Normal Forms

What is a normal form?

A class of context free grammars with restrictive properties on the structure of a CFG in the class.

Why normal form?

Easier to parse a CFG in normal form.

How normal form?

Transformations that maintain equivalent grammars — grammars that generate the same language.
Overview of Transformations

1. Eliminate recursion on the start symbol:
   \[ A \rightarrow aSb \]

2. Eliminate \( \lambda \) rules: \( A \rightarrow \lambda \)

3. Eliminate chain rules: \( A \rightarrow B \)

4. Eliminate useless symbols:
   \[ S \quad \rightarrow \quad \text{Sb} \mid \lambda \]
   \[ A \quad \rightarrow \quad a \]

5. Attain Chomsky normal form:
   1. \( A \rightarrow BC \), where \( B \neq S \) and \( C \neq S \);
   2. \( A \rightarrow a \);
   3. \( S \rightarrow \lambda \).
Eliminating Recursion on Start Symbol

**Lemma 5.1.1** Let $G = (V, \Sigma, P, S)$ be a CFG. Then there is a CFG $G' = (V', \Sigma, P', S')$ equivalent to $G$ such that every rule of $G'$ is of the form

$$A \rightarrow w,$$

where $A \in V'$ and $w \in ((V - \{S\}) \cup \Sigma)^*$. 

**Proof.** If $S$ does not appear on the right-hand side of any rules in $P$, then $G' = G$. Otherwise, choose $S' \notin V \cup \Sigma$, and set

$$V' = V \cup \{S'\},$$

$$P' = P \cup \{S' \rightarrow S\}.$$
Example

\[ S \rightarrow SS \mid (S) \mid \lambda \]

Add new start symbol \( S' \) and obtain:

\[ S' \rightarrow S \]

\[ S \rightarrow SS \mid (S) \mid \lambda \]
Eliminating $\lambda$ Rules

Example 5.1.2

$$S \rightarrow ACA$$

$$A \rightarrow aAa | B | C$$

$$B \rightarrow bB | b$$

$$C \rightarrow cC | \lambda$$

A **nullable variable** is one that can derive $\lambda$. Example: $\{C, A, S\}$.

A **noncontracting grammar** has no $\lambda$ rules.

An **essentially noncontracting grammar** has no nullable variables except perhaps $S$. 
Finding Nullable Variables

Algorithm 5.1.2

Find_Nullable(G)

▷ G = (V, Σ, P, S)

NULL ← \{A | (A → λ) ∈ P\}

repeat
    PREV ← NULL
    for A ∈ V
        do for (A → w) ∈ P
            do if w ∈ PREV*
                then NULL ← NULL ∪ A
    until NULL = PREV

return NULL

Run on Example 5.1.2
Constructing a Noncontracting Grammar

Delete every $\lambda$ rule.

For any rule $A \rightarrow w$, where $w$ contains nullable variables, add rules $A \rightarrow w'$, where $w' \neq \lambda$ and some of the occurrences of nullable variables have been deleted.

Example 5.1.2

\[
\begin{align*}
S & \rightarrow ACA \mid AC \mid CA \mid AA \mid A \mid C \\
A & \rightarrow aAa \mid aa \mid B \mid C' \\
B & \rightarrow bB \mid b \\
C' & \rightarrow cC' \mid c
\end{align*}
\]

Really need an essentially noncontracting grammar
Exercise

Chapter 5, Exercise 5:

Construct an equivalent essentially noncontracting grammar for

\[
S \rightarrow ABC \mid aBC \\
A \rightarrow aA \mid BC \\
B \rightarrow bB \mid \lambda \\
C \rightarrow cC \mid \lambda
\]
Eliminating Chain Rules

- Start with an essentially noncontracting grammar \( G = (V, \Sigma, P, S) \).

- For each \( A \in V \), compute

\[
\text{CHAIN}(A) = \{ B \in V \mid A \xrightarrow{G}^* B \}.
\]

- Delete all chain rules from \( P \).

- If \( A \xrightarrow{G}^* B \) and \( B \rightarrow w \) is not a chain rule, then include rule \( A \rightarrow w \).
Example

Example 5.2.1

Find the CHAIN sets for this grammar:

\[
S \rightarrow ACA \mid CA \mid AA \mid AC \mid A \mid C \mid \lambda \\
A \rightarrow aAa \mid aa \mid B \mid C \\
B \rightarrow bB \mid b \\
C \rightarrow cC \mid c
\]

and construct an equivalent grammar without chain rules.
Example (continued)

Algorithm: Build a graph whose arcs are the chain rules:

```
S → A → B
    ↓   
    C
```

and read off the chain sets:

\[
\text{CHAIN}(S) = \{S, A, B, C\} \quad \text{CHAIN}(B) = \{B\}
\]

\[
\text{CHAIN}(A) = \{A, B, C\} \quad \text{CHAIN}(C) = \{C\}
\]

