Topic on "Ambiguity" — Skipped

Chomsky Normal Form (CNF)
Two CFGs $G_1$ and $G_2$ are equivalent, denoted by $G_1 \equiv G_2$, if $L(G_1) = L(G_2)$

For automata (FA) and grammars (CFGs), we like to convert an automation or grammar into equivalent ones in some "standardized" forms:

- for their combinatorial properties and applications

Common CFGs in "normal forms":
- Chomsky Normal Form (CNF)
- Greibach Normal Form (GNF)

Will study CNF and CFGs (used in the proof of Pumping Lemma for context-free).
A CFG $G = (V, \Sigma, P, S)$ is in CNF if every production rule of $P$ is of the form

1. $A \rightarrow BC$ where $A, B, C \in V$ and $a \in \Sigma$

or $A \rightarrow a$

(Additional requirement: $B, C \in V - \{S\}$)

Then, what about the possibility of $\varepsilon \in L(G)$?

(If $\varepsilon \in L(G)$, then $S$ can not derive $\varepsilon$ with production rules of form $\Box$)

So, we augment $\Box$:

A CFG $G = (V, \Sigma, P, S)$ is in CNF if:

1. if $\varepsilon \notin L(G)$, then every production rule is of the form $A \rightarrow BC$ where $B, C \in V - \{S\}$ and $a \in \Sigma$

and

2. if $\varepsilon \in L(G)$, then $S \rightarrow \varepsilon$ is the only exception production rule of $P$. 
Obviously, we want to prove that:

\[ \forall G \in CFG, \exists G' \in CFG \text{ such that } \]
\[ G' \text{ is in CNF and } G = G', \quad L(G) = L(G') \]

Proceed in 3 steps:
1. Eliminate \( E \)-productions
2. Eliminate unit-productions
3. Convert to CNF

Step 1. Eliminate \( E \)-productions

A production of the form \( A \rightarrow E \) is called an \( E \)-production.

Certainly we cannot eliminate all possible \( E \)-productions (what if \( E \in L(G) \)?)?

In this step:
Find CFG \( G_1 = (V_1, \Sigma, P_1, S_1) \)
such that \( G = G_1 \), and
\( S_1 \) - "new" start variable for \( G_1 \)
(to make sure that \( S_1 \) does not appear on
the right-hand side of any production rule - see (1) and \( P_1 \) does not have any \( E \)-production
except possibly \( S_1 \rightarrow E \).
How do we obtain such CFG $G_1'$?

A variable $A \in V$ (in CFG $H$) is called nullable if $A \Rightarrow^* \epsilon$.

Can we determine the set $\text{null}(H)$ - the set of all nullable variables in $H$?

Recurision! (or iteration in program)

Basis: if $A \Rightarrow \epsilon$, a production, then $A \in \text{null}(H)$

Induction: if $A \Rightarrow X_1X_2 \ldots X_n$ is a production and $X_1, X_2, \ldots, X_n \in \text{null}(H)$ then $A \in \text{null}(H)$.

(by-product: $\epsilon \in L(H)$ iff start variable $\Rightarrow^* \epsilon$

iff start variable $\in \text{null}(H)$)

Once we have computed $\text{null}(G_1)$, we can construct CFG $G_1 = (V_1, I, P_1, S_1)$ described at the bottom of page 3 as follows:
\textbf{CFG} \( G = (V, \Sigma, P, S) \)

\textbf{CFG} \( G' = (V', \Sigma, P', S) \)

1. Include \( S_1 \rightarrow S \) into \( P_1 \).

2. For every production rule
   \[ A \rightarrow X_1X_2 \cdots X_n \in P \cup \{ S_1 \rightarrow S \} \]
   (where \( n \geq 1 \),
   \( X_1, X_2, \ldots, X_n \in V \cup \Sigma \)),

   we include all production rules of the following forms into \( P_1 \):
   \[ A \rightarrow X'_1X'_2 \cdots X'_n \]  \( \quad (\text{into } P_1) \)
   where
   \[ X'_i = \begin{cases} X_i & \text{if } X_i \notin \text{null}(G) \\ \epsilon & \text{or } X_i \in \text{null}(G) \end{cases} \]
   for \( i = 1, 2, \ldots, n \).

   Except that, when \( X_i \in \text{null}(G) \)
   for all \( i = 1, 2, \ldots, n \),

   we do not include the possibility
   \[ A \rightarrow \epsilon \epsilon \cdots \epsilon \]  \( n \text{ copies} \)
   into \( P_1 \).

