Previously:
Context-free grammars
Pushdown automata
Equivalence, direction 1: CFG \rightarrow PDA

Today:
Equivalence, direction 2: PDA \rightarrow CFG
Assignment 4 due
Exam 1 graded

Exam 1 statistics

- Min: 70 (rounded)
- Avg: 86 (rounded)
- Max: 97 (rounded)
- Std. dev.: 7.6

A long, long time ago...
Example of a CFG

Production rules: substitutions

Non-terminals: variable that can have a substitutions

Terminals: symbols that are part of the alphabet, no substitutions

Start variable: left side of top-most rule

A → aAb
A → B
B → ε

Formal CFG notation

Productions = rules of the form
head → body

head is a variable
body is a string of zero or more variables and/or terminals

Start Symbol = variable that represents “the language”

Notation: G = (V, Σ, P, S)

V = variables
Σ = terminals
P = productions
S = start symbol

Pushdown Automata

Add a stack to a FA
Typically non-deterministic
An automaton equivalent to CFGs

Finite state control

x b c c a a input

stack

y

z

x, y → z

q

Notation

If at state p with next input symbol x and top of stack is y:

Go to state q and replace y by z on stack

x = ε: Ignore input; don’t read.
y = ε: Ignore top of stack and push z.
z = ε: Pop y.
Equivalence of CFGs and PDAs

For every CFG, we showed we can generate a nondeterministic PDA that recognizes the language the CFG generates.

Now we'll show that we can generate a CFG to generate the language a nondeterministic PDA recognizes.

**PDA to CFG**

Convert PDA $P$ into CFG $G$.

First modify $P$ to be a normalized PDA $N$ so that:

- It has a single accept state, $q_{accept}$
- Create $\epsilon$-transitions from old accept states to this new accept state
- It empties the stack before accepting
  - Push a special character $\$$ on the stack in the start state (introducing a new start state in the process)
  - Introduce a new temporary state $q_{temp}$ that replaces $q_{accept}$, which has transitions popping all characters from the stack (except $\$$)

Introduce transition:

$\text{q}_{\text{temp}} \xrightarrow{\epsilon, \$$} \epsilon \xrightarrow{\epsilon} \text{q}_{\text{accept}}$

Continued…

Each transition either pushes a symbol onto the stack or pops one off the stack, but not both at the same time

Replace a simultaneous pop/push move with a two-transition rule that goes through a new state

E.g.,

$\text{q}_i \xrightarrow{a, b \rightarrow c} \text{q}_j$

(read $a$ from input, pop $b$ from stack, push $c$)

Introduce special state $q'_{temp}$ plus 2 transitions, one doing pop and one doing push:

$\text{q}_i \xrightarrow{a, b \rightarrow \epsilon} \text{q}'_{temp} \xrightarrow{\epsilon, \epsilon \rightarrow c} \text{q}_j$

Replace a transition that neither pops nor pushes with two transitions that push and then immediately pop some newly-created dummy stack symbol

$\text{q}_i \xrightarrow{a, \epsilon \rightarrow \epsilon} \text{q}_j \xrightarrow{a, \epsilon \rightarrow X} \text{q}'_{temp} \xrightarrow{\epsilon, X \rightarrow \epsilon} \text{q}_j$

$\text{q}_i \xrightarrow{a, \epsilon \rightarrow X} \text{q}_j \xrightarrow{a, \epsilon \rightarrow \epsilon} \text{q}'_{temp} \xrightarrow{\epsilon, X \rightarrow \epsilon} \text{q}_j$
Normalizing the PDA: Example

Original PDA:

\[ L(N) = (ba \cup baa)^*b \cup \varepsilon \]

Pure push pop

Make sure the stack is always active by replacing inactive stack moves by a push followed by immediate pop of a dummy symbol.

Pure push pop

Any move that replaces the top letter on the stack should be changed into a pop followed by a push.

Unique accept state

Turn off original accept states and connect to a new accept state.

