Chapter 15: CFG = PDA *

Peter Cappello
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106
cappello@cs.ucsb.edu

• The corresponding textbook chapter should be read before attending this lecture.

• These notes are not intended to be complete. They are supplemented with figures, and other material that arises during the lecture period in response to questions.

*Based on Theory of Computing, 2nd Ed., D. Cohen, John Wiley & Sons, Inc.
For every CFG there is a PDA

- We illustrate the general case with an example. Consider the following CFG in Chomsky Normal Form:

\[
\begin{align*}
S & \rightarrow SB \mid AB \\
A & \rightarrow CC \\
B & \rightarrow b \\
C & \rightarrow a
\end{align*}
\]

- Draw the corresponding PDA on the board.

- \( \Sigma = \{a, b\} \) and \( \Gamma = \{S, A, B, C\} \).

- Run the PDA on \( S \Rightarrow SB \Rightarrow ABB \Rightarrow CCBB \Rightarrow aCBB \Rightarrow aaBB \Rightarrow aabB \Rightarrow aabb \).
• The PDA emulates leftmost derivations of the CFG.

• At every step in a leftmost derivation, the working string is of the form $terminal^*nonterminal^*$: some string of terminals followed by some string of nonterminals.

• At every step in the PDA’s emulation of the derivation, the working string’s terminals are the sequence of symbols already read from the input tape; the sequence of nonterminals is the contents of the PDA’s stack.

• When a word is accepted by the PDA, every symbol has been read from the input tape, and the stack is empty.

This corresponds to a derived working string of only terminals: a word in the language generated by the CFG.
The Algorithm

1. For any CNF grammar, $G$, construct the following PDA fragment consisting of a START, a PUSH $S$, and a POP: Draw the fragment.

2. For each production of the form $X \rightarrow YZ$, for nonterminals $X, Y$, and $Z$, add the following PUSH-loop fragment: Draw the fragment.

3. For each production of the form $X \rightarrow t$, for nonterminals $X$ and terminal $t$, add the following READ-loop fragment: Draw the fragment.

4. If $\Lambda \in L(G)$, then augment the CNF grammar with $S \rightarrow \Lambda$ and add the following self-loop. Draw the fragment.

5. Add a fragment that accepts when the stack is empty and the input has been completely read. Draw the fragment.
For every PDA there is a CFG

• The 1st step in this proof is to put PDAs in a standard format, called conversion form.

• We introduce a new “marker state” called a HERE state.

The HERE state:

– has graphical shape of a diamond
– can be placed on any edge
– can be subscripted (so that references to it are unique).
– can have multiple, unlabelled out-edges (which consequently imply the use of nondeterminism).
• **Definition:** A PDA is in **conversion form** when:

1. There is only 1 ACCEPT state.
2. There are no REJECT states.
3. Every READ or HERE is followed immediately by a POP.
4. The path between 2 POP states contains a READ or a HERE.
5. Branching occurs only at READ or HERE states. Edges have 1 label.
6. The stack initially has a “$”. If popped, it is pushed immediately, except when accepting. The stack is never popped when empty. Immediately before accepting, $ is popped.
7. The PDA begins with the sequence START → POP $→ PUSH $→ \{ \text{READ} \mid \text{HERE} \}.$
8. The input word is read entirely before accepting.
• We show how any PDA can be converted into this form, 1 constraint at a time.

1. **There is only 1 ACCEPT state.**
   Replace all ACCEPT states with 1. Direct all edges to the former ACCEPT states to this 1 ACCEPT state.

2. **There are no REJECT states.**
   Simply remove REJECT states; rejection is implicit.

3. **Every READ or HERE is followed immediately by a POP.**
   If READ or HERE is not immediately followed by a POP, insert a POP followed by a PUSH of the symbol POPed.
   **Draw this.**
4. **The path between 2 POP states contains a READ or a HERE.**
   If the path between 2 POPs contains neither a READ nor a HERE, insert a HERE state immediately after the 1st POP.
   Draw this.

5. **Branching occurs only at READ or HERE states. Edges have 1 label.**
   We transform all branching at POP statements into branching at READ or HERE states.
   - Draw the construction for READ before branching POP.
   - Illustrate this construction on the POP-PUSH construction. There must be a READ or HERE preceding the branching POP, not another POP.
6. The stack initially has a “$”. If popped, it is pushed immediately, except when accepting. The stack is never popped when empty. Immediately before accepting, $ is popped.

