Context-Free Grammars

Normal Forms

Chapter 11
Normal Forms

A normal form $F$ for a set $C$ of data objects is a form, i.e., a set of syntactically valid objects, with the following two properties:

- For every element $c$ of $C$, except possibly a finite set of special cases, there exists some element $f$ of $F$ such that $f$ is equivalent to $c$ with respect to some set of tasks.

- $F$ is simpler than the original form in which the elements of $C$ are written. By “simpler” we mean that at least some tasks are easier to perform on elements of $F$ than they would be on elements of $C$. 
Normal Forms

If you want to design algorithms, it is often useful to have a limited number of input forms that you have to deal with.

Normal forms are designed to do just that. Various ones have been developed for various purposes.

Examples:

- Clause form for logical expressions to be used in resolution theorem proving
- Disjunctive normal form for database queries so that they can be entered in a query by example grid.
- Various normal forms for grammars to support specific parsing techniques.
Clause Form for Logical Expressions

Given:

[1] \( \forall x ((\text{Roman}(x) \land \text{know}(x, \text{Marcus})) \rightarrow (\text{hate}(x, \text{Caesar}) \lor (\forall y (\exists z (\text{hate}(y, z) \rightarrow \text{thinkcrazy}(x, y))))) )\)

[2] \( \text{Roman}(\text{Paulus}) \)

[3] \( \neg \text{hate}(\text{Paulus}, \text{Caesar}) \)

[4] \( \text{hate}(\text{Flavius}, \text{Marcus}) \)

[5] \( \neg \text{thinkcrazy}(\text{Paulus}, \text{Flavius}) \)

Prove: \( \neg \text{know}(\text{Paulus}, \text{Marcus}) \)

Sentence [1] in clause form:

\( \neg \text{Roman}(x) \lor \neg \text{know}(x, \text{Marcus}) \lor \text{hate}(x, \text{Caesar}) \lor \neg \text{hate}(y, z) \lor \text{thinkcrazy}(x, y) \)
Disjunctive Normal Form for Queries

The Query by Example (QBE) grid:

<table>
<thead>
<tr>
<th>Category</th>
<th>Supplier</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(category = fruit and supplier = Aabco)

<table>
<thead>
<tr>
<th>Category</th>
<th>Supplier</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td>Aabco</td>
<td></td>
</tr>
</tbody>
</table>

(category = fruit or category = vegetable)
Disjunctive Normal Form for Queries

(category = fruit or category = vegetable)

<table>
<thead>
<tr>
<th>Category</th>
<th>Supplier</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vegetable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Disjunctive Normal Form for Queries

(category = fruit and supplier = Aabco)
or
(category = vegetable and supplier = Botrexco)

<table>
<thead>
<tr>
<th>Category</th>
<th>Supplier</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td>Aabco</td>
<td></td>
</tr>
<tr>
<td>vegetable</td>
<td>Botrexco</td>
<td></td>
</tr>
</tbody>
</table>
Disjunctive Normal Form for Queries

But what about:

\[(\text{category} = \text{fruit} \text{ or } \text{category} = \text{vegetable}) \text{ and } (\text{supplier} = \text{A} \text{ or } \text{supplier} = \text{B})\]

This isn’t right:

<table>
<thead>
<tr>
<th>Category</th>
<th>Supplier</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td>Aabco</td>
<td></td>
</tr>
<tr>
<td>vegetable</td>
<td>Botrexco</td>
<td></td>
</tr>
</tbody>
</table>
Disjunctive Normal Form for Queries

(category = fruit or category = vegetable) and (supplier = Aabco or supplier = Botrexco)

becomes

(category = fruit and supplier = Aabco) or (category = fruit and supplier = Botrexco) or (category = vegetable and supplier = Aabco) or (category = vegetable and supplier = Botrexco)

<table>
<thead>
<tr>
<th>Category</th>
<th>Supplier</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td>Aabco</td>
<td></td>
</tr>
<tr>
<td>fruit</td>
<td>Botrexco</td>
<td></td>
</tr>
<tr>
<td>vegetable</td>
<td>Aabco</td>
<td></td>
</tr>
<tr>
<td>vegetable</td>
<td>Botrexco</td>
<td></td>
</tr>
</tbody>
</table>
Normal Forms for Grammars

**Chomsky Normal Form**, in which all rules are of one of the following two forms:

- \( X \rightarrow a \), where \( a \in \Sigma \), or
- \( X \rightarrow BC \), where \( B \) and \( C \) are elements of \( V - \Sigma \).

