Notes on Context Free Grammars

Geoffrey Matthews

Department of Computer Science
Western Washington University

February 10, 2017

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
- http://www.cs.sunysb.edu/~cse350/slides/cfg3.pdf

Context Free Grammar

A context free grammar is a grammar where all the rules are the following form:

$$S \rightarrow w$$

where S is a single nonterminal and w is a string of terminals and nonterminals.

Context Free Grammar Examples

$$S \rightarrow aS|\epsilon$$

$$\begin{aligned} S &\rightarrow ABC \\ A &\rightarrow a \\ B &\rightarrow b \\ C &\rightarrow c \end{aligned}$$

$$\begin{aligned} S &\rightarrow AB|A \\ A &\rightarrow aA|a \\ B &\rightarrow Bb|b \end{aligned}$$

Context Free Grammar for Arithmetic Expressions

$$
\begin{array}{lcl}
E & \rightarrow & T \\
E & \rightarrow & E + E \\
E & \rightarrow & E * E \\
E & \rightarrow & (E) \\
T & \rightarrow & a \\
T & \rightarrow & b \\
T & \rightarrow & T0 \\
T & \rightarrow & T1
\end{array}
$$

Note that T could have been reprsented by a regular language.

Context Free Grammar for Programming Language

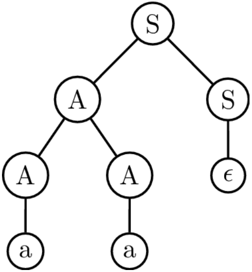
$$
\begin{array}{rcl}
S & \rightarrow & \textbf{while } E \textbf{ do } S \mid \textbf{if } E \textbf{ then } S \textbf{ else } S \mid I \textbf{ := } E \\
S & \rightarrow & \{ \; SL \; \} \\
L & \rightarrow & SL \textbf{ ; } \mid \epsilon \\
E & \rightarrow & \ldots \\
I & \rightarrow & \ldots
\end{array}
$$

- Reference manuals for programming languages usually give the syntax of the language as a CFG.
- Note that keywords, punctuation, *etc.* can be represented by a regular language.

Derivations

- Start with S
- Find a rule for a nonterminal.
- Replace nonterminal with RHS.
- Until no more nonterminals.

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



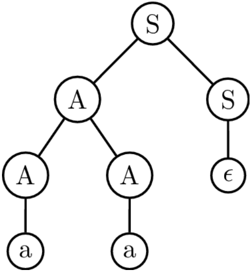
Parse Tree

$$S \Rightarrow AS \Rightarrow AAS \Rightarrow AA \Rightarrow Aa \Rightarrow aa$$

Derivations

- Start with S
- Find a rule for a nonterminal.
- Replace nonterminal with RHS.
- Until no more nonterminals.

$$
\begin{array}{rcl}
S & \rightarrow & AS|\epsilon \\
A & \rightarrow & AA|a
\end{array}
$$



Parse Tree

$$S \Rightarrow AS \Rightarrow AAS \Rightarrow AA \Rightarrow Aa \Rightarrow aa$$

The **language of a grammar** is the set of all sentences for which there exists a derivation.

Derivations

- If there is more than one possible tree for some sentence, the grammar is **ambiguous**.
- There are usually many possible derivations, but only one tree.
- Important derivations are **leftmost** and **rightmost**.

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

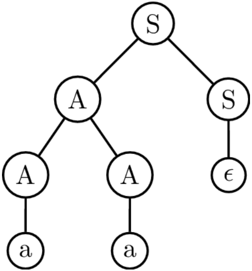
Rightmost:

$\underline{S} \Rightarrow A\underline{S} \Rightarrow AA\underline{S} \Rightarrow A\underline{A} \Rightarrow \underline{A}a \Rightarrow aa$

$\underline{S} \Rightarrow A\underline{S} \Rightarrow \underline{A} \Rightarrow A\underline{A} \Rightarrow \underline{A}a \Rightarrow aa$

Leftmost:

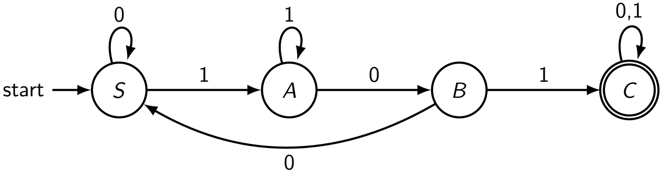
$\underline{S} \Rightarrow \underline{A}S \Rightarrow \underline{A}AS \Rightarrow a\underline{A}S \Rightarrow aa\underline{S} \Rightarrow aa$



Parse Tree

Prove that this grammar is ambiguous.

