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 for Context-free 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

Normal Forms for Context-free 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 ε-transitions. This is useful because such PDAs are guaranteed to halt.

Rule Substitution

$X \rightarrow \mathtt{a}Y\mathtt{c}$
$Y \rightarrow \mathtt{b}$
$Y \rightarrow ZZ$

We can replace the X rule with the rules:

$X \rightarrow \mathtt{abc}$
$X \rightarrow \mathtt{a}ZZ\mathtt{c}$

$$X \Rightarrow \mathtt{a}Y\mathtt{c} \Rightarrow \mathtt{a}ZZ\mathtt{c}$$

Rule Substitution

Theorem: Let G contain the rules:

$$X \to \alpha Y \beta \qquad \text{and} \qquad Y \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n \,,$$

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

$$X \to \alpha\gamma_1\beta, \qquad X \to \alpha\gamma_2\beta, \qquad \ldots, \qquad X \to \alpha\gamma_n\beta.$$

The new grammar G' will be equivalent to G.

Rule Substitution

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

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

Proof:

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

  If $X \to \alpha Y \beta$ 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:
  $S \Rightarrow \ldots \Rightarrow \delta X \phi \Rightarrow \quad\quad\quad \delta\alpha\gamma_k\beta\phi \Rightarrow \ldots \Rightarrow w$

- Every string in $L(G')$ is also in $L(G)$: Every new rule can be simulated by old rules.

Normal Forms for Context-free Grammars

Definition: A symbol X in V is *useless* in a CFG G=(V, $\sum$, R, S) if there does not exist a derivation of the form $S \Rightarrow^* wXy \Rightarrow^* wxy$ where w, x, y are in $\sum^*$.

Definition: A symbol X in V is *inaccessible* in a CFG G = (V, $\sum$, R, S) if X does not appear in any sentential form. (inaccessible implies not *reachable*)

Definition: A symbol X is *generating* if $X \Rightarrow^* w$ for some w in $\sum^*$.

Normal Forms for Context-free Grammars

Algorithm (NF1) (Is L(G) empty?)

Input: A CFG G = (V, ∑, R, S).

Output: "YES" if L(G) = ∅, "NO" otherwise.

Method: Construct sets N_0, N_1, …recursively as follows:

1. $N_0 = \emptyset$; i = 1;
2. $N_i = \{A \mid A \rightarrow \alpha \text{ in R and } \alpha \text{ in } (N_{i-1} \cup \sum)^* \} \cup N_{i-1}$;
3. If $N_i \neq N_{i-1}$ then begin i = i +1; go to 2; end;
4. $N_e = N_i$;
5. If S is in N_e then output "NO" else output "YES".

Normal Forms for Context-free Grammars

Example:

$S \rightarrow AA \mid AB$

$A \rightarrow SA \mid AB$

$B \rightarrow BB \mid b$

Normal Forms for Context-free Grammars

Algorithm (NF2):

(Removal of inaccessible symbols)

Input: A CFG $G = (V, \sum, R, S)$.

Output: CFG $G' = (V', \sum', R', S)$ such that

(1) $L(G') = L(G)$,

(2) No X in V' is unreachable.

Method: Construct sets $N_0, N_1, \ldots$ recursively as follows:

1. $N_0 = \{S\}$; $i = 1$;
2. $N_i = \{X \mid \text{some } A \rightarrow \alpha X \beta \text{ is in R and A in } N_{i-1}\} \cup N_{i-1}$;
3. If $N_i \neq N_{i-1}$ then begin i=i+1; go to 2; end;
4. $V' = N_i \cap V$; $\sum' = N_i \cap \sum$; R' = those productions of R with symbols from N_i; $S = S$;

Normal Forms for Context-free Grammars

Algorithm (NF3): (Useless Symbol Removal)

Input: A CFG $G = (V, \sum, R, S)$.

Output: CFG $G' = (V', \sum', R', S)$ such that

(1) $L(G') = L(G)$,

(2) No X in V' is useless.