Advantages:

- Parsers can use binary trees.
- Exact length of derivations is known:

```
S
  /\   /
 A  B
  /\   /
 A  B   B
     /\
    B B
```

\( a \hspace{1cm} a \hspace{1cm} b \hspace{1cm} b \hspace{1cm} b \)
Normal Forms for Grammars

**Greibach Normal Form**, in which all rules are of the following form:

- \( X \rightarrow a \beta \), where \( a \in \Sigma \) and \( \beta \in (V - \Sigma)^* \).

Advantages:

- Every derivation of a string \( s \) contains \(|s|\) rule applications.

- Greibach normal form grammars can easily be converted to pushdown automata with no \( \epsilon \)-transitions. This is useful because such PDAs are guaranteed to halt.
Normal Forms Exist

**Theorem:** Given a CFG $G$, there exists an equivalent Chomsky normal form grammar $G_C$ such that:

$$L(G_C) = L(G) - \{\varepsilon\}.$$ 

**Proof:** The proof is by construction.

**Theorem:** Given a CFG $G$, there exists an equivalent Greibach normal form grammar $G_G$ such that:

$$L(G_G) = L(G) - \{\varepsilon\}.$$ 

**Proof:** The proof is also by construction.
Converting to a Normal Form

1. Apply some transformation to $G$ to get rid of undesirable property 1. Show that the language generated by $G$ is unchanged.

2. Apply another transformation to $G$ to get rid of undesirable property 2. Show that the language generated by $G$ is unchanged and that undesirable property 1 has not been reintroduced.

3. Continue until the grammar is in the desired form.
Rule Substitution

\[ X \rightarrow aYc \]
\[ Y \rightarrow b \]
\[ Y \rightarrow ZZ \]

We can replace the \( X \) rule with the rules:

\[ X \rightarrow abc \]
\[ X \rightarrow aZZc \]
Rule Substitution

**Theorem:** Let $G$ contain the rules:

$$X \rightarrow \alpha Y \beta \quad \text{and} \quad Y \rightarrow \gamma_1 | \gamma_2 | \cdots | \gamma_n,$$

Replace $X \rightarrow \alpha Y \beta$ by:

$$X \rightarrow \alpha \gamma_1 \beta, \quad X \rightarrow \alpha \gamma_2 \beta, \quad \ldots, \quad X \rightarrow \alpha \gamma_n \beta.$$

The new grammar $G'$ will be equivalent to $G$. 
Rule Substitution

*Theorem*:

Let $G$ contain the rules:

$$X \rightarrow \alpha Y \beta \quad \text{and} \quad Y \rightarrow \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n$$

Replace $X \rightarrow \alpha Y \beta$ by:

$$X \rightarrow \alpha \gamma_1 \beta, \quad X \rightarrow \alpha \gamma_2 \beta, \quad \ldots, \quad X \rightarrow \alpha \gamma_n \beta.$$ 

The new grammar $G'$ will be equivalent to $G$. 
Rule Substitution

Replace $X \rightarrow \alpha Y\beta$ by:

$X \rightarrow \alpha \gamma_1\beta, \quad X \rightarrow \alpha \gamma_2\beta, \quad \ldots, \quad X \rightarrow \alpha \gamma_n\beta$.

Proof:

● Every string in $L(G)$ is also in $L(G')$:

If $X \rightarrow \alpha Y\beta$ is not used, then use same derivation.
If it is used, then one derivation is:
$S \Rightarrow \ldots \Rightarrow \delta X\phi \Rightarrow \delta \alpha Y\beta\phi \Rightarrow \delta \alpha \gamma_k\beta\phi \Rightarrow \ldots \Rightarrow w$

Use this one instead:
$S \Rightarrow \ldots \Rightarrow \delta X\phi \Rightarrow \delta \alpha \gamma_k\beta\phi \Rightarrow \ldots \Rightarrow w$

● Every string in $L(G')$ is also in $L(G)$: Every new rule can be simulated by old rules.
Conversion to Chomsky Normal Form

1. Remove all $\varepsilon$-rules, using the algorithm $\text{removeEps}$.

2. Remove all unit productions (rules of the form $A \to B$).

3. Remove all rules whose right hand sides have length greater than 1 and include a terminal:

   (e.g., $A \to aB$ or $A \to BaC$)

4. Remove all rules whose right hand sides have length greater than 2:

   (e.g., $A \to BCDE$)
Removing $\varepsilon$-Productions

Remove all $\varepsilon$ productions:

(1) If there is a rule $P \rightarrow \alpha Q \beta$ and $Q$ is nullable,

    Then: Add the rule $P \rightarrow \alpha \beta$.