All regular languages are context free languages



$$
\begin{aligned}
S &\rightarrow 0S \mid 1A \\
A &\rightarrow 0B \mid 1A \\
B &\rightarrow 0S \mid 1C \\
C &\rightarrow 0C \mid 1C \mid \epsilon
\end{aligned}
$$

Generate All Possible Sentences from a CF grammar

Length 1 derivations:

$$
\begin{aligned}
S &\rightarrow abS \mid bAc \mid d \\
A &\rightarrow aA \mid \epsilon
\end{aligned}
$$

$$
\begin{aligned}
S &\Rightarrow abS \\
S &\Rightarrow bAc \\
S &\Rightarrow d
\end{aligned}
$$

1. d

Generate All Possible Sentences from a CF grammar

Length 2 derivations:

$$\begin{array}{rcl} S & \to & abS \mid bAc \mid d \\ A & \to & aA \mid \epsilon \end{array}$$

$$\begin{array}{rcl} S & \Rightarrow & abS \Rightarrow ababS \\ S & \Rightarrow & abS \Rightarrow abbAc \\ S & \Rightarrow & abS \Rightarrow abd \\ S & \Rightarrow & bAc \Rightarrow baAc \\ S & \Rightarrow & bAc \Rightarrow bc \end{array}$$

1. d
2. abd
3. bc

Generate All Possible Sentences from a CF grammar

$$S \rightarrow abS \mid bAc \mid d$$
$$A \rightarrow aA \mid \epsilon$$

Length 3 derivations:

$$
\begin{aligned}
S &\Rightarrow abS \Rightarrow ababS \Rightarrow abababS \\
S &\Rightarrow abS \Rightarrow ababS \Rightarrow ababbAc \\
S &\Rightarrow abS \Rightarrow ababS \Rightarrow ababd \\
S &\Rightarrow abS \Rightarrow abbAc \Rightarrow abbaAc \\
S &\Rightarrow abS \Rightarrow abbAc \Rightarrow abbc \\
S &\Rightarrow bAc \Rightarrow baAc \Rightarrow baaAc \\
S &\Rightarrow bAc \Rightarrow baAc \Rightarrow bac
\end{aligned}
$$

1. d
2. abd
3. bc
4. $ababd$
5. $abbc$
6. bac
7. ...

CFG problems

- Any regular language.
- $a^n b^n$
- $a^n b^{2n}$
- $a^n b^{3n}$
- $a^{4n+5} b^{3n+2}$
- $a^n b^m a^n$
- Even length palindromes
- Odd length palindromes
- All palindromes
- All strings with the same number of a's and b's
- $\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$

Removing ϵ from grammars

$$
\begin{array}{rcl}
S & \rightarrow & aDaE \\
D & \rightarrow & bD \mid E \\
E & \rightarrow & cE \mid \epsilon
\end{array}
$$

- Find all nonterminals N such that $N \Rightarrow \epsilon$.
- Make new rules from old by removing one or more of the null nonterminals.
- Remove all null productions $N \rightarrow \epsilon$.
- May have to keep $S \rightarrow \epsilon$, but only if $\epsilon \in L(S)$.

