & \rightarrow & AS \mid \epsilon \\ A & \rightarrow & AA \mid a \end{array}$$



$$S \xRightarrow{S \rightarrow AS} AS \xRightarrow{A \rightarrow AA} AAS \xRightarrow{A \rightarrow a} aAS \xRightarrow{A \rightarrow a} aaS \xRightarrow{S \rightarrow \epsilon} aa$$

# PDA for any grammar

- ▶ Start with  $\$S$  on the stack, pad input on the right with  $\$$
- ▶ Pop variables and replace them with a rule RHS
- ▶ Match terminals on the stack with terminals on the input
- ▶ When you find  $\$$  on both stack and input, accept

$$S \rightarrow aSc \mid b$$

*aabcc*

$$\begin{array}{lcl} S & \xRightarrow{S \rightarrow aSc} & aSc \\ & \xRightarrow{S \rightarrow aSc} & aaSc \\ & \xRightarrow{S \rightarrow b} & aabcc \end{array}$$

Stack	Input	Rule
$\$S$	aabcc $\$$	$S \rightarrow aSc$
$\$cSa$	aabcc $\$$	match
$\$cS$	abcc $\$$	$S \rightarrow aSc$
$\$ccSa$	abcc $\$$	match
$\$ccS$	bcc $\$$	$S \rightarrow b$
$\$ccb$	bcc $\$$	match
$\$cc$	cc $\$$	match
$\$c$	c $\$$	match
$\$$	$\$$	accept

# PDA for any grammar

$$S \rightarrow aSc \mid b$$

$$S \xRightarrow{S \rightarrow aSc} aSc \xRightarrow{S \rightarrow aSc} aaSc \xRightarrow{S \rightarrow b} aabcc$$

$$qa\$ \rightarrow qN\$S$$

$$qb\$ \rightarrow qN\$S$$

$$qaS \rightarrow qRaSc$$

$$qaS \rightarrow qRb$$

$$qbS \rightarrow qRaSc$$

$$qbS \rightarrow qRb$$

$$qaa \rightarrow qR\epsilon$$

$$qbb \rightarrow qR\epsilon$$

$$q\$\$ \rightarrow qR\epsilon$$

Stack	Input	Rule
\$	aabcc\$	push $S$
\$S	aabcc\$	$S \rightarrow aSc$
\$cSa	aabcc\$	match
\$cS	abcc\$	$S \rightarrow aSc$
\$ccSa	abcc\$	match
\$ccS	bcc\$	$S \rightarrow b$
\$ccb	bcc\$	match
\$cc	cc\$	match
\$c	c\$	match
\$	\$	match
		accept

# $LL(k)$ grammars: deterministic PDAs

- ▶ Start with  $S$  on the stack
- ▶ Pop variables and replace them with a rule RHS
- ▶ Match terminals on the stack with terminals on the input
- ▶ The trick is knowing *which rule to use* when we pop a variable
- ▶ If we can always determine this, the PDA will be deterministic
- ▶  $LL(k)$  means we find a leftmost derivation by scanning the input left to right, and have to lookahead at most  $k$  symbols.
- ▶ With  $LL(1)$ , we construct a **chart** showing, for any given (input,variable) pair, which rule to use.

# An $LL(1)$ grammar and chart for $a^nbc^n$

$$S \rightarrow aSc \mid b$$

$aabcc$

$$\begin{array}{lcl}
 S & \xRightarrow{S \rightarrow aSc} & aSc \\
 & \xRightarrow{S \rightarrow aSc} & aaSc \\
 & \xRightarrow{S \rightarrow b} & aabcc
 \end{array}$$

	a	b	c	\$
S	$S \rightarrow aSc$	$S \rightarrow b$		

Stack	Input	Rule
\$S	aabcc\$	$S \rightarrow aSc$
\$cSa	aabcc\$	match
\$cS	abcc\$	$S \rightarrow aSc$
\$ccSa	abcc\$	match
\$ccS	bcc\$	$S \rightarrow b$
\$ccb	bcc\$	match
\$cc	cc\$	match
\$c	c\$	match
\$	\$	accept