(2) Delete all rules $Q \rightarrow \varepsilon$. 
Removing ε-Productions

Example:

\[ S \rightarrow aA \]
\[ A \rightarrow B \mid CDC \]
\[ B \rightarrow \varepsilon \]
\[ B \rightarrow a \]
\[ C \rightarrow BD \]
\[ D \rightarrow b \]
\[ D \rightarrow \varepsilon \]
Unit Productions

A *unit production* is a rule whose right-hand side consists of a single nonterminal symbol.

Example:

\[
S \rightarrow X \ Y \\
X \rightarrow A \\
A \rightarrow B \mid a \\
B \rightarrow b \\
Y \rightarrow T \\
T \rightarrow Y \mid c
\]
Removing Unit Productions

\[ \text{removeUnits}(G) = \]
1. Let \( G' = G \).
2. Until no unit productions remain in \( G' \) do:
   2.1 Choose some unit production \( X \rightarrow Y \).
   2.2 Remove it from \( G' \).
   2.3 Consider only rules that still remain. For every rule \( Y \rightarrow \beta \), where \( \beta \in V^* \), do:
      Add to \( G' \) the rule \( X \rightarrow \beta \) unless it is a rule that has already been removed once.
3. Return \( G' \).

After removing epsilon productions and unit productions, all rules whose right hand sides have length 1 are in Chomsky Normal Form.
Removing Unit Productions

\[
\text{removeUnits}(G) = \\
1. \text{Let } G' = G. \\
2. \text{Until no unit productions remain in } G' \text{ do:} \\
    2.1 \text{Choose some unit production } X \rightarrow Y. \\
    2.2 \text{Remove it from } G'. \\
    2.3 \text{Consider only rules that still remain. For every rule } Y \rightarrow \beta, \\
    \text{where } \beta \in V^*, \text{ do:} \\
    \text{Add to } G' \text{ the rule } X \rightarrow \beta \text{ unless it is a rule that } \\
    \text{has already been removed once.} \\
3. \text{Return } G'. \\
\]

Example:
\[
S \rightarrow X \ Y \\
X \rightarrow A \\
A \rightarrow B \mid a \\
B \rightarrow b \\
Y \rightarrow T \\
T \rightarrow Y \mid c
\]
Removing Unit Productions

\[ \text{removeUnits}(G) = \]

1. Let \( G' = G \).
2. Until no unit productions remain in \( G' \) do:
   2.1 Choose some unit production \( X \rightarrow Y \).
   2.2 Remove it from \( G' \).
   2.3 Consider only rules that still remain. For every rule \( Y \rightarrow \beta \), where \( \beta \in V^* \), do:
      Add to \( G' \) the rule \( X \rightarrow \beta \) unless it is a rule that has already been removed once.
3. Return \( G' \).

Example:

\[
\begin{align*}
S & \rightarrow X \ Y \\
X & \rightarrow A \\
A & \rightarrow B \mid a \\
B & \rightarrow b \\
Y & \rightarrow T \\
T & \rightarrow Y \mid c \\
A & \rightarrow a \mid b \\
B & \rightarrow b \\
T & \rightarrow c \\
X & \rightarrow a \mid b \\
Y & \rightarrow c
\end{align*}
\]
Mixed Rules

removeMixed(G) =
1. Let $G' = G$.
2. Create a new nonterminal $T_a$ for each terminal $a$ in $\Sigma$.
3. Modify each rule whose right-hand side has length greater than 1 and that contains a terminal symbol by substituting $T_a$ for each occurrence of the terminal $a$.
4. Add to $G$, for each $T_a$, the rule $T_a \rightarrow a$.
5. Return $G'$.

Example:

$A \rightarrow a$
$A \rightarrow a \ B$
$A \rightarrow B_a \ C$
$A \rightarrow B_b \ C$
Mixed Rules

\textit{removeMixed}(G) =

1. Let \( G' = G \).
2. Create a new nonterminal \( T_a \) for each terminal \( a \) in \( \Sigma \).
3. Modify each rule whose right-hand side has length greater than 1 and that contains a terminal symbol by substituting \( T_a \) for each occurrence of the terminal \( a \).
4. Add to \( G \), for each \( T_a \), the rule \( T_a \rightarrow a \).
5. Return \( G' \).