Remember: each move must either push or pop from the stack.
Empty stack

Make sure the stack empties its content by adding a new dummy empty stack symbol and new start/accept states

\[ \epsilon, \epsilon \rightarrow \$
\]

\[ b, \epsilon \rightarrow X
\]

\[ a, \epsilon \rightarrow \epsilon
\]

\[ \epsilon \rightarrow D
\]

\[ \epsilon \rightarrow \epsilon
\]

\[ \epsilon \rightarrow Y
\]

\[ \epsilon \rightarrow D
\]

\[ \epsilon \rightarrow \epsilon
\]

\[ X \rightarrow \epsilon
\]

\[ Y \rightarrow \epsilon
\]

\[ \epsilon \rightarrow D
\]

\[ \epsilon \rightarrow \epsilon
\]

\[ D \rightarrow \epsilon
\]

\[ PDA \text{ to CFG: Intuitive description}
\]

Consider normalized PDA \( N = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}) \)

Starts in \( q_{\text{start}} \) with an empty stack

Ends in \( q_{\text{accept}} \) with an empty stack

In general, can define the language \( L_{pq} \), for any two states \( p, q \in Q \)

which is the language of all strings that start in \( p \) with an empty stack, and end in \( q \) with an empty stack

For each pair of states \( p \) and \( q \), define a symbol \( S_{pq} \) in the CFG for the language \( L_{pq} \)

Language of \( N \) is \( L_{q_{\text{start}}q_{\text{accept}}} \)

Steps to process \( w \in L_{pq} \)

Two possibilities:

During the processing of \( w \) the stack becomes empty at some intermediate state \( r \).

This means a word of \( L_{pq} \) can be formed by concatenating:

- a word of \( L_{pr} \) (which brought \( N \) from state \( p \) to state \( r \) with an empty stack)
- a word of \( L_{rq} \) (which brought \( N \) from state \( r \) to state \( q \) with an empty stack)

Stack is never empty in the middle of \( N \)'s transit from \( p \) to \( q \) in processing \( w \).

The first transition (from, say, \( p \) to \( p_1 \)) must have been a push.

The last transition (from, say, \( q_1 \) to \( q \)) must have been a pop.

The pop popped exactly the symbol pushed by the first transition from \( p \) to \( p_1 \).

Case where the stack is never empty between \( p \) and \( q \):

In other words:

The PDA read \( a \) from input as it moved from \( p \) to \( p_1 \).

The PDA read \( b \) from input as it moved from \( q_1 \) to \( q \).

Then \( w = ayb \), where \( y \) is an input that causes the PDA \( N \) to start from \( p \) with an empty stack and end in \( q \) with an empty stack i.e., \( y \in L_{p_1q_1} \).

Formally, if there is a push transition (pushing \( X \) onto the stack) from \( p \) to \( p_1 \) (reading \( a \)) and a pop transition from \( q_1 \) to \( q \) (popping \( X \) and reading \( b \)), then a word in \( L_{pq} \) can be constructed from the expression \( aL_{p_1q_1}b \).

Note that either or both of \( a \) or \( b \) could be \( \epsilon \).
The construction

For every state \( p \), introduce the rule

\[
A_{pp} \rightarrow \epsilon
\]

Empty string can always be considered as getting you from \( p \) to \( p \) without doing anything to the stack, since nothing was read

Concatenation rule

For the case where the stack empties in the middle of transition from \( p \) to \( q \), introduce, for all states \( p, q, r \) of \( N \), the rule

\[
A_{pq} \rightarrow A_{pr} \: A_{rq}
\]

Recursion rule

For the case where the stack is never empty, for any given states \( p, p_1, q_1, r \) of \( N \), such that there is a push transition from \( p \) to \( p_1 \) and a pop transition from \( q_1 \) to \( r \) (that push and pop the same symbol), introduce an appropriate rule

Formally, for \( p, p_1, q_1, r \) of \( N \) with the form

introduce the rule

\[
A_{pr} \rightarrow aA_{p_1q_1}b
\]
Formal definition (from Sipser)

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$ be a PDA

There is a context-free grammar $G$ with non-terminals \{A_{pq} \mid p, q \in Q\}

Rules:

- For each $p, q, r, s \in Q, t \in \Gamma, a, b \in \Sigma$, if $\delta(p, a, \epsilon)$ contains $(r, t)$ and $\delta(s, b, t)$ contains $(q, \epsilon)$, put the rule $A_{pq} \rightarrow aA_{rs}b$ in $G$
- For each $p, q, r \in Q, p \neq r$, put the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ in $G$
- For each $p \in Q$, put the rule $A_{pp} \rightarrow \epsilon$ in $G$

The grammar

The rules for generating paths give a grammar to generate all labels of such paths.

The grammar has non-terminals $A_{qr}$ which will generate all strings $x$ that are processed when passing from state $q$ to state $r$.

Q: Under this assumption, what should the production body (right hand side) for the start variable $S$ be?

The Grammar Symbols

$A$: $S = A_{q_{\text{start}}q_{\text{accept}}}$, where $q_{\text{start}}$ is the start state and $q_{\text{accept}}$ is the final state.