Removing ϵ from grammars

$$
\begin{aligned}
S &\rightarrow aDaE \\
D &\rightarrow bD \mid E \\
E &\rightarrow cE \mid \epsilon
\end{aligned}
$$

- Null nonterminals: D and E

Original Production	New Productions
$S \rightarrow aDaE$	$S \rightarrow aaE \mid aDa \mid aa$
$D \rightarrow bD$	$D \rightarrow b$
$D \rightarrow E$	$D \rightarrow \epsilon$
$E \rightarrow cE$	$E \rightarrow c$
$E \rightarrow \epsilon$	none

Final grammar:

$$
\begin{aligned}
S &\rightarrow aDaE \mid aaE \mid aDa \mid aa \\
D &\rightarrow bD \mid b \mid E \\
E &\rightarrow cE \mid c
\end{aligned}
$$

Chomsky Normal Form

- All rules must be in one of these forms:

$$\begin{array}{rcl} A & \rightarrow & BC \\ A & \rightarrow & a \\ S & \rightarrow & \epsilon \end{array}$$

- A, B and C are nonterminals, a is a single terminal, and S is the start symbol.
- The last rule is necessary only if the language contains ϵ.
- If a grammar is in CNF, what do we know about the tree?
- If a grammar is in CNF, how long is a derivation?

Converting to Chomsky Normal Form

1. Add S_0, a new start symbol, and the rule $S_0 \to S$.
2. Eliminate ϵ rules.
3. Eliminate **unit** rules, $A \to B$:
   - Find all rules $B \to W$, where W is a string longer than one.
   - Add $A \to W$ for all of them.
4. Fix longer rules:
   - Replace $A \to UVWXYZ$ with
   - $A \to UA_1$, $A_1 \to VA_2$, $A_2 \to WA_3$, $A_3 \to XA_4$, $A_4 \to YZ$.
5. For each terminal x, add a rule $X \to x$ and replace all terminals in long (length two) strings with the corresponding nonterminals.

Example converting to Chomsky Normal Form

$$\begin{aligned} S &\rightarrow aSb \mid T \\ T &\rightarrow cT \mid \epsilon \end{aligned}$$

Step 1:
$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow aSb \mid T \\ T &\rightarrow cT \mid \epsilon \end{aligned}$$

Step 2:
$$\begin{aligned} S_0 &\rightarrow S \mid \epsilon \\ S &\rightarrow aSb \mid ab \mid T \\ T &\rightarrow cT \mid c \end{aligned}$$

Step 3:
$$\begin{aligned} S_0 &\rightarrow aSb \mid ab \mid cT \mid c \mid \epsilon \\ S &\rightarrow aSb \mid ab \mid cT \mid c \\ T &\rightarrow cT \mid c \end{aligned}$$

Step 4:
$$\begin{aligned} S_0 &\rightarrow aD \mid ab \mid cT \mid c \mid \epsilon \\ S &\rightarrow aD \mid ab \mid cT \mid c \\ D &\rightarrow Sb \\ T &\rightarrow cT \mid c \end{aligned}$$

Step 5:
$$\begin{aligned} S_0 &\rightarrow AD \mid AB \mid CT \mid c \mid \epsilon \\ S &\rightarrow AD \mid AB \mid CT \mid c \\ D &\rightarrow SB \\ T &\rightarrow CT \mid c \\ A &\rightarrow a \\ B &\rightarrow b \\ C &\rightarrow c \end{aligned}$$

Note on Eliminating Unit Rules

If we have two mutually recursive unit rules, for example in the following grammar:

$$\begin{array}{rcl} A & \rightarrow & B|XY|UVW \\ B & \rightarrow & A|DE|FGH \end{array}$$

It doesn't hurt to eliminate both of them, so long as the remainder of the clauses is added to each:

$$\begin{array}{rcl} A & \rightarrow & XY|UVW|DE|FGH \\ B & \rightarrow & DE|FGH|XY|UVW \end{array}$$

Any derivation that used the two unit rules, such as

$$A \Rightarrow B \Rightarrow A \Rightarrow B \Rightarrow FGH$$

Can be replaced by a short-circuited version

$$A \Rightarrow FGH$$

Consequences of Chomsky Normal Form

- What do all trees look like?
- What do all derivations look like?
- How long are the derivations?

Consequences of Chomsky Normal Form

- What do all trees look like?
- What do all derivations look like?
- How long are the derivations?
- There are finitely many derivations for a given string.
- To find a parse we can exhaustively search all derivations of this length or less.