   – Replace POP $\rightarrow$ with POP $\rightarrow$ PUSH $. (Remove extra POP branches, as needed.)
   – Replace the ACCEPT state with the construction that empties the stack before accepting.

7. The PDA begins with the sequence START $\rightarrow$ POP $\rightarrow$ PUSH $\rightarrow$ { READ | HERE }.
   This is straightforward.

8. The input word is read entirely before accepting.
   Use the “while ( input.getChar() <> ’ ’ );” construction, ensuring all input is READ before accepting.
• Illustrate the conversion with a PDA for $\{a^{2n}b^n\}$.
  Draw the before and after PDAs.
• A PDA in conversion form can be seen as a graph of path segments:
  
  **From:** A START, READ, or HERE
  
  **To:** A READ, HERE, or ACCEPT
  
  **Reading:** 0 or 1 input symbol
  
  **Popping:** Exactly 1 stack symbol
  
  **Pushing:** Some string (including $\Lambda$) onto the stack.

• The START, READ, HERE, and ACCEPT states are the PDA’s joints.

• Highlight the 7 path segments of the example converted PDA.

• Here is the path segment table:
<table>
<thead>
<tr>
<th>Path segment</th>
<th>From</th>
<th>To</th>
<th>Read</th>
<th>Pop</th>
<th>Push</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>START</td>
<td>READ₁</td>
<td>Λ</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>2</td>
<td>READ₁</td>
<td>READ₁</td>
<td>a</td>
<td>$</td>
<td>a$</td>
</tr>
<tr>
<td>3</td>
<td>READ₁</td>
<td>READ₁</td>
<td>a</td>
<td>a</td>
<td>aa</td>
</tr>
<tr>
<td>4</td>
<td>READ₁</td>
<td>HERE</td>
<td>b</td>
<td>a</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>HERE</td>
<td>READ₂</td>
<td>Λ</td>
<td>a</td>
<td>—</td>
</tr>
<tr>
<td>6</td>
<td>READ₂</td>
<td>HERE</td>
<td>b</td>
<td>a</td>
<td>—</td>
</tr>
<tr>
<td>7</td>
<td>READ₂</td>
<td>ACCEPT</td>
<td>Δ</td>
<td>$</td>
<td>—</td>
</tr>
</tbody>
</table>

- For every word accepted by the PDA, there is a path from START to ACCEPT. These paths can be decomposed in path segments.

- For $aaaabb$, the accepting path can be described by the path segment sequence: 1, 2, 3, 3, 3, 4, 5, 6, 5, 7.

- An accepted path in an FA corresponds to strings of letters; in a converted PDA, it corresponds to strings of path segments.
• The set of path segment words (e.g., 1, 2, 3, 3, 3, 4, 5, 6, 5, 7) that correspond to accepted inputs is the 1st step to constructing a CFG for the language accepted by the original PDA.

• The plan for completing this proof is as follows:

  1. Give a CFG that generates the path segment words that correspond to accepting paths.

  2. Transform this CFG into one that generates the words accepted by the original PDA (i.e., in the original set of terminals).

• The constraints that the CFG must embody include:

  – The path segment word starts with the path segment that begins with START.

  – Path segment i’s endpoint is path segment i + 1’s begin point. Such a path segment sequence is called joint-consistent.
– When a path segment pops a character, it should, in fact, be on the top of the stack.
Such a path segment sequence is called **stack-consistent**.
Illustrate with $a^{2n}b^n$ example.

- The set of **terminals** of the accepting path language is $\{s_1, s_2, \ldots, s_n\}$, where there are $n$ path segments.

- We define a set of **nonterminals** of the form $Net(X, Y, Z)$, where
  - $X, Y \in \{START, READ_i, HERE_j, ACCEPT\}$
  - $Z \in \Gamma$. 


• $Net(X, Y, Z)$ means:

— There is a path from $X$ to $Y$ (involving 1 or more path segments)
— The net affect on the stack of traversing this path is that $Z$ is popped from the stack.

  * Other things may have been pushed on the stack during the traversal, but eventually the stack was popped down to and including $Z$;
  * Nothing under $Z$ was ever popped.