Method:

1. Apply algorithm NF1 to G to get N_e.
2. Let $G_1 = ((V \cap N_e,) \cup \sum, \sum, R_1, S)$ where R_1 contains those productions of R involving only symbols from $N_e \cup \sum$.
3. Apply algorithm NF2 to G_1 to obtain $G' = (V', \sum', R', S)$

Normal Forms for Context-free Grammars

Example:

$S \rightarrow a \mid A$

$A \rightarrow AB$

$B \rightarrow b$

Normal Forms for Context-free Grammars

Definitions:

1. $A \rightarrow \varepsilon$ is called *ε-production*
2. $A \rightarrow \alpha$ is called *A-production*
3. $A \rightarrow B$ where A and B are in V is called *unit production* (or *single production*).
4. A variable A is *nullable* if $A \Rightarrow^* \varepsilon$.
5. A CFG $G = (V, \sum, R, S)$ is *ε-free* if either (i) R has no ε-productions, or (ii) there is exactly one ε-production $S \rightarrow \varepsilon$ and S does not appear on the right side of any production in R.

Normal Forms for Context-free Grammars

Algorithm (NF4): Conversion to ε-free grammar

Input: A CFG $G = (V, \sum, R, S)$.

Output: Equivalent ε-free CFG $G' = (V', \sum', R', S')$

Method:

1) Construct $N_e = \{A \mid A \text{ in } V\text{-}\sum \text{ and } A \Rightarrow^+ \varepsilon\}$
2) If $A \rightarrow \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \ldots B_k \alpha_k$ is in R, $k \geq 0$ and for $1 \leq i \leq k$ each B_i in N_e but no symbol in any α_j is in N_e, $0 \leq j \leq k$, add to P′ all productions of the form $A \rightarrow \alpha_0 X_1 \alpha_1 X_2 \alpha_2 \ldots X_k \alpha_k$ where X_i is either B_i or ε, without adding $A \rightarrow \varepsilon$ to R′
3) If S is in N_e add to R′ the productions $S' \rightarrow \varepsilon \mid S$ where S′ is a new symbol and set $V' = V \cup \{S'\}$. Otherwise set $V' = V$ and $S' = S$.
4) $G' = (V', \sum, P', S')$

Normal Forms for Context-free Grammars

Example:

$S \rightarrow aSbS \mid bSaS \mid \varepsilon$

Normal Forms for Context-free Grammars

Algorithm (NF5): Removal of unit productions

Input: An ε-free CFG $G = (V, \sum, R, S)$.

Output: Equivalent ε-free CFG $G' = (V, \sum, R', S)$

Method

1. Construct for each A in V-$\sum$ the sets $N_A = \{B \mid A \Rightarrow^* B\}$ as follows:
   (a) $N_0 = \{A\}$; $i = 1$;
   (b) $N_i = \{C \mid B \rightarrow C \text{ is in } R \text{ and } B \text{ in } N_{i-1}\} \cup N_{i-1}$
   (c) If $N_i \neq N_{i-1}$ then $i=i+1$; go to (b); else $N_A = N_i$
2. Construct R' as follows: If $B \rightarrow \alpha$ is in R and not a unit production, place $A \rightarrow \alpha$ in R' for A such that B is in N_A.
3. $G' = (V, \sum, R', S)$

Normal Forms for Context-free Grammars

Example:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$E \rightarrow (E) \mid a$

Normal Forms for Context-free Grammars

Definition: A CFG G = (V, ∑, R, S) is said to be *cycle-free* if there is no derivation of the form $A \Rightarrow^* A$ for any A in V-∑. G is said to be *proper* it it is cycle-free, ε-free, and has no useless symbols.

ε-free and no unit productions imply cycle-free.

Chomsky Normal Form

A CFG G = (V, $\sum$, R, S) is said to be in *Chomsky Normal Form (CNF)* if each production in R is one of the forms

(1) $A \rightarrow BC$ with A, B, C in V-$\sum$, or

(2) $A \rightarrow a$ with a in $\sum$, or

(3) If ε is in L(G), then $S \rightarrow \varepsilon$ is a production and S does not appear on the right side of any production

Chomsky Normal Form

Algorithm (NF6) Conversion to CNF

Input: A proper CFG $G=(V, \sum, R, S)$ with no single production

Output: A CFG $G_1=(V_1, \sum, R_1, S)$ such that $L(G) = L(G_1)$

Method: From G, construct G_1 as follows:

1) Add each production of the form $A \rightarrow a$ in R to R_1
2) Add each production of the form $A \rightarrow BC$ in P to P_1 (A, B, C are non terminals)
3) 3) If $S \rightarrow \varepsilon$ is in R, add $S \rightarrow \varepsilon$ to R_1

