Transformation of context-free grammar

Sometimes it is useful to transform a context-free grammar to follow specific form, to facilitate the syntax analyzer (Parser), from one side and to facilitate the algorithms design and the implementation process from other side, thus our CFG should be cleaned in the previously stage, by eliminate ,The useless symbols, $\varepsilon - rules, A \longrightarrow B\ (rules)$, left recursion, then take specific form such as : the Chomsky Normal Form, and Greibach Normal Form

1-Elimination of useless symbols

Definition: we say that the symbol $X \in N \cup T$ is useless symbol in CFG G If $\nexists$ derivation of the form $S \Longrightarrow^* wXy \Longrightarrow^* wxy\ , x, y \in T^*$

These symbols may be:

- **Non-generating symbols**
- **Unreachable symbols**

1-1-Non-Generating Symbols

These symbols don't produce terminal strings.

$$\{w\ where\ A \overset{*}{\Rightarrow} w, w \in T^*\} = \emptyset?$$

Method:

1-Any terminal symbol is considered as generating symbol.

2-Any non-terminal symbol has rules contains only generating symbols, is considered as generating symbol.

3-Repeate 2 until no new.

4-Remove each non- generating symbol with its rules

Example

S → AB | a

A → aA | a

B → bB

C → c

-{a,b,c} are generating symbols

- A → a, then A is generating symbol

-C → c, then C is generating symbol

- S → a, then S is generating symbol

- B → bB, loop forever, then B is no generating symbol

Now we remove B and all grammatical rules that contain B.

We obtain:

S → A | a

A → aA | a

C → c

1-2-Unreachable Symbols

All symbols can't be reached from the axiom S.

Idea:

- Begin by S
- Any symbol appears in rules

  S → reachable.

- Repeat
- Delete any unreachable symbol.

Example

We continue with the above grammar:

S → A | a

A → aA | a

C → c

We observe that the symbol C cant reached by any path from S.

Then the grammar became:

S → A | a

A → aA | a

We may describe formally by:

Algorithm no-generating symbols elimination

Input : a CFG G=(N,T,S,P)

Output: CFG Grammar G' without non-generating symbols and

L(G)=L(G')

Create sets recursively: N0,N1,…,

1-$N_0 = \emptyset$,i=1

2-$N_i = \{A/A \to \alpha \in P \wedge \alpha \in (N_{i-1} \cup T)^*\} \cup N_{i-1}$

3-if $N_i \neq N_{i-1}$ then i=i+1;goto2;

Else

$N_g = N_i$

Remove:

All non terminals $\notin N_g$

All rules containing it.

Algorithm Unreachable symbols elimination

Input : a CFG G=(N,T,S,P)

Output: CFG Grammar G' without unreachable symbols and L(G)=L(G')

Create sets recursively: N0,N1,…,

1-$V_0 = \{S\}$,i=1

2-$V_i = \{X / \exists A \to \alpha X \beta \in P \wedge A \in V_{i-1}\} \cup V_{i-1}$

3-if $V_i \neq V_{i-1}$ then i=i+1;goto2;

Else

$N' = V_i \cap N$ and $T' = V_i \cap T$

P' contains all rules use symbols in Vi

Algorithm Useless symbols from CFG elimination

Input : a CFG G=(N,T,S,P)

Output: CFG Grammar G' without useless symbols and L(G)=L(G')

No-generating symbols elimination;

Unreachable symbols elimination;

2-Elimination of $\varepsilon - rules\ (\varepsilon - free\ grammar)$

Algorithm epsilon-free

Input: CFG G=(N,T,P,S)

Output: CFG G1=(N1,T,P1,S1) $(\varepsilon - free)$

Method

We construct

1-$N_e = \{A, A \in N, A \Longrightarrow^{+} \varepsilon\}$

2-Let the grammatical rules :

$$A \longrightarrow \alpha_0\beta_1\alpha_1\beta_2 \ldots \in P \text{ and } \beta_i \in N_e$$

Add to p1 all rules

$A \longrightarrow \alpha_0 X_1 \alpha_1 X_2 \ldots X_i \in \{\beta_i, \varepsilon\}$ Without adding $A \longrightarrow \varepsilon$

3-If $S \in N_e$ then add the two rules $S1 \longrightarrow S|\varepsilon$

Where S1 is a new non terminal symbol $N1 = N \cup \{S1\}$

Else N1=N,S1=S

4- G1=(N1,T,P1,S1)