Illustrate $Net(X, Y, Z)$.
There are 3 rules for creating productions in our CFG:

1. The initial production is:
   \[ S \rightarrow Net(START, ACCEPT, \$). \]
2. For path segment, \( s_i \), from \( X \) to \( Y \) that pops \( Z \) and has no Push entry, include a production of the form:
   \[ Net(X, Y, Z) \rightarrow s_i \]
   Illustrate with \( \{a^{2n}b^n\} \) example: \( Net(READ_1, HERE, a) \rightarrow s_4 \)
3. For each path segment, \( i \), from \( X \) to \( Y \) that pops \( Z \) and pushes \( m_1, \ldots, m_n \), include productions of the form
   \[ Net(X, S_n, Z) \rightarrow s_i Net(Y, S_1, m_1) \cdots Net(S_{n-1}, S_n, m_n) \]
   where \( S_1, \ldots, S_n \) are states in the PDA.
   Illustrate with \( \{a^{2n}b^n\} \) example.
   \[ Net(READ_1, READ_2, a) \rightarrow s_3 Net(READ_1, HERE, a) Net(HERE, \ldots) \]

It may be that rule 3 produces productions that are useless: They
cannot derive a string of terminals.

- We illustrate the CFG construction with our running example. We abbreviate \texttt{READ}_i by \texttt{R}_i, and \texttt{HERE} by \texttt{H}.
The productions derived from the path segments of the conversion PDA for \( \{a^{2n}b^n\} \) are:

  - For rule 1, we add production

    \[ 1: S \rightarrow \text{Net}(\texttt{START}, \texttt{ACCEPT}, \$) \]
- For rule 2, we add 4 productions for the 4 path segments that push no symbols:

2: \( Net(R_1, H, a) \rightarrow s_4 \)
3: \( Net(H, R_2, a) \rightarrow s_5 \)
4: \( Net(R_2, H, a) \rightarrow s_6 \)
5: \( Net(R_2, ACCEPT, \$) \rightarrow s_7 \)
– Rule 3 applies to 3 path segments: \( s_1, s_2, \) and \( s_3. \)

For \( s_1, \) it results in productions of the form

\[
\text{Net}(\text{START}, X, \$) \rightarrow s_1 \text{Net}(R_1, X, \$),
\]

where \( X \) can be \( \text{READ}_1, \text{READ}_2, \text{HERE}, \) and \( \text{ACCEPT}, \)
yielding 4 productions:

6: \( \text{Net}(\text{START}, R_1, \$) \rightarrow s_1 \text{Net}(R_1, R_1, \$) \)

7: \( \text{Net}(\text{START}, R_2, \$) \rightarrow s_1 \text{Net}(R_1, R_2, \$) \)

8: \( \text{Net}(\text{START}, H, \$) \rightarrow s_1 \text{Net}(R_1, H, \$) \)

9: \( \text{Net}(\text{START}, \text{ACCEPT}, \$) \rightarrow s_1 \text{Net}(R_1, \text{ACCEPT}, \$) \)
For $s_2$, it results in productions of the form 
$Net(R_1, X, \$) \rightarrow s_2 Net(R_1, Y, a) Net(Y, X, \$)$,
where $X$ can be $READ_1$, $READ_2$, $HERE$, and $ACCEPT$,
and $Y$ can be $READ_1$, $READ_2$, and $HERE$, yielding 12 productions:

10: $Net(R_1, R_1, \$) \rightarrow s_2 Net(R_1, R_1, a) Net(R_1, R_1, \$)$
11: $Net(R_1, R_1, \$) \rightarrow s_2 Net(R_1, R_2, a) Net(R_2, R_1, \$)$
12: $Net(R_1, R_1, \$) \rightarrow s_2 Net(R_1, H, a) Net(H, R_1, \$)$
13: $Net(R_1, R_2, \$) \rightarrow s_2 Net(R_1, R_1, a) Net(R_1, R_2, \$)$
14: $Net(R_1, R_2, \$) \rightarrow s_2 Net(R_1, R_2, a) Net(R_2, R_2, \$)$
15: $Net(R_1, R_2, \$) \rightarrow s_2 Net(R_1, H, a) Net(H, R_2, \$)$
16: $Net(R_1, H, \$) \rightarrow s_2 Net(R_1, R_1, a) Net(R_1, H, \$)$
17: $Net(R_1, H, \$) \rightarrow s_2 Net(R_1, R_2, a) Net(R_2, H, \$)$
18: $Net(R_1, H, \$) \rightarrow s_2 Net(R_1, H, a) Net(H, H, \$)$
19: $Net(R_1, \textit{ACCEPT}, \$) \rightarrow s_2 Net(R_1, R_1, a) Net(R_1, \textit{ACCEPT}, \$)$
20: $Net(R_1, \textit{ACCEPT}, \$) \rightarrow s_2 Net(R_1, R_2, a) Net(R_2, \textit{ACCEPT}, \$)$
21: $Net(R_1, \textit{ACCEPT}, \$) \rightarrow s_2 Net(R_1, H, a) Net(H, \textit{ACCEPT}, \$)$
For $s_3$, it results in productions of the form
\[ \text{Net}(R_1, X, a) \rightarrow s_3 \text{Net}(R_1, Y, a)\text{Net}(Y, X, a), \]
where $X$ can be $\text{READ}_1$, $\text{READ}_2$, $\text{HERE}$, and $\text{ACCEPT}$, and $Y$ can be $\text{READ}_1$, $\text{READ}_2$, and $\text{HERE}$, yielding 12 productions:

