Today we look at the equivalence of CFG and PDA. Some of the details of the proof are a little tricky.

Homework #4 on Chapters 13, 14 and 15 is assigned, see end of notes. Answers are due by 10/22/13.

Program #1 is due today.

Chapter 15

Errata in Chapter 15:

p. 319, In the chart the label on the edge from POP to PUSH C should be "A" not "S".
p. 328 line 4 should read
   "All branching, in or out, deterministic or nondeterministic, ...".
p. 330 The discussion of Condition 5 should also include the rectification of the case when multiple branches are entering a POP. This can be handled either by bifurcating the POP into multiple POP’s with one incoming branch each or by replacing it by a HERE (which allows branching in and out followed by a POP.
p. 331, Top chart: Delete the arrowhead coming into PUSH a from the left.
p. 331, Bottom chart: The POP state leading into the accept state should be labeled POP3. And, to be consistent with the next page as well as the preceding discussion, this PDA should have no b-edge from READ2 to POP1. Instead, it should have a b-edge to the new state POP7 just to its left, and then POP7 should have an a-edge straight up to the HERE state.
p. 335, Step 4: insert “joint-consistent and" in front of "STACK-consistent".
p. 345, Note that the POP in the PDA shown gets replaced by a HERE on the next page followed by POP’s.

Chapter 15 shows that the set of all languages accepted by PDAs is exactly the same as the set of languages generated by CFGs. The proof is in two parts, Theorems 30 and 31, and it is the constructive nature of the proofs of these theorems that is a little involved, particularly Theorem 31. The proof of Theorem 31 is probably the longest of any in the book.

THEOREM 30

Given a CFG that generates language L, there is a PDA that accepts exactly L.
THEOREM 31

Given a PDA that accepts the language L, there exists a CFG that generates exactly L.

Proof of Theorem 30

By construction.
Recall that any CFG can be written in Chomsky Normal Form (CNF) where the right hand side of a rule is either a single terminal or a pair of nonterminals.
Recall also that the leftmost derivation of a word is one where the leftmost nonterminal is replaced at each stage.

We take a CFG in CNF and construct a PDA where

The stack alphabet is the set of nonterminals.
The tape alphabet is the set of terminals.

The PDA starts by pushing S on the stack
The central operation of the PDA is a POP action from which we branch on the symbol popped. There is a branch out of this action for each rule in the CNF form of the grammar. We branch on the symbol on the left-hand side of the rule to one of two kinds of actions:

1. If the right-hand side of the rule is two nonterminals, we PUSH the two RHS symbols onto the stack. The rightmost of the two symbols is pushed first.
2. If the right-hand side of the rule is a terminal, we READ from the input. If the symbol we read is not the one on the right-hand side of the rule, we “crash”.

After one or other of these two actions, return to the POP action.

If, when the stack is empty, there is no more input then we ACCEPT.

See example grammar at the end of page 318, the corresponding PDA on page 319 and the trace of the machine acting on input string $aab$ on pages 320 and 321.

If the string is in the language then there is a leftmost derivation and it is this derivation that the action of the PDA represents. The PDA is non-deterministic so we are saying that if the input is in the language there is some sequence of actions that will cause it to transition to an ACCEPT state.

Another example of grammar-into-machine is on Page 322.

The algorithm for constructing the PDA from the grammar rules is on pages 323-325
1. We need an initial action that pushes the start symbol on the stack then goes to the central POP.
2. For each production rule having two non-terminals on the right-hand side, remember the grammar is in CNF, there is a circuit from the POP back to the POP that pushes the rightmost then the leftmost non-terminal on the stack.
3. For each production rule having a terminal on the right-hand side there is a circuit from the POP back to the POP that includes a READ. If the character READ is the appropriate terminal then we return to the POP otherwise the machine crashes.
4. Finally, there is a transition from the POP when blank (\(\Delta\)) is popped. We read from the tape and if a blank is read we ACCEPT otherwise the machine crashes.

There is another example of grammar-into-machine followed by a trace on pages 325-326.

Proof of Theorem 31

Now we have to show that for every PDA there is a corresponding CFG.

