Context Free Languages and Pushdown Automata

Geoffrey Matthews

Department of Computer Science
Western Washington University

May 22, 2015

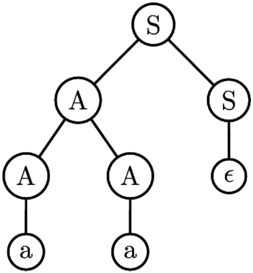
Readings

- 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

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 constuct a leftmost derivation from left to right.

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



$$S \Rightarrow AS \Rightarrow AAS \Rightarrow aAS \Rightarrow aaS \Rightarrow aa$$

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

$$S \to aSc \mid b$$

$$aabcc$$

$$\begin{aligned} S &\Rightarrow aSc \\ &\Rightarrow aaScc \\ &\Rightarrow aabcc \end{aligned}$$

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

PDA for any grammar

$$S \to aSc \mid b$$

$$\begin{array}{rcl}
qa\$ & \to & qN\$S \\
qb\$ & \to & qN\$S \\
qaS & \to & qRaSc \\
qaS & \to & qRb \\
qbS & \to & qRaSc \\
qbS & \to & qRb \\
qaa & \to & qR\varepsilon \\
qbb & \to & qR\varepsilon \\
q\$\$ & \to & qR\varepsilon
\end{array}$$

$$S \Rightarrow aSc \Rightarrow aaScc \Rightarrow aabcc$$

Stack	Input	Rule
\$	aabcc\$	push S
\$S	aabcc\$	$S \to aSc$
\$cSa	aabcc\$	match
\$cS	abcc\$	$S \to aSc$
\$ccSa	abcc\$	match
\$ccS	bcc\$	$S \to b$
\$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 $a^n bc^n$

$$S \to aSc \mid b$$

$aabcc$

$$\begin{aligned} S &\Rightarrow aSc \\ &\Rightarrow aaScc \\ &\Rightarrow aabcc \end{aligned}$$

	a	b	c	$
S	$S \to aSc$	$S \to b$		

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

Another $LL(1)$ grammar and chart

$$
\begin{array}{lcl}
S & \rightarrow & ASb \mid C \\
A & \rightarrow & a \\
C & \rightarrow & cC \mid \epsilon \\
S & \Rightarrow & ASb \\
 & \Rightarrow & aSb \\
 & \Rightarrow & aASbb \\
 & \Rightarrow & aaSbb \\
 & \Rightarrow & aaCbb \\
 & \Rightarrow & aacCbb \\
 & \Rightarrow & aaccCbb \\
 & \Rightarrow & aaccbb
\end{array}
$$

	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

$LL(1)$ parsing by recursive descent

$$S \to aSc \mid b$$

$$aabcc$$

$$\begin{aligned} S &\Rightarrow aSc \\ &\Rightarrow aaScc \\ &\Rightarrow aabcc \end{aligned}$$