In addition to this start variable, the other variables are all $A_{qr}$ for which there is a path going from $q$ to $r$ that starts and ends with an empty stack.

Note that Sipser doesn’t require the extra condition that there exist a path from $q$ to $r$ which starts and ends with an empty stack; his method generates all possible combinations. However, those pairs $qr$ for which no such path exists will create useless variables $A_q$, which end up cluttering the grammar and making the construction extremely ugly, even on the simplest PDAs. On the other hand, it is not obvious how one would determine a priori which of the pairs don’t have such paths, which probably explains why Sipser didn’t include this condition.

Grammar Rules

- **Basis rule:** Add a production $A_{qq} \rightarrow \epsilon$ for each state $q$ in the PDA.
- **Concatenation rule:** Add a production $A_{pr} \rightarrow A_{pq}A_{qr}$ for all $p, q, r$ when $A_{pr}, A_{pq}$ and $A_{qr}$ are all in $V$ (the variables of the CFG).
- **Recursion rule:** Add a production $A_{ps} \rightarrow aA_{qr}b$ for all $p, s, q, r$ when $A_{ps}$ and $A_{qr}$ are in $V$.

Transitions $(q, X) \in \delta(p, a, \epsilon), (s, \epsilon) \in \delta(r, b, X)$ for the same stack symbol $X$ exist in the PDA.
Example

PDA in the normalized form:

A: “CNP” = correctly nested parentheses, including sets of pairs [e.g., ()()]. The number of Xs on the stack reflects how deep the current nesting is.

Q: What are the variables for the equivalent grammar? What is the start variable?

A: \( V = \{ A_{qs}, A_{qq}, A_{rr}, A_{ss}, A_{rq}, A_{sq}, A_{sr}, A_{qr} \} \)

\( S = A_{qs} \)

Are there any useless variables?

- We don’t need \( A_{rq}, A_{sq}, A_{sr} \) because the paths go in the wrong direction
- We don’t need \( A_{qr} \) or \( A_{rs} \) because can’t add or remove $ while at \( r \)
  - I.e., no transition where you both begin and end with an empty stack

Productions from the Base Rule

Add a production \( A_{qq} \rightarrow \epsilon \) for each state \( q \) in the PDA

Empty string can always be considered as getting you from \( p \) to \( p \) without doing anything to the stack, since nothing was read

\( A_{qq} \rightarrow \epsilon \)
\( A_{rr} \rightarrow \epsilon \)
\( A_{ss} \rightarrow \epsilon \)
Productions from the concatenation rule

Add a production $A_{pq} \rightarrow A_{pq}$ for all $p, q$ when $A_{pq}$ are all in $V$

If you can get from some state $p$ to another state $p_1$, starting and ending with the stack empty (regardless of stack activity in the processing of moving from $p$ to $p_1$), and from $q$ to $q$ under the same conditions, then combine paths to get a path from $p$ to $q$.

$V = \{A_{qs}, A_{qq}, A_{rr}, A_{ss}\}$

Productions from the recursion rule

Add a production $A_{pq} \rightarrow aA_{pq}$ for all $p, q$ when $A_{pq}$ are in $V$

For any given states $p, p_1, q_1, q$ of $N$, such that there is a push transition from $p$ to $p_1$ and a pop transition from $q_1$ to $q$ (that push and pop the same symbol), i.e., there exist transitions $\delta(p, a, \epsilon)$ contains $(p_1, X)$ and $\delta(q_1, b, X)$ contains $(q, \epsilon)$, put the rule $A_{pq} \rightarrow aA_{pq}$

Full Grammar

$A_{qs} \rightarrow A_{rr} | A_{qq} A_{qs} | A_{qs} A_{ss}$
$A_{rr} \rightarrow \epsilon | A_{rr} A_{rr} | (A_{rr})$
$A_{qq} \rightarrow \epsilon | A_{qq} A_{qq}$
$A_{ss} \rightarrow \epsilon | A_{ss} A_{ss}$

Simplifications

Apparently $A_{qq}$ and $A_{ss}$ are purely self-referential, so there is no way to terminate them – that is, no string can be derived from them.