Example:

\begin{align*}
A & \rightarrow a \\
A & \rightarrow a \ B \\
A & \rightarrow B_a \ C \\
A & \rightarrow B_b \ C \\
A & \rightarrow T_a \ B \\
A & \rightarrow B T_a \ C \\
A & \rightarrow B T_b \ C \\
T_a & \rightarrow a \\
T_b & \rightarrow b
\end{align*}
Long Rules

\[ \text{removeLong}(G) = \]
1. Let \( G' = G \).
2. For each rule \( r \) of the form:
   \[ A \rightarrow N_1N_2N_3N_4...N_n, \quad n > 2 \]
   create new nonterminals \( M_2, M_3, \ldots M_{n-1} \).
3. Replace \( r \) with the rule \( A \rightarrow N_1M_2 \).
4. Add the rules:
   \[ M_2 \rightarrow N_2M_3, \]
   \[ M_3 \rightarrow N_3M_4, \ldots \]
   \[ M_{n-1} \rightarrow N_{n-1}N_n. \]
5. Return \( G' \).

Example:
\[ A \rightarrow BCDEF \]
An Example

\[
\begin{align*}
S & \rightarrow aACa \\
A & \rightarrow B \mid a \\
B & \rightarrow C \mid c \\
C & \rightarrow cC \mid \epsilon
\end{align*}
\]

`removeEps` returns:

\[
\begin{align*}
S & \rightarrow aACa \mid aAa \mid aCa \mid aa \\
A & \rightarrow B \mid a \\
B & \rightarrow C \mid c \\
C & \rightarrow cC \mid c
\end{align*}
\]
An Example

\[
\begin{align*}
S & \rightarrow aACa \mid aAa \mid aCa \mid aa \\
A & \rightarrow B \mid a \\
B & \rightarrow C \mid c \\
C & \rightarrow cC \mid c
\end{align*}
\]

Next we apply \textit{removeUnits}:
Remove \( A \rightarrow B \). Add \( A \rightarrow C \mid c \).
Remove \( B \rightarrow C \). Add \( B \rightarrow cC \) \((B \rightarrow c, \text{ already there})\).
Remove \( A \rightarrow C \). Add \( A \rightarrow cC \) \((A \rightarrow c, \text{ already there})\).

So \textit{removeUnits} returns:
\[
\begin{align*}
S & \rightarrow aACa \mid aAa \mid aCa \mid aa \\
A & \rightarrow a \mid c \mid cC \\
B & \rightarrow c \mid cC \\
C & \rightarrow cC \mid c
\end{align*}
\]
An Example

\[\begin{align*}
S & \rightarrow aACa | aAa | aCa | aa \\
A & \rightarrow a | c | cC \\
B & \rightarrow c | cC \\
C & \rightarrow cC | c
\end{align*}\]

Next we apply \textit{removeMixed}, which returns:

\[\begin{align*}
S & \rightarrow TaACTa | TaATa | TaCTa | TaTa \\
A & \rightarrow a | c | TcC \\
B & \rightarrow c | TcC \\
C & \rightarrow TcC | c \\
Ta & \rightarrow a \\
Tc & \rightarrow c
\end{align*}\]
An Example

\[ S \rightarrow T_aACT_a \mid T_aAT_a \mid T_aCT_a \mid T_aT_a \]

\[ A \rightarrow a \mid c \mid T_cC \]

\[ B \rightarrow c \mid T_cC \]

\[ C \rightarrow T_cC \mid c \]

\[ T_a \rightarrow a \]

\[ T_c \rightarrow c \]

Finally, we apply \textit{removeLong}, which returns:

\[ S \rightarrow T_aS_1 \quad S \rightarrow T_aS_3 \quad S \rightarrow T_aS_4 \quad S \rightarrow T_aT_a \]

\[ S_1 \rightarrow AS_2 \quad S_3 \rightarrow AT_a \quad S_4 \rightarrow CT_a \]

\[ S_2 \rightarrow CT_a \]

\[ A \rightarrow a \mid c \mid T_cC \]

\[ B \rightarrow c \mid T_cC \]

\[ C \rightarrow T_cC \mid c \]

\[ T_a \rightarrow a \]

\[ T_c \rightarrow c \]
The Price of Normal Forms

\[ E \rightarrow E + E \]
\[ E \rightarrow (E) \]
\[ E \rightarrow \text{id} \]

Converting to Chomsky normal form:

\[ E \rightarrow E E' \]
\[ E' \rightarrow P E \]
\[ E \rightarrow L E'' \]
\[ E'' \rightarrow E R \]
\[ E \rightarrow \text{id} \]
\[ L \rightarrow ( \]
\[ R \rightarrow ) \]
\[ P \rightarrow + \]

Conversion doesn’t change weak generative capacity but it may change strong generative capacity.
Island Grammars

Suppose that:

● Inputs are ill-formed:

  ● “Um, I uh need a copy of uh my bill for er Ap, no May, I think, or June, maybe all of them uh, I guess that would work.”