10: $\text{Net}(R_1, R_1, a) \rightarrow s_2 \text{Net}(R_1, R_1, a)\text{Net}(R_1, R_1, a)$
11: $\text{Net}(R_1, R_1, a) \rightarrow s_2 \text{Net}(R_1, R_2, a)\text{Net}(R_2, R_1, a)$
12: $\text{Net}(R_1, R_1, a) \rightarrow s_2 \text{Net}(R_1, H, a)\text{Net}(H, R_1, a)$
13: $\text{Net}(R_1, R_2, a) \rightarrow s_2 \text{Net}(R_1, R_1, a)\text{Net}(R_1, R_2, a)$
14: $\text{Net}(R_1, R_2, a) \rightarrow s_2 \text{Net}(R_1, R_2, a)\text{Net}(R_2, R_2, a)$
15: $\text{Net}(R_1, R_2, a) \rightarrow s_2 \text{Net}(R_1, H, a)\text{Net}(H, R_2, a)$
16: $\text{Net}(R_1, H, a) \rightarrow s_2 \text{Net}(R_1, R_1, a)\text{Net}(R_1, H, a)$
17: $\text{Net}(R_1, H, a) \rightarrow s_2 \text{Net}(R_1, R_2, a)\text{Net}(R_2, H, a)$
18: $\text{Net}(R_1, H, a) \rightarrow s_2 \text{Net}(R_1, H, a)\text{Net}(H, H, a)$

21
19: \(Net(R_1, ACCEPT, a) \rightarrow s_2 Net(R_1, R_1, a) Net(R_1, ACCEPT, a)\)
20: \(Net(R_1, ACCEPT, a) \rightarrow s_2 Net(R_1, R_2, a) Net(R_2, ACCEPT, a)\)
21: \(Net(R_1, ACCEPT, a) \rightarrow s_2 Net(R_1, H, a) Net(H, ACCEPT, a)\)

- Claim: a word is generated by this CFG \(\Rightarrow\) there is a corresponding accepting path in the PDA.

The grammar generates a sequence of path segments that:

1. starts in a START state
2. goes through a sequence of path segments that is:
   - joint-consistent
   - stack-consistent.
3. ends in an ACCEPT state.

Thus, it corresponds to an accepting path in the original PDA.
• Claim: An accepting path in the PDA ⇒ there is a corresponding word generated by this CFG.

1. Every accepting path is associated with a sequence of stack changes.
2. Every stack change is a Net nonterminal (i.e., whether the net effect was to pop 1 symbol, or push some symbols).
3. Every stack change is equivalent to either:
   – a path segment
   – a sequence of smaller stack changes.
4. Each sequence of smaller stack changes is represented by a production.
5. The nonterminals representing these smaller stack changes, are further decomposed with productions, ultimately leading to a sequence of path segments: a sequence of terminals in the CFG.

• In sum,
1. Given a PDA, transform it to an equivalent PDA in conversion form.
2. Identify the path segments.
3. Construct the summary table of path segments.
4. Construct the CFG, using the 3 rules, from the summary table.
5. Finally, given the CFG above, we construct a CFG for the language accepted by the PDA.

• The grammar is a simple extension of the grammar above:
  For each path segment $s_i$ that reads symbol $\alpha$, add the production:
  $$s_i \rightarrow \alpha$$
  $\alpha$ includes $\Lambda$ (not reading any input) and $\Delta$ (reading a blank).
• We finally add the production $\Delta \rightarrow \Lambda$ to eliminate the blank from the word accepted.
Thus for any PDA $A$ there is a CFG $G$ such that $L(A) = L(G)$.