We can therefore remove the variables $A_{qq}, A_{ss}$

Becomes:

$A_{qs} \rightarrow A_{rr} | A_{qs}$
$A_{rr} \rightarrow \epsilon | A_{rr} A_{rr} | (A_{rr})$
Showing that the grammar works…

\[ A_{qs} \rightarrow A_{rr} | A_{qs} \]
\[ A_{rr} \rightarrow \varepsilon | A_{rr} A_{rr} | (A_{rr}) \]

Rename variables to get:

\[ S \rightarrow T | S \]
\[ T \rightarrow \varepsilon | TT | (T) \]

So the final (cleaned up) grammar is

\[ T \rightarrow \varepsilon | TT | (T) \]

Another example

Consider the language \( L = \{wcw^R \mid w \in \{a, b\}^* \} \).

A non-normalized PDA for this language is:

\[ a, \varepsilon \rightarrow a \]
\[ b, \varepsilon \rightarrow b \]
\[ a, a \rightarrow \varepsilon \]
\[ b, b \rightarrow \varepsilon \]
\[ s \rightarrow c, \varepsilon \rightarrow \varepsilon \]

Convert to normalized form

1. Create new start and accepting states
2. All transitions either pop or push except \( c, \varepsilon \rightarrow \varepsilon \); change to two transitions that push and pop a dummy symbol

Generate grammar

1. Add start symbol and a production \( A_{qq} \rightarrow \varepsilon \) for each state \( q \) in the PDA

\[ S \rightarrow A_{s'q} | A_{qq} \rightarrow \varepsilon \]
\[ A_{s'q} \rightarrow \varepsilon \]
\[ A_{qf} \rightarrow \varepsilon \]
\[ A_{ss} \rightarrow \varepsilon \]
\[ A_{aa} \rightarrow \varepsilon \]
Generate grammar

2. Add a production \( A_{pr} \rightarrow A_{pq} A_{qr} \) for all \( p, q, r \) when \( A_{pr}, A_{pq} \) and \( A_{qr} \) are all in \( V \)

\[ A_{s'} \rightarrow A_{s's'}A_{s'a} | A_{s'a}a \]

Generate grammar

3. Add a production \( A_{ps} \rightarrow aA_{qr}b \) for all \( p, s, q, r \) when \( A_{ps} \) and \( A_{qr} \) are in \( V \) and transitions \( (q, X) \in \delta(p, a, \varepsilon) \) and \( (s, \varepsilon) \in \delta(r, b, X) \) for the same stack symbol \( X \) exist in the PDA

\[ A_{s'} \rightarrow \$A_{sf}\$
\[ A_{sf} \rightarrow cA_{qq}\varepsilon | bA_{sf}b | aA_{sf}a \]

Final grammar

<table>
<thead>
<tr>
<th>Final grammar</th>
<th>More readable</th>
<th>Simplified</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow A_{s'o} )</td>
<td>( S \rightarrow T )</td>
<td>( \rightarrow Z )</td>
</tr>
<tr>
<td>( A_{s'o} \rightarrow \varepsilon )</td>
<td>( R \rightarrow \varepsilon )</td>
<td>( Z \rightarrow c )</td>
</tr>
<tr>
<td>( A_{oo} \rightarrow \varepsilon )</td>
<td>( U \rightarrow \varepsilon )</td>
<td>( Z \rightarrow bZb )</td>
</tr>
<tr>
<td>( A_{pp} \rightarrow \varepsilon )</td>
<td>( V \rightarrow \varepsilon )</td>
<td>( Z \rightarrow aZa )</td>
</tr>
<tr>
<td>( A_{qq} \rightarrow \varepsilon )</td>
<td>( W \rightarrow \varepsilon )</td>
<td></td>
</tr>
<tr>
<td>( A_{qq} \rightarrow \varepsilon )</td>
<td>( X \rightarrow \varepsilon )</td>
<td></td>
</tr>
<tr>
<td>( A_{pp} \rightarrow \varepsilon )</td>
<td>( T \rightarrow RT )</td>
<td></td>
</tr>
<tr>
<td>( A_{pp} \rightarrow \varepsilon )</td>
<td>( T \rightarrow TX )</td>
<td></td>
</tr>
<tr>
<td>( A_{pp} \rightarrow \varepsilon A_{pp} )</td>
<td>( T \rightarrow \varepsilon E )</td>
<td></td>
</tr>
<tr>
<td>( Z \rightarrow cV )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( Z \rightarrow bZb )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( Z \rightarrow aZa )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( R, U, W, X \) contribute only \( \varepsilon \) so can be eliminated
- \( T \rightarrow RT \) and \( T \rightarrow TX \) then become \( T \rightarrow T \), which is obviously unnecessary
- \( S \) is superfluous because it only gets you to \( T \)
- \( T \) is superfluous because it only gets you to \( Z \)