This is a bit complicated. We make it a little simpler by first transforming the PDA into “conversion form” PDA. A conversion form PDA may include “marker” states (called “HERE”) that have no affect on stack or input. Also a conversion form PDA has the following eight properties.

1. Only one ACCEPT state.
2. No REJECT states
3. Every READ/HERE goes directly to a POP
4. No two POPs in a row
5. All branching at READ/HERE, not at POP
6. Prior to START, push $ on stack. Stack not popped beyond this point. If $ is ever popped, it is pushed back except if transition to ACCEPT when it can be left off.
7. The PDA must begin with POP of the $ followed by a push of the $ then transition to initial README or HERE
8. Entire input string must be consumed before machine can ACCEPT.

We can see that any PDA can be put into a “conversion form” PDA having these eight properties:

1. Trivial: if we have multiple ACCEPT states, make all incoming edges go to single ACCEPT.
2. Trivial: eliminate all REJECT states and incoming edges. Machine will now crash instead.
3. We can introduce POP-PUSH pairs of states (that therefore have no net affect on the stack) as needed to satisfy this requirement.
4. We can insert a HERE state between two POPs
5. No problem. Machine is non-deterministic anyway. Just move multiple branches back to previous READ. So instead of

![Diagram](image.png)

We have

![Diagram](image2.png)

6. Just means that initial stack top is $ not $  
7. OK – given 6  
8. Just add an input-consuming loop before ACCEPT.

We are going to this trouble because it is one of 6 steps showing that for every PDA there is an equivalent CFG. The 6 steps, see page 335, are:

1. Start with PDA.  
2. Modify it so it is conversion form; see above.  
3. From the PDA in conversion form, derive a table representing the machine.  
4. Every word accepted by the PDA corresponds to a sequence of rows in the table.  
   We define a “row language” to be the set of such sequences.  
5. We define a CFG that generates the sequences in the row language.  
6. We modify the CFG in a simple way so that it generates the words accepted by the original PDA. Thus from the PDA we have derived a CFG.
As Cohen points out, many books define a PDA as a table to begin with thus removing
the need for steps 1, 2 and 3 but presenting the PDA as graph gives us a consistent format
across FA, TG and PDA. So now we are at step 3.

**Step 3: Derive a table from the conversion form PDA**

Because of the way in which we have limited branching, the PDA in conversion form is a
collection of “path segments” which can be listed in a table with 5 column headings

<table>
<thead>
<tr>
<th>FROM</th>
<th>TO</th>
<th>READ</th>
<th>POP</th>
<th>PUSH</th>
</tr>
</thead>
</table>

The FROM node is START, READ or HERE
The TO node is READ, HERE or ACCEPT
The READ entry indicates either no reading or the one character read.
The POP entry shows the one character popped
The PUSH entry lists the characters (if any) pushed on the stack

Cohen has an example machine on Page 332 and the corresponding table on Page 333.
The order of the rows in a table is not significant.

**Step 4: The row language**

Every word accepted by the PDA corresponds to a path through the PDA which, because
of step 3, can now be represented as a sequence of row entries.

However, every such sequence must be

- **“joint consistent”** The node at the end of one segment must be the same as the
  node at the beginning of the next segment and

- **“stack consistent”** As we traverse the sequence, the stack actions must work –
  we cannot push a $b$ for example then pop an $a$.

The row language is the set of these row sequences. So a word in the row language might
be something like

Row$_{23}$Row$_{19}$Row$_{27}$

Now we have to show that we can devise a CFG for this row language
**Step 5: A CFG for the row language**

Terminals will be symbols representing individual rows such as Row₅ and Row₂₁.

Non terminals in the CFG will look like

\[ \text{Net}(X,Y,Z) \]

This just represents the fact that the net effect of the path from X to Y is to pop Z from the stack.

For example,

\[ \text{Net}(\text{READ1}, \text{HERE}, a) \]

means there is a sequence of rows representing a path from READ1 to HERE that has the net effect of popping ‘a’ off the stack.