Greibach Normal Form

- All rules must be in one of these forms:

$$\begin{aligned} A &\rightarrow aB_1B_2\dots B_n \\ S &\rightarrow \epsilon \end{aligned}$$

- $B_1B_2\dots B_n$ is a (possibly empty) string of nonterminals.
- The last rule is needed only if the language contains ϵ.
- There can be no left recursion.
- What do the trees look like?
- How long are the derivations?

Converting to Greibach Normal Form

1. Add S_0, a new start symbol, and the rule $S_0 \rightarrow S$.
2. Eliminate **unit** rules, $A \rightarrow B$.
3. Remove left recursion.
4. Eliminate ϵ rules.
5. Make substitutions as needed.
   - Can be very difficult and explode to many rules.

Consequences of Greibach Normal Form

- All derivations of a string of length n have n steps.
- There are only finitely many derivations of length n or less.
- To find a parse we can exhaustively search all derivations of length n or less.
- There exists an **effective procedure** to find a parse for a CFL.

Pumping Lemma for CF Languages

- If a CFL is infinite, it must have recursion in it somewhere:

$$\begin{array}{rcl} S & \rightarrow & uNy \\ N & \rightarrow & vNx \mid w \end{array}$$

- This means derivations like this are possible:

$$S \Rightarrow uNy \Rightarrow uvNxy \Rightarrow uvvNxxy \Rightarrow \ldots \Rightarrow uv^4Nx^4y \Rightarrow uv^4wx^4y$$

Pumping Lemma for CF Languages

Theorem

If a language L is context-free, then there exists some $p \geq 1$ such that any string s in L with $|s| \geq p$ can be written as

$$s = uvxyz$$

with substrings u,v,x,y,z, such that

1. $|vxy| \leq p$
2. $|vy| \geq 1$
3. uv^nxy^nz *is in* L *for all* $n \geq 0$

Proof that $L = a^n b^n c^n$ is not CF

- Suppose L is CF, then p exists as in the theorem.
- $a^p b^p c^p \in L$ by definition.
- From theorem, $a^p b^p c^p = uvxyz$ and $uv^n xy^n z \in L$ for all n, but we don't know which parts are where.
- Case 1: v or y contains two different letters.
  - Then uv^2xy^2z must have letters out of alphabetical order.
  - Then $uv^2xy^2z \notin L$, contradiction.
- Case 2: v and y each contain only one kind of letter.
  - Then uv^2xy^2z contains more of 1 or 2 kinds of letter, not all three.
  - Then $uv^2xy^2z \notin L$, contradiction.

Pumping Lemma exercises (some are hard)

Show that each of the following languages is not CF.

- a^n where n is prime
- a^m where $m = n^2$
- $a^\ell b^m c^n$ where $\ell < m < n$
- $a^n b^n c^i$ where $i \leq n$
- ww where $w \in (a+b)^*$
- $a^n b^n a^n$

CF Languages are closed under union, product, and closure

- Let S_1 and S_2 be the start symbols for L_1 and L_2.
- A grammar for $L_1 \cup L_2$ can be constructed starting with

$$S \to S_1 \mid S_2$$

- A grammar for $L_1 L_2$ can be constructed starting with

$$S \to S_1 S_2$$

- A grammar for L_1^* can be constructed starting with

$$S \to S_1 S \mid \epsilon$$

CF languages are NOT closed under complement

- $L = a^\ell b^m c^n$ where either $\ell \neq m$ or $m \neq n$ is CF
  - (exercise)
- The complement of L is $a^n b^n c^n$, which is not CF
  - (see above)

CF languages are NOT closed under intersection

- $L_1 = a^m b^m c^n$ is CF
  - (exercise)
- $L_2 = a^m b^n c^n$ is CF
  - (exercise)
- $L_1 \cap L_2 = a^n b^n c^n$, which is not CF
  - (see above)

The intersection of a CF language and a regular language is CF

- Given a PDA for one and a DFSA for the other:
- Create a new PDA with states that are the cross product of the states of the two machines.
- As input is processed, run both machines in parallel.
- Accept if both accept.