Construct the equivalent grammar without chain rules:

\[
S \rightarrow ACA | CA | AA | AC | aAa | aa
\]
\[
\hspace{1cm} | bB | b | cC' | c | \lambda
\]

\[
A \rightarrow aAa | aa | bB | b | cC' | c
\]

\[
B \rightarrow bB | b
\]

\[
C \rightarrow cC' | c
\]
Theorem on Chain Rules

**Theorem 5.2.3** Let $G$ be an essentially noncontracting CFG. Then there is a CFG $G'$ equivalent to $G$ such that $G'$ has no chain rules.
Eliminating Useless Symbols

Let $G = (V, \Sigma, P, S)$ be a CFG. A symbol $x \in (V \cup \Sigma)$ is **useful** if

$$S \xrightarrow{\ast \ G} u x v \xrightarrow{\ast \ G} w,$$

for some $u, v \in (V \cup \Sigma)^*$ and $w \in \Sigma^*$. If $x \in (V \cup \Sigma)$ is not useful, then it is **useless**.

Example (page 129):

$$S \rightarrow AC | BS | B$$

$$A \rightarrow aA | aF$$

$$B \rightarrow CF | b$$

$$C \rightarrow cC | D$$

$$D \rightarrow aD | BD | C$$

$$E \rightarrow aA | BSA$$

$$F \rightarrow bB | b$$
Algorithms

1. **Algorithm 5.3.2.** Construct

   \[ \text{TERM} = \left\{ A \mid A \xrightarrow{+\cdot G} w \text{ with } w \in \Sigma^* \right\}. \]

2. Delete variables that do not generate terminal strings:

   \[ G'' = \{ V - \text{TERM}, \Sigma, P', S \}. \]

3. **Algorithm 5.3.4.** Construct

   \[ \text{REACH} = \left\{ A \mid S \xrightarrow{\ast\cdot G} uAv \right\}. \]

4. Delete variables that are not reachable:

   \[ G''' = \{ V - \text{TERM} - \text{REACH}, \Sigma, P'', S \}. \]
Variables That Derive Terminal Strings

• Example (page 129):

\[
G : S \rightarrow AC | BS | B \\
A \rightarrow aA | aF \\
B \rightarrow CF | b \\
C \rightarrow cC | D \\
D \rightarrow aD | BD | C \\
E \rightarrow aA | BSA \\
F \rightarrow bB | b
\]

• Find variables that derive terminal strings by a bottom-up search.

\[
\text{TERM} = \{B, F, S, A, E\}
\]

• What is \(G'\)?
Reachable Variables

• Resulting grammar:

\[ G' : S \rightarrow BS \mid B \]
\[ A \rightarrow aA \mid aF \]
\[ B \rightarrow b \]
\[ E \rightarrow aA \mid BSA \]
\[ F \rightarrow bB \mid b \]

• Find reachable variables by a search starting at \( S \).

\[ \text{REACH} = \square ? \]

• What is \( G'' \) ?
Theorem on Useless Symbols

Theorem 5.2.3 Let $G$ be a CFG. Then there is a CFG $G'$ equivalent to $G$ such that $G'$ has no useless symbols.
Chomsky Normal Form

A CFG $G$ is in **Chomsky normal form** if all rules have one of these forms:

1. $A \rightarrow BC$, where $B \neq S$ and $C \neq S$ (binary rule);
2. $A \rightarrow a$;
3. $S \rightarrow \lambda$.

**Theorem 5.4.2.** For every CFL $L$, there is a Chomsky normal form grammar that generates $L$. 
Creating Binary Rules

• Example 5.4.1:

\[ S \rightarrow aABC \mid a \]
\[ A \rightarrow aA \mid a \]
\[ B \rightarrow bcB \mid bc \]
\[ C \rightarrow cC \mid c \]

• Need new variables and rules:

\[ S \rightarrow A'T_1 \mid a \]
\[ A' \rightarrow a \]
\[ T_1 \rightarrow AT_2 \]
\[ T_2 \rightarrow BC \]
\[ A \rightarrow A'A \mid a \]
\[ B \rightarrow B'T_3 \mid B'C' \]
\[ T_3 \rightarrow C'B \]
\[ C \rightarrow C'C \mid c \]
\[ B' \rightarrow b \]
\[ C' \rightarrow c \]
Reaching Chomsky Normal Form

1. Eliminate recursion on the start symbol:
   \[ A \rightarrow aSb \]

2. Eliminate \( \lambda \) rules: \( A \rightarrow \lambda \)

3. Eliminate chain rules: \( A \rightarrow B \)

4. Eliminate useless symbols:
   \[ S \rightarrow Sb \mid \lambda \]
   \[ A \rightarrow a \]

5. Attain Chomsky normal form by eliminating non-binary rules.
Exercise

For the CFG

\[
\begin{align*}
S & \rightarrow AB \mid BCS \\
A & \rightarrow aA \mid C \\
B & \rightarrow bbB \mid b \\
C & \rightarrow cC \mid \lambda
\end{align*}
\]

construct an equivalent CNF grammar. (Chapter 5, Exercises 4, 12, and 22)