	a	b	c	$
S	$S \to aSc$	$S \to b$		

```
(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

$$
\begin{aligned}
S &\to ASb \mid C \\
A &\to a \\
C &\to cC \mid \epsilon \\
S &\Rightarrow ASb \\
&\Rightarrow aSb \\
&\Rightarrow aASbb \\
&\Rightarrow aaSbb \\
&\Rightarrow aaCbb \\
&\Rightarrow aacCbb \\
&\Rightarrow aaccCbb \\
&\Rightarrow aaccbb
\end{aligned}
$$

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

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

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aA \mid a \\ B &\rightarrow bB \mid c \end{aligned}$$

$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow aAB \\ &\Rightarrow aaB \\ &\Rightarrow aabB \\ &\Rightarrow aabbB \\ &\Rightarrow aabbc \end{aligned}$$

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

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

$$\begin{array}{lcl} S & \rightarrow & AB \\ A & \rightarrow & aA \mid a \\ B & \rightarrow & bB \mid c \end{array}$$

$$\begin{array}{lcl} S & \rightarrow & aAB \\ A & \rightarrow & aA \mid \epsilon \\ B & \rightarrow & bB \mid c \end{array}$$

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

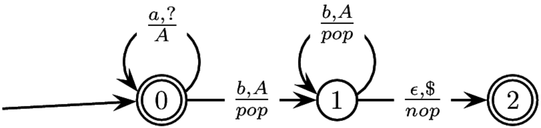
An $LL(k)$ grammar whose language has no $LL(k-1)$ grammar

$$
\begin{array}{rcl}
S & \rightarrow & aSA \mid \epsilon \\
A & \rightarrow & a^{k-1}bS \mid c \\
S & \stackrel{*}{\Rightarrow} & aaaAAA \\
 & \stackrel{*}{\Rightarrow} & aaa\overbrace{\underbrace{aa\ldots a}_{k-1}bS}^{A}\overbrace{\underbrace{aa\ldots a}_{k-1}bS}^{A}\overbrace{c}^{A}
\end{array}
$$

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

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

$$\underbrace{aaaaa\ldots}_{n} \qquad \underbrace{aaaaa\ldots}_{n}\underbrace{bbbbb\ldots}_{n}$$



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

$$
\begin{array}{lcl}
S & \rightarrow & A \mid B \\
A & \rightarrow & aA \mid \epsilon \\
B & \rightarrow & bB \mid c
\end{array}
$$

	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

$$
\begin{aligned}
S &\rightarrow abcX \mid abcY \\
X &\rightarrow xX \mid x \\
Y &\rightarrow yY \mid y
\end{aligned}
$$

$$
\begin{aligned}
S &\rightarrow aB \\
B &\rightarrow bC \\
C &\rightarrow cX \mid cY \\
X &\rightarrow xX \mid x \\
Y &\rightarrow yY \mid y
\end{aligned}
$$

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.

$$\begin{aligned} A &\Rightarrow Aa \\ &\Rightarrow Aaa \\ &\Rightarrow Aaaa \\ &\Rightarrow \ldots \\ &\Rightarrow Aaaaaaaaaaaaaaaaaaaaaa \\ &\Rightarrow baaaaaaaaaaaaaaaaaaaa \end{aligned}$$

Immediate left recursion can be removed

$$A \quad \rightarrow \quad Aw \mid y$$

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

$$\begin{array}{rcl} A & \rightarrow & yB \\ B & \rightarrow & wB \mid \epsilon \end{array}$$

More general immediate left recursion

$$A \quad \rightarrow \quad Aw_1 \mid Aw_2 \mid \ \ldots \ \mid Aw_n \mid x_1 \mid x_2 \mid \ \ldots \ \mid x_m$$

More general immediate left recursion

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

$$\begin{aligned} A &\rightarrow x_1 B \mid x_2 B \mid \ldots \mid x_m B \\ B &\rightarrow w_1 B \mid w_2 B \mid \ldots \mid w_n B \mid \epsilon \end{aligned}$$

Removing left recursion example

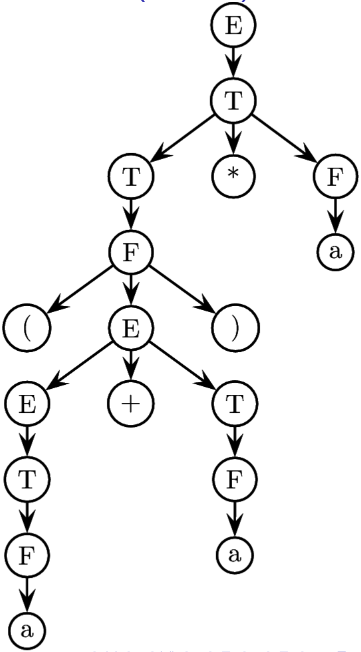
$$A \rightarrow Aa \mid b$$

$$\begin{aligned} A &\rightarrow bB \\ B &\rightarrow aB \mid \epsilon \end{aligned}$$

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

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

- 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?



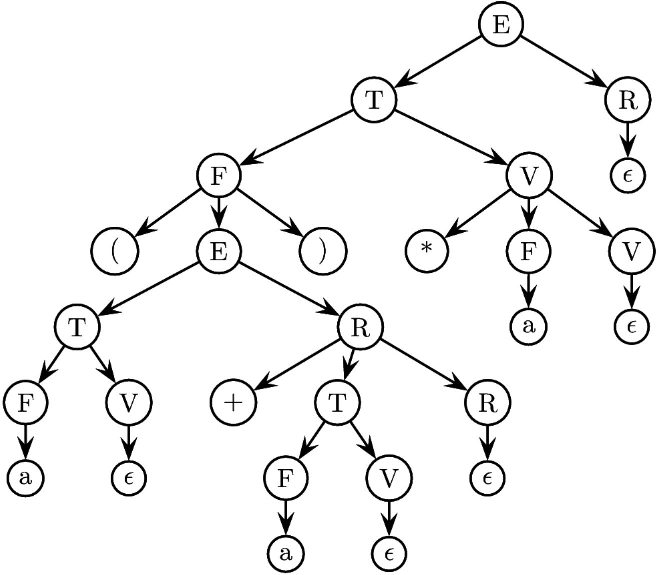
Remove the left recursion

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

$$
\begin{aligned}
E &\rightarrow TR \\
R &\rightarrow +TR \mid \epsilon \\
T &\rightarrow FV \\
V &\rightarrow *FV \mid \epsilon \\
F &\rightarrow (E) \mid a
\end{aligned}
$$

What happened to the tree?

$$
\begin{array}{lcl}
E & \rightarrow & TR \\
R & \rightarrow & +TR \mid \epsilon \\
T & \rightarrow & FV \\
V & \rightarrow & *FV \mid \epsilon \\
F & \rightarrow & (E) \mid a
\end{array}
$$



Indirect left recursion

$$
\begin{array}{rcl}
S & \rightarrow & Bb \\
B & \rightarrow & Sa \mid a
\end{array}
$$

$$S \Rightarrow Bb \Rightarrow Sab$$

Removing indirect left recursion

$$\begin{array}{rcl} S & \rightarrow & Bb \\ B & \rightarrow & Sa \mid a \end{array}$$

Replace B's RHS in S's RHS:

$$S \rightarrow Sab \mid ab$$

Now remove the immediate left recursion

$$\begin{array}{rcl} S & \rightarrow & abT \\ T & \rightarrow & abT \mid \epsilon \end{array}$$

Removing indirect left recursion

$$
\begin{array}{lcl}
A & \rightarrow & Bb \mid e \\
B & \rightarrow & Cc \mid f \\
C & \rightarrow & Ad \mid g
\end{array}
$$

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

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

$$
\begin{array}{lcl}
A & \rightarrow & gcbD \mid fbD \mid eD \\
D & \rightarrow & dcbD \mid \epsilon
\end{array}
$$

$LL(k)$ problems

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

- $\{a, ba, bba\}$
- $a^* b$
- $a^{n+1} b c^n$
- $a^m b^n c^{m+n}$