EXAMPLE

Let G=({S},{a,b},P,S)

$$P = (S \rightarrow aSbS|bSaS|\varepsilon)$$

Give the equivalent $\varepsilon - free\ grammar$

1-$N_e = \{S\}$

2-

$$A \longrightarrow \alpha_0\beta_1\alpha_1\beta_2 \ldots \in P \; and \; \beta_i \in N_e$$

$$S \rightarrow aSbS$$

Add

$$S \rightarrow aSbS|abS|aSb|ab$$

The same thing for :

$$S \rightarrow bSaS$$

Add

$$S \rightarrow bSaS|baS|bSa|ba$$

3- $S \in N_e$ THEN N1={S1,S}

$$P1 = (S1 \rightarrow S|\varepsilon; S \rightarrow aSbS|abS|aSb|ab|bSaS|baS|bSa|ba)$$

3-Elimination of unit production ($A \longrightarrow B \; (rules)$)

Algorithm Elim;

Input: CFG G=(N,T,P,S)

Output: CFG G1=(N,T,P1,S) $(without \; A \rightarrow B)$

Method

1-We construct for each $A \in N, N_A = \{B \; where \; A \Rightarrow^* B\}$ as follow:

$$a - N_0 = \{A\}, b - i = 1, N_i = \{C, B \longrightarrow C \in P; and \; B \in N_{i-1}\} \cup N_{i-1}$$

$$c - if \; N_i \neq N_{i-1}, then \; i = i + 1, goto \; (b)$$

Else $N_A = N_i$

2-we construct P1 as :

If $B \longrightarrow \alpha \in P, and\ is\ not\ of\ A \longrightarrow B$ Then add
$A \longrightarrow \alpha\ to\ P1\ for\ each\ A\ such\ as\ B \in N_A$

3-G1=(N,T,P1,S)

EXAMPLE

Apply the algorithm on the following grammar:

$$P = (E \longrightarrow E + T|T, T \longrightarrow T * F|F, F \longrightarrow (E)|a)$$

We construct N_E, N_T, N_F

$$N_0 = \{E\}, N_1 = \{E, T\}, N_2 = \{E, T, F\}, N_3 = N_2$$

$$N_E = \{E, T, F\}, N_T = \{T, F\}, N_F = \{F\}$$

$$P1 = (E \longrightarrow E + T|T * F|(E)|a, T \longrightarrow T * F|(E)|a, F \longrightarrow (E)|a)$$

Definition

Each CFG is a cycle- free iff do not contains derivation of the form:

$$A \Longrightarrow^{+} A$$

Definition

Each CFG cycle-free,$\varepsilon - free$, and not contains useless symbols is a monotone grammar.

4- Chomsky Normal Form (CNF):

Definition: *A context-free grammar is in Chomsky normal form if each rule has the form:* $A \to BC; A \to a$

$a \in T, and\ A \neq \varepsilon, B \neq \varepsilon$

And with the possibility of $S \to \varepsilon$.

Algorithm CNF;

Input: monotone CFG grammar not contains rules of the form of $A \longrightarrow B$

Output: CFG in CNF.

We construct P1 as follow:

1-Add each rule of the form $A \longrightarrow a$ from P to P1

2-Add each rule of the form $A \longrightarrow BC$ from P to P1.

3-If $S \longrightarrow \varepsilon$ THEN add it to P1

4-for each rule $A \longrightarrow X_1 X_2 \ldots X_k, k > 2$ add toP1 the following rules:

$$A \longrightarrow X'_1 < X_2 \ldots X_k >$$

$$< X_2 \ldots X_k > \longrightarrow X'_2 < X_3 \ldots X_k >$$

…..

$$< X_{k-1} X_k > \longrightarrow X'_{k-1} X'_k$$

$X'_i replaced\ by\ X_i, if\ X_i \in N, and\ is\ a\ new\ non\ tyer, inal\ if\ X_i \in T$

5-For each rule of the form:$A \longrightarrow X_1 X_2, X_1\ or\ X_2 \in T$ ADD TO P1

$A \longrightarrow X_1{}'X_2{}'$

6-for each
non terminal A'*produced by step* 4,5 *add to* $P1$ *the rule* $A' \rightarrow a$

Example

Let G=(N,T,S,P) be a monotone grammar

$$P = (S \longrightarrow aAB|BA; A \longrightarrow BBB|a; B \longrightarrow AS|b)$$

$$P1 = (S \longrightarrow BA; A \longrightarrow a; B \longrightarrow AS; B \longrightarrow b)$$

$S \longrightarrow aAB$ k=3 REPLACED BY

$$S \longrightarrow X_1' < AB >$$

$$< AB > \longrightarrow AB$$

$$X_1' \longrightarrow a$$

$A \longrightarrow BBB$ REPLACED BY $A \longrightarrow B < BB >; < BB > \longrightarrow BB$

N1={S;A,B,X'1,<AB>,<BB>}.

5-Greibach Normal Form (GNF)

Definition

G is