\textbf{Remarks}:

1. A production rule \( A \rightarrow X_1X_2 \cdots X_n \in P \cup \{ S_1 \rightarrow S \} \),
   with \( X_1, X_2, \ldots, X_n \in \text{null}(G) \),

   we include \( \frac{2^n - 1}{2} \) productions in \( P_1 \)
   "exponential increase"
Remarks:

2. Can you see that, if \( \epsilon \in L(G) \)
\[ \Leftrightarrow S \in \text{null}(G) \]
eventually, we examine no production \( S_1 \rightarrow \epsilon \)
and then include \( S_1 \rightarrow \epsilon \) into \( P_1 \).

as needed!

Step 2. Eliminate unit-productions

Now, the CFG \( G_1 = (V_1, \Sigma, P_1, S_1) \) is equivalent to \( G \) (can you see why \( G_1 \equiv G \) ?) and is free of productions except possibly \( S_1 \rightarrow \epsilon \in P_1 \).

We construct a CFG \( G_2 = (V_2, \Sigma, P_2, S_2) \)
such that \( G_2 \equiv G_1 \) and \( G_2 \) has no "unit" production.

A unit production is of no form
\[ A \rightarrow B \] with variables \( A, B \)
\[(\text{so } A \rightarrow a \text{ with terminal } a, \text{ is non-unit})\]
How can we safely eliminate a unit-production from \( G_1 \) (remember, we need \( G_1 = G_2 \))?

Seems that:
- For every unit-production \( A \rightarrow B \in P_1 \)
- And every nonunit-production \( B \rightarrow \alpha \in P_1 \)
  
  *We include* \( A \rightarrow \alpha \) *in* \( P_2 \)  

**Correct decision:**

\[
\text{CFG } G_1 = (V, \Sigma, P_1, S_1) \quad \text{CFG } G_2 = (V_2, \Sigma, P_2, S_1)
\]

if \( \quad A \Rightarrow^* B \) \hspace{1cm} \text{(in } G_1 \text{)}
and (every) \( B \rightarrow \alpha \) \hspace{1cm} \text{is non-unit in } P_1

Then we include \( A \rightarrow \alpha \) \hspace{1cm} \text{into } P_2

The detection \( "A \Rightarrow^* B" \) is not a simple
look-up in \( P_1 \)!

Is it difficult to decide \( A \Rightarrow^* B \)
when \( A, B \in V_1 \)?
How can \( A \Rightarrow^* B \)?

Remember, in \( G_1 \), there is no \( \epsilon \)-production (and \( S_1 \) does not appear in the right-hand side of any production), no derivation \( A \Rightarrow^* B \)

must be equivalent to a multi-step derivation of a "unit"-production:

\[
A = C_1 \Rightarrow a_1 \Rightarrow C_2 \Rightarrow a_2 \Rightarrow \ldots \Rightarrow a_n \Rightarrow C_n = B
\]

for some \( n \geq 1 \) and \( C_1, C_2, \ldots, C_n \in V_1 \)

(Can \( n \) be arbitrarily large?)

Note that, by Pigeonhole principle, we can limit such \( n \) to the number of variables in \( V_1 \).

Thus, the detection \( A \Rightarrow^* B \) is equivalent to detection of \( A = C_1 \Rightarrow C_2 \Rightarrow \ldots \Rightarrow C_n = B \)

for some \( n \leq |V_1| \).
Step 3. Convert to CNF.
Now, we have $CFG \ G_2 = (V_2, \Sigma, P_2, S_2)$
such that $G_2 = G_1 = G$ (why?)
$G_2$ is free of $\varepsilon$-production (why?)
except possibly $S_2 \rightarrow \varepsilon$ (into $P_2$)
and $G_2$ is free of unit-production (why?)

Finally, we convert $G_2$ to a $CFG \ G'$ in CNF
as desired.

$CFG \ G_2 = (V_2, \Sigma, P_2, S_2) \quad CFG \ G' = (V_1, \Sigma, P', S_1)$
in CNF

1. If $S_2 \rightarrow \varepsilon \in P_2$ then include $S_2 \rightarrow \varepsilon$ into $P'$

2. For every $A \rightarrow \alpha \in P_2$:
   
   Case when $|\alpha| = 1$:
   
   Note that $A \rightarrow \alpha$ must be non-unit,
   (i.e., no rule from $A \rightarrow \alpha_i$ in $\Sigma$)
   
   Then, include $A \rightarrow \alpha$ into $P'$

   Case when $|\alpha| \geq 2$:
   
   ?
Case when $|x| > 2$:

Say, $A \rightarrow x$ is of the form

$A \rightarrow X_1 X_2 \ldots X_n$ with $n \geq 2$.