# Another $LL(1)$ grammar and chart

$$S \rightarrow ASb \mid C$$

$$A \rightarrow a$$

$$C \rightarrow cC \mid \epsilon$$

$$S \xRightarrow{S \rightarrow ASb} ASb$$

$$\xRightarrow{A \rightarrow a} aSb$$

$$\xRightarrow{S \rightarrow ASb} aASbb$$

$$\xRightarrow{A \rightarrow a} aaSbb$$

$$\xRightarrow{S \rightarrow C} aaCbb$$

$$\xRightarrow{C \rightarrow cC} aacCbb$$

$$\xRightarrow{C \rightarrow cC} aaccCbb$$

$$\xRightarrow{C \rightarrow \epsilon} aaccb$$

	a	b	c	\$
S	$S \rightarrow ASb$		$S \rightarrow C$	$S \rightarrow C$
A	$A \rightarrow a$			
C		$C \rightarrow \epsilon$	$C \rightarrow cC$	$C \rightarrow \epsilon$

Stack	Input	Rule
\$S	aaccbb\$	$S \rightarrow ASb$
\$bSA	aaccbb\$	$A \rightarrow a$
\$bSa	aaccbb\$	match
\$bS	accbb\$	$S \rightarrow ASb$
\$bbSA	accbb\$	$A \rightarrow a$
\$bbSa	accbb\$	match
\$bbS	ccbb\$	$S \rightarrow C$
\$bbC	ccbb\$	$C \rightarrow cC$
\$bbCc	ccbb\$	match
\$bbC	cbb\$	match
\$bbC	cbb\$	$C \rightarrow cC$
\$bbCc	cbb\$	match
\$bbC	bb\$	$C \rightarrow \epsilon$
\$bb	bb\$	match
\$b	b\$	match
\$	\$	accept



# Some grammars are not $LL(1)$

$$S \rightarrow AB$$

$$A \rightarrow aA|a$$

$$B \rightarrow bB|c$$

$$\begin{array}{lcl}
 S & \xRightarrow{S \rightarrow AB} & AB \\
 & \xRightarrow{A \rightarrow Aa} & aAB \\
 & \xRightarrow{A \rightarrow a} & aaB \\
 & \xRightarrow{B \rightarrow bB} & aabB \\
 & \xRightarrow{B \rightarrow c} & aabc
 \end{array}$$

Stack	Input	Rule
\$S	aabc\$	$S \rightarrow AB$
\$BA	aabc\$	$A \rightarrow aA$
\$BAa	aabc\$	match
\$BA	abc\$	$A \rightarrow a$
\$Ba	abc\$	match
\$B	bc\$	$B \rightarrow bB$
\$Bb	bc\$	match
\$B	c\$	$B \rightarrow c$
\$c	c\$	match
\$	\$	accept

	a	b	c	\$
S	$S \rightarrow AB$			
A	$A \rightarrow Aa \mid a$			
B		$B \rightarrow Bb$	$B \rightarrow c$	

# LL(1) parsing by recursive descent

$S \rightarrow aSc \mid b$

*aabcc*

$$\begin{array}{lcl} S & \xRightarrow{S \rightarrow aSc} & aSc \\ & \xRightarrow{S \rightarrow aSc} & aaSc \\ & \xRightarrow{S \rightarrow b} & aabcc \end{array}$$

	a	b	c	\$
S	$S \rightarrow aSc$	$S \rightarrow b$		

```
(define (S)
  (cond
    ((front? 'a)
     (displayln "S -> aSc")
     (match 'a)
     (S)
     (match 'c))
    ((front? 'b)
     (displayln "S -> b")
     (match 'b))
    (else
     (error))))
```

# LL(1) parsing by recursive descent

$S \rightarrow ASb \mid C$

$A \rightarrow a$

$C \rightarrow cC \mid \epsilon$

$S \xRightarrow{S \rightarrow ASb} ASb$

$\xRightarrow{A \rightarrow a} aSb$

$\xRightarrow{S \rightarrow ASb} aASbb$

$\xRightarrow{A \rightarrow a} aaSbb$

$\xRightarrow{S \rightarrow C} aaCbb$

$\xRightarrow{C \rightarrow cC} aacCbb$

$\xRightarrow{C \rightarrow cC} aaccCbb$

$\xRightarrow{C \rightarrow \epsilon} aaccbb$

```
(define (S2)
  (cond ((front? 'a)
        (A) (S2) (match 'b))
        ((front? 'c)
        (C))
        ((front? '$)
        (C))
        (else (error))))

(define (A)
  (cond ((front? 'a)
        (match 'a))
        (else (error))))

(define (C)
  (cond ((front? 'b) )
        ((front? 'c)
        (match 'c) (C))
        ((front? '$) )
        (else (error))))
```

	a	b	c	\$
S	$S \rightarrow ASb$		$S \rightarrow C$	$S \rightarrow C$
A	$A \rightarrow a$			
C		$C \rightarrow \epsilon$	$C \rightarrow cC$	$C \rightarrow \epsilon$

A non- $LL(1)$  grammar for  $a^m b^n c$ ,  $m \geq 1$  and  $n \geq 0$

$$S \rightarrow AB$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid c$$

$$\begin{array}{lcl} S & \xRightarrow{S \rightarrow AB} & AB \\ & \xRightarrow{A \rightarrow aA} & aAB \\ & \xRightarrow{A \rightarrow a} & aaB \\ & \xRightarrow{B \rightarrow bB} & aabB \\ & \xRightarrow{B \rightarrow bB} & aabbB \\ & \xRightarrow{B \rightarrow c} & aabbc \end{array}$$

- Need to see two letters to determine which production to use.