The following are the kinds of production rules that we will have in our CFG

\[
\begin{align*}
\text{Net}(X,Y,Z) & \rightarrow \text{Row}_m \\
\text{Net}(X,Y,Z) & \rightarrow \text{Row}_n \text{ Net}(P,Q,R) \ldots \text{Net}(W,Y,Z)
\end{align*}
\]

There are 3 rules for generating productions from a table

Rule 1: There is always a production \[ S \rightarrow \text{Net}(\text{START}, \text{ACCEPT}, \text{$$}) \]

Rule 2: For every row \( i \) with empty PUSH entry there is a production of the form

\[ \text{Net}(\text{FROM}, \text{TO}, \text{POP}) \rightarrow \text{Row}_i \]

Rule 3: For every row \( i \) with a non-empty PUSH entry there is a collection of productions generally of the form

\[ \text{Net} (\text{FROM}, \text{blah}, \text{POP}) \rightarrow \text{Row}_i \text{Net} (\text{TO}, x, x) \ldots \text{Net} (x, x, x) \]

See the example on pages 340 and 341 derived from rows 1, 2 and 3 of the table on page 333.

Every string in this language derived from the start symbol S is a sequence of rows that is joint and stack consistent. Every path through the PDA can be generated from this grammar.
Step 6: Modified CFG generates language of PDA

To make the grammar generate the actual words accepted by the PDA instead of the "row language" we just need to add productions that replace “Rowi” by an appropriate symbol. The appropriate symbol is the character under the READ heading in row \( i \) of the table.

The language generated by this CFG is exactly the one accepted by the PDA, thus Theorem 31 is proved.

Example

See the example on pages 345-348.

We start with a simple CFG for even length strings consisting of only a’s. By Theorem 30 we know that there is a PDA for this language and it is easy to derive from the productions. This part of the example ends at the end of Page 345 with a diagram of the PDA.

The rest of the example is an illustration of Theorem 31 and takes much longer. The conversion form of the machine of page 345 is shown at the top of page 346. The table form of this machine is on pp346-347 and has 9 rows. Cohen does not list all 44 productions derived from the table, just the general form of them; see page 347. Then we add the 9 productions that lead to the symbols actually read by the PDA thus ending up with a 53-rule grammar equivalent to the 2-rule grammar we started with!

In the next chapter we will see that there are some languages that cannot be defined by CFG.

Reading Assignment

Read Chapter 15 but don’t get too bogged down in the details. If you have time, look ahead to Chapter 16.
Homework #4

Here is Homework #4 – due October 22. As usual, each of the five questions is worth a maximum of 20 points. Covers Chapters 13, 14 and 15

1. Each of the following CFGs has a production using the symbol $\Lambda$ and yet $\Lambda$ is not a word in its language. Using the algorithm in Chapter 13, show that there are other CFGs for these languages that do not use $\Lambda$-productions.

   (i)  $S \rightarrow aX \mid bX$
        $X \rightarrow a \mid b \mid \Lambda$

   (ii) $S \rightarrow aX \mid bS \mid a \mid b$
        $X \rightarrow aX \mid a \mid \Lambda$

   (iii) $S \rightarrow aS \mid bX$
        $X \rightarrow aX \mid \Lambda$

   (iv) $S \rightarrow XaX \mid bX$
        $X \rightarrow XaX \mid XbX \mid \Lambda$

2. Convert each of the following CFGs to CNF

   (i)  $S \rightarrow SS \mid a$

   (ii) $S \rightarrow aSa \mid SSa \mid a$

   (iii) $S \rightarrow aXX$
        $X \rightarrow aS \mid bS \mid a$

   (iv) $E \rightarrow E + E$
        $E \rightarrow E * E$
        $E \rightarrow (E)$
        $E \rightarrow 7$

        The terminals are + * ( ) 7
3. Consider the following deterministic PDA

(i) What is the language accepted by this PDA?
(ii) Find a CFG that generates this language.
(iii) Is this language regular?
4. Consider the following nondeterministic PDA

Show that the language recognized by this machine is

\[
\text{TRAILINGCOUNT} = \{sa^{\text{length}(s)}\}
\]

= any string \(s\) over the alphabet \{ab\} followed by as many a’s as \(s\) has letters

5. Using the algorithm of Theorem 30, construct a PDA that accepts the same language as the following grammar.

\[
\begin{align*}
S & \rightarrow XaaX \\
X & \rightarrow aX \mid bX \mid \Lambda
\end{align*}
\]