Chomsky Normal Form

4) For each production of the form

$A \rightarrow X_1 \ldots X_k$ in R, k > 2, add to $R_1$the following productions:

$A \rightarrow Y_1 \langle X_2 \ldots X_k \rangle$

$\langle X_2 \ldots X_k \rangle \rightarrow Y_2 \langle X_3 \ldots X_k \rangle$

…

$\langle X_{k-1} X_k \rangle \rightarrow Y_{k-1} Y_k$

where Y_j stands for X_i if X_i is a nonterminal otherwise it is a new nonterminal added to V_1

5) For each production of the form $A \rightarrow X_1 X_2$ where either X_1 or X_2 or both are terminals add to R_1 the production $A \rightarrow Y_1 Y_2$

6) For each terminal a replaced by a Y, add the production $Y \rightarrow a$

Chomsky Normal Form

Example:

$S \rightarrow bA \mid aB$

$A \rightarrow bAA \mid aS \mid a$

$B \rightarrow aBB \mid bB \mid b$

Greibach Normal Form

Recursive, left-recursive, right-recursive:

A *nonterminal* A in a CFG G = (V,∑,R,S) is said to be *recursive* if $A \Rightarrow^{+} \alpha A \beta$ for some β and α. If $\alpha=\varepsilon$, then A is left-recursive, if $\beta=\varepsilon$, then A is right-recursive.

A *grammar* with at least one left-(right-) recursive nonterminal is said to be *left-(right-) recursive*

Greibach Normal Form

Lemma: Let $G = (V,\sum,R,S)$ be a CFG in which $A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$ are all the A-productions in P and no β begins with A. Let $G_1 = (V \cup \{B\}, \sum, R_1, S)$ where R_1 is R with the above productions are replaced by

$A \rightarrow \beta_1 \mid \ldots \mid \beta_n \mid \beta_1 B \mid \ldots \mid \beta_n B$

$B \rightarrow \alpha_1 \mid \ldots \mid \alpha_m \mid \alpha_1 B \mid \ldots \mid \alpha_m B$

Then $L(G) = L(G_1)$

Greibach Normal Form

Algorithm – elimination of left recursion

Input: A proper CFG $G=(V,\sum,R,S)$

Output: A CFG G_1 with no left-recursion

Method: Let $V-\sum = \{A_1, ..., A_n\}$. Transform G so that if $A_i \rightarrow \alpha$ is a production, then α begins with either a terminal or a symbol A_j such that $j > i$.

this can be accomplished by substitution for lower numbered productions and eliminating immediate left-recursion.

Greibach Normal Form

Example:

$A \rightarrow BC \mid a$

$B \rightarrow CA \mid Ab$

$C \rightarrow AB \mid CC \mid a$

Greibach Normal Form

Definition: A CFG G = (V,∑,R,S) is said to be in Greibauch normal form (GNF) if G is ε-free and each non ε-production is of the form $A \rightarrow a\alpha$ where a is a terminal and α is a string in $(V-\sum)^*$.

Greibach Normal Form

Lemma: Let G=(V,∑,R,S) be a non-left recursive grammar. Then there is linear order < on V-∑ such that $A \rightarrow B\alpha$ is in R, the A < B.

Proof: Let R be a relation A R B if and only if $A \Rightarrow^* B\alpha$ for some α. R is a partial order

Extend R to a linear order

Greibach Normal Form

Algorithm: Conversion to GNF

Input: G=(V,∑,R,S), G non-left-recursive, proper CFG

Output: A CFG G_1 in GNF

Method: 1) Construct a linear order on V-∑

2) Set i= n-1

3) If i=0 go to step 5. Otherwise replace each production of the form $A_i \rightarrow A_j\alpha$, j>i, by
$A_i \rightarrow \beta_1\alpha \mid \ldots \mid \beta_m\alpha$ where $A_j \rightarrow \beta_1 \mid \ldots \mid \beta_m$ are all the A_j-productions (β_i's begin with terminal)

4) Set i=i-1; go to step 2

5) For each production $A \rightarrow aX_1 \ldots X_k$, if X_j is a terminal replace it by a nonterminal Y_j and add a production $Y_j \rightarrow X_j$

Greibach Normal Form

Example:

$E \rightarrow T \mid TE_1$

$E_1 \rightarrow +T \mid +TE_1$

$T \rightarrow F \mid FT_1$

$T_1 \rightarrow *F \mid *FT_1$

$F \rightarrow (E) \mid a$