Sometimes we can find  $LL(1)$  to replace a non- $LL(1)$

$$S \rightarrow AB$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid c$$

$$S \rightarrow aAB$$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow bB \mid c$$

- ▶ Same language, different grammars.
- ▶  $LL(1)$  is a property of **grammars**, not **languages**.

An  $LL(k)$  grammar whose language has no  $LL(k - 1)$  grammar

$$S \rightarrow aSA \mid \epsilon$$

$$A \rightarrow a^{k-1}bS \mid c$$

$$S \xRightarrow{*} aaaAAA$$

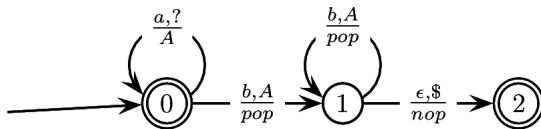
$$\xRightarrow{*} aaa \underbrace{aa \dots a}_{k-1} \overset{A}{bS} \underbrace{aa \dots a}_{k-1} \overset{A}{bS} \overset{A}{c}$$

# A deterministic language with no $LL(k)$ grammar

$$\{a^n \mid n \geq 0\} \cup \{a^n b^n \mid n \geq 0\}$$

$\underbrace{aaaaa \dots}_n$

$\underbrace{aaaaa \dots}_n \underbrace{bbbbbb \dots}_n$



## An $LL(1)$ grammar without terminals on RHS

$$S \rightarrow A \mid B$$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow bB \mid c$$

	a	b	c	\$
S	$S \rightarrow A$	$S \rightarrow B$	$S \rightarrow B$	$S \rightarrow A$
A	$A \rightarrow aA$			$A \rightarrow \epsilon$
B		$B \rightarrow bB$	$B \rightarrow c$	



## Transforming some $LL(k)$ to $LL(1)$ by Left-Factoring

$$S \rightarrow abcX \mid abcY$$

$$X \rightarrow xX \mid x$$

$$Y \rightarrow yY \mid y$$

$$S \rightarrow aB$$

$$B \rightarrow bC$$

$$C \rightarrow cX \mid cY$$

$$X \rightarrow xX \mid x$$

$$Y \rightarrow yY \mid y$$

# Left recursion is not $LL(k)$

$$A \rightarrow Aa \mid b$$

- ▶ The entire input must be read before we know how many times to apply the first rule, before we apply the second.

$A$	$\xRightarrow{A \rightarrow Aa}$	$Aa$
	$\xRightarrow{A \rightarrow Aa}$	$Aaa$
	$\xRightarrow{A \rightarrow Aa}$	$Aaaa$
	$\xRightarrow{A \rightarrow Aa}$	$\dots$
	$\xRightarrow{A \rightarrow Aa}$	$Aaaaaaaaaaaaaaaaaaaaaa$
	$\xRightarrow{A \rightarrow b}$	$baaaaaaaaaaaaaaaaaaaaaa$

## Immediate left recursion can be removed

$$A \rightarrow Aw \mid y$$

## Immediate left recursion can be removed

$$A \rightarrow Aw \mid y$$

- ▶ Any string in the language must start with a  $y$  followed by 0 or more  $w$ 's

## Immediate left recursion can be removed

$$A \rightarrow Aw \mid y$$

- ▶ Any string in the language must start with a  $y$  followed by 0 or more  $w$ 's

$$A \rightarrow yB$$

$$B \rightarrow wB \mid \epsilon$$

## More general immediate left recursion

$$A \rightarrow Aw_1 \mid Aw_2 \mid \dots \mid Aw_n \mid x_1 \mid x_2 \mid \dots \mid x_m$$