By introducing new variables in $G'$, we may assume that each $X_i$ is a "variable".

For example: $A \rightarrow B a a \& A$

We include new variables $C_a, C_b$ into $V'$

and productions $C_a \rightarrow a$, $C_b \rightarrow b$ into $P'$.

Then, include the production $A \rightarrow B C_a C_b A$ into $P'$.

Hence, we may assume that the production $A \rightarrow X_1 X_2 \ldots X_n$ with $n \geq 2$

of $X_1, X_2, \ldots, X_n \in V'$

all variables

How do we achieve the "binary form" $\eta$

CNF ?
Example: $A \rightarrow BCAB$ produces

View $A \rightarrow BCABB$

as $A \rightarrow BY_1 \in P'$ (in CNF)

and $Y_1 \rightarrow CABB$ (not yet in CNF)

$\rightarrow Y_2$

as $Y_1 \rightarrow CY_2 \in P'$ (in CNF)

and $Y_2 \rightarrow ABB$ (not yet in CNF)

$\rightarrow Y_3$

as $Y_2 \rightarrow AY_3$ (in CNF)

and $Y_3 \rightarrow BB$ (in CNF)

Our descriptions (in three steps) are in more details than the text [Sip12].

Read Example 2.10 in [Sip12].
PDA - finite automaton with a "memory" in the form of stack
- One stack
- Unbounded depth

"pushdown store" ~ stack

Informally:

1-way read-only input tape

Current "configuration"

Next "configuration"
Formally, a PDA $M$ is a 6-tuple

$$(Q, \Sigma, \Gamma, \delta, q_0, F)$$

where $Q$, $\Sigma$, $q_0$, and $F$ are defined as for FAs, and 

$\Gamma$ - stack alphabet

(mostly $\Sigma \cap \Gamma = \emptyset$

but not necessarily)

and 

$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \to \mathcal{P}(Q \times (\Gamma \cup \{\epsilon\}))$

non-deterministic (1-step) transition function

power set of $Q \times (\Gamma \cup \{\epsilon\})$

Computation (1-step and multi-step)

acceptance

1-step:

$\delta(q, a, A)$ with $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$, $A \in \Gamma \cup \{\epsilon\}$

$$= \begin{cases} \{(p_1, B_1), (p_2, B_2), \ldots, (p_n, B_n)\} & \text{when } n \geq 0, \ P_i \in Q \ \text{for } i = 1, 2, \ldots, n, \ B_i \in \Gamma \cup \{\epsilon\} \\ \emptyset & \text{when } n = 0 \end{cases}$$

when $n = 0$

$\delta(q, a, A) = \emptyset$ (no transition)

when $n \geq 1$

$|\delta(q, a, A)| \geq 1$ (one or more transitions)
Examine one of the nondeterministic moves/ transitions: 
$s(q, a, A)$ includes $(p_1, B_1)$, say.

- When $a = \varepsilon$, the transition is $\varepsilon$-move.
  (read-head is not advanced)

- When $a \in \Sigma$, the transition is regular move.
  (read-head is advanced)

- Note that $A \in \Gamma \cup \{ \varepsilon \}$ and $B_1 \in \Gamma \cup \{ \varepsilon \}$:
  $A \in \Gamma \cup \{ \varepsilon \}$ combinations

  $A = \varepsilon \land B = \varepsilon$: the PDA "ignores" no stack.
  (in that configuration)

  $A = \varepsilon \land B \in \Gamma$: the PDA "pushes" symbol $B$
  onto tmp-stack

  $A \in \Gamma \land B = \varepsilon$: the PDA "pops" symbol $A$
  out of tmp-stack

  $A \in \Gamma \land B \in \Gamma$: the PDA overwrites symbol $A$
  by symbol $B$ at tmp-stack

  [actually: a pop-move followed by a push-move of $A$]
Multi-step: (Similar to NFAs with the additional data structure of a stack)