  ● <b><i>bold italic</b></i>

● Inputs are not well understood.
Island Grammars

Suppose that:

- Inputs are a blend of multiple languages:
  - Spanglish
  - A Web page with HTML, Java, etc.

- Inputs contain parts we care about and parts we don’t:
  - Extracting the top-level structure of an XML document and ignoring the content.

A key application: reverse engineering.
Island Grammars

An island grammar contains:

- A set of detailed rules that describe the fragments that we care about. We’ll call these fragments *islands*.

- A set of flexible rules that can match everything else. We’ll call everything else the *water*.

We could use regular expressions to find islands. But what if we need to match delimiters?

A context-free island grammar is a hybrid between a context-free grammar and a set of patterns described as regular expressions.
An Island Grammar

[1] <input> → <chunk>*
[2] <chunk> → CALL <id> (<expr>) {cons(CALL)}
[3] <chunk> → CALL ERROR (<expr>) {reject}
[4] <chunk> → <water>
[5] <water> → Σ* {avoid}
Stochastic Context-Free Grammars

A stochastic context-free grammar $G$ is a quintuple: $(V, \Sigma, R, S, D)$:

- $V$ is the rule alphabet,
- $\Sigma$ is a subset of $V$,
- $R$ is a finite subset of $(V - \Sigma) \times V^*$,
- $S$ can be any element of $V - \Sigma$,
- $D$ is a function from $R$ to $[0 - 1]$.

For every nonterminal symbol $n$, the sum of the probabilities associated with all rules whose left-hand side is $n$ must be 1.
Stochastic Context-Free Example

PalEven = \{ww^R : w \in \{a, b\}^*\}.

But now suppose:
- a’s occur three times as often as b’s do.

\[ G = (\{S, a, b\}, \{a, b\}, R, S, D): \]
\[
\begin{align*}
S & \rightarrow aSa \quad [\cdot72] \\
S & \rightarrow bSb \quad [\cdot24] \\
S & \rightarrow \varepsilon \quad [\cdot04]
\end{align*}
\]
Stochastic Context-Free Grammars

The probability of a particular parse tree $t$:

Let $C$ be the collection (in which duplicates count) of rules $r$ that were used to generate $t$. Then:

$$\Pr(t) = \prod_{r \in C} \Pr(r)$$

Example:

- $S \rightarrow aSa$ [.72]
- $S \rightarrow bSb$ [.24]
- $S \rightarrow \varepsilon$ [.04]

$S \Rightarrow aSa \Rightarrow aaSaSaa \Rightarrow aabSbaa \Rightarrow aabbaa$

$=.72 \times .72 \times .24 \times .04 = .00497664$
Stochastic Context-Free Grammars

Stochastic grammars can be used to answer two important kinds of questions:

● In an error-free environment: given $s$, find the most likely parse tree for it.

● In a noisy environment: given a set of possible true strings $X$ and an observed string $o$, find the particular string $s$ (and possibly also the most likely parse for it) that is most likely to have been the one that was actually generated.
Stochastic Context-Free Grammars

- The noisy environment problem: Finding the probability that string $w$ was generated: If $T$ is the set of possible parse trees for $w$, then the total probability of generating $w$ is:

$$\Pr(w) = \sum_{t \in T} \Pr(t).$$

So the highest probability sentence $s$ is:

$$s = \arg \max_{w \in X} \Pr(w \mid o)$$

$$= \arg \max_{w \in X} \frac{\Pr(o \mid w) \Pr(w)}{\Pr(o)}.$$


Complementary base pairs (C – G and A – U) form hydrogen bonds that determine the 3-D structure of the molecule. Other pairs (e.g., G – U) can also bond, but with lower probability.
Three Dimensional RNA Structure

(a)

(b)

<family> → <tail> <stemloop> [1]
<tail> → <base> <base> <base> [1]
<stemloop> → C <stemloop-5> G [.23]
<stemloop> → G <stemloop-5> C [.23]
<stemloop> → A <stemloop-5> U [.23]
<stemloop> → U <stemloop-5> A [.23]
<stemloop>→ G <stemloop-5> U [.03]
<stemloop>→ U <stemloop-5> G [.03]
<stemloop-5>→ …
Stochastic context-free grammars and generation

Stochastic CFGs can generate very realistic English texts.

Applications:
- Fake papers: SClgen
- Spam obfuscation