## More general immediate left recursion

$$A \rightarrow Aw_1 \mid Aw_2 \mid \dots \mid Aw_n \mid x_1 \mid x_2 \mid \dots \mid x_m$$

$$A \rightarrow x_1B \mid x_2B \mid \dots \mid x_mB$$

$$B \rightarrow w_1B \mid w_2B \mid \dots \mid w_nB \mid \epsilon$$

## Removing left recursion example

$$A \rightarrow Aa \mid b$$

$$A \rightarrow bB$$

$$B \rightarrow aB \mid \epsilon$$

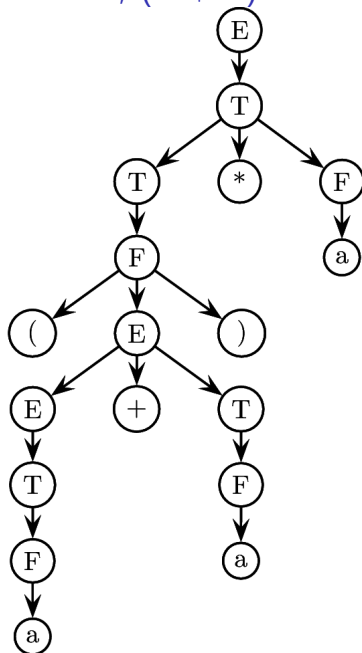


## Removing left recursion from arithmetic, $(a + a) * a$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid a$$



- ▶ Note we get the phrase structure of the expression reflected in the tree.
- ▶ What kind of tree do we get from  $a + a + a$ ?
- ▶ How could we make it right associative?

## Remove the left recursion

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid a$$

$$E \rightarrow TR$$

$$R \rightarrow +TR \mid \epsilon$$

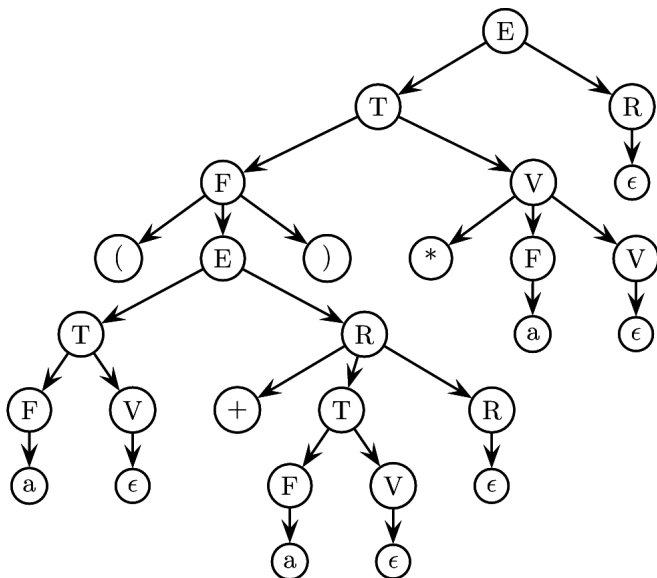
$$T \rightarrow FV$$

$$V \rightarrow *FV \mid \epsilon$$

$$F \rightarrow (E) \mid a$$

# What happened to the tree?

$E \rightarrow TR$   
 $R \rightarrow +TR \mid \epsilon$   
 $T \rightarrow FV$   
 $V \rightarrow *FV \mid \epsilon$   
 $F \rightarrow (E) \mid a$



# Indirect left recursion

$$S \rightarrow Bb$$

$$B \rightarrow Sa \mid a$$

$$S \Rightarrow Bb \Rightarrow Sab$$

## Removing indirect left recursion

$$S \rightarrow Bb$$

$$B \rightarrow Sa \mid a$$

Replace  $B$ 's RHS in  $S$ 's RHS:

$$S \rightarrow Sab \mid ab$$

Now remove the immediate left recursion

$$S \rightarrow abT$$

$$T \rightarrow abT \mid \epsilon$$

# Removing indirect left recursion

$$A \rightarrow Bb \mid e$$

$$B \rightarrow Cc \mid f$$

$$C \rightarrow Ad \mid g$$

1. Replace  $B$ 's RHS in  $A$ 's RHS:

$$A \rightarrow Ccv \mid fb \mid e$$

2. Now replace  $C$ 's RHS

$$A \rightarrow Adcb \mid gcb \mid fb \mid e$$

3. Now remove the immediate left recursion

$$A \rightarrow gcbD \mid fbD \mid eD$$

$$D \rightarrow dcbD \mid \epsilon$$

- This method can be applied generally, but the rules multiply.

# $LL(k)$ problems

Find an  $LL(1)$  grammar and parse table for the following languages

- ▶  $\{a, ba, bba\}$
- ▶  $a^*b$
- ▶  $a^{n+1}bc^n$
- ▶  $a^mb^nc^{m+n}$