A string $x \in \Sigma^*$ is accepted by $M$

if we can write $x = a_1 a_2 \ldots a_n$ such that

- for some $n > 0$ when $n = 0$: $a_1 = \varepsilon$
- $a_i \in \Sigma \cup \{\varepsilon\}$ for $i = 1, 2, \ldots, n$

such that there exist

- a sequence of states $p_0, p_1, \ldots, p_{n+1} \in Q$
- and a sequence of stack contents $a_0, a_1, \ldots, a_{n+1} \in \Sigma^*$

\[
\begin{align*}
\text{state: } &p_0 &p_1 &p_2 & &p_3 & &p_4 & &p_{n+1} \\
\text{stack: } &a_0 &a_1 &a_2 & &a_3 & &a_4 & &a_{n+1}
\end{align*}
\]

\[\begin{align*}
\{ \text{if } p_0 \in F \}
\end{align*}\]

\[\begin{cases}
\alpha_i = \lambda \beta \\
\alpha_{i+1} = B \beta \\
\delta(p_i, a_i, A) \text{ includes } (p_{i+1}, \lambda)
\end{cases}\]

\[L(M) = \{ x \in \Sigma^* \mid M \text{ accepts } x \} \]
Heuristics in constructing PDA:

- States: as for FAs
  for consumed input \( x \)
  M enters \( \delta \) iff "you can characterize the consumed \( x \)"

- Stack: a first-in last-out data structure that allows the PDA to remember.
  Remember: infinite amount of information on consumed input
  It is a "stack".

- States: also used for implementing "loops"/iterations

- Bottom-stack marker:
  Used to detect the "bottom stack"
  (imagine without a bottom-stack marker, how can the PDA detect the bottom stack?)
  \( \delta(q, a, \epsilon) = 5 \rightarrow 3 \)
  Does not necessarily mean the stack is actually empty!
Example 1. Back to our non-regular language

\[ L = \{ a^i b^i \mid i \geq 0 \} \]

No program \( S \) (in the PDA \( M \)):

\[ \begin{align*}
1. \text{as if the input } x \text{ to } M \text{ is in } L \\
\text{accepted by } M \\
\text{and general enough} \\
\text{not short/specific strings } \notin L \\
2. \text{Construct } S \text{ to accept such "general" } x \\
3. \text{Check to see if such } S \text{ is sufficient} \\
\text{to accept extreme cases of } x \\
\text{if not, need to "patch" } S \text{ and} \\
\text{re-check if } S \text{ works for "general" } x \text{ again}
\end{align*} \]

\[ \begin{align*}
4. \text{after we have verified that } S \text{ works for} \\
\text{general and extreme cases of } x, \\
\text{make sure that if } x \notin L, \\
S \text{ responds correctly by} \\
\{ \text{not accepting } x \} \text{ or } \{ \text{halting and not accepting} \\
\text{halting and "hang"} \}
\end{align*} \]
Back to Example 1: How do we use states and the stack?

(To accept a general string $x \in L$, i.e., of the form $a^n b^n$ where $n \geq 0$)

- We have learned that, just by using two states, an FA cannot accept such $x$.
  - We use the stack to remember the number of $a$ as in consumed input.

- Use states to implement two loops:
  - Loop to decrement each $a$ by one $A$ in the stack.
  - Loop to check each $b$ versus one $A$ in the stack.

- When the stack is empty (really empty),
  we have encountered $a^n b^n$ for some $n \geq 0$.
  - So, use bottom-stack marker.
$S(\delta, \epsilon, \epsilon) \text{ includes } (q_2, Z_0)$

$S(q_2, a, \epsilon) \text{ includes } (q_2, A)$

$q_2$ - loop to rememb er each $a$ by one $A$ in stack

$S(q_2, b, A) \text{ includes } (q_3, \epsilon)$

in $q_2$, when the first $b$ is read,
exit $q_2$ and enter $q_3$ - loop to check off $b$ and $A$

$S(q_3, b, A) \text{ includes } (q_3, \epsilon)$

$S(q_3, \epsilon, Z_0) \text{ includes } (q_4, \epsilon)$

in $q_3$, when $Z_0$ is exposed,
guess that there is no more input

Remember: no program $S$

as if no input should be accepted

also, as in FAs, if such guess is not correct, $M$ will "hang".
Check:

if $x \in L$ then $M$ accepts $x$:

1. general $x \ a^i b^i, i > 0$ $M$ works

2. extreme $x \ a^0 b^0 = \varepsilon$ Need to make $q_0$ as accepting as well

Once we add $q_0$, $q_0$, $q_4$ and $q_0$ are accepting

as an accepting, check that

$M$ still works correctly for 1

if $x \notin L$ then $M$ does not accept $x$:

$a^i b^j, i < j$ $M$ does not accept $x$

$a^i b^j, i > j$ $M$ does not accept $x$

not of the form $a^* b^*$ $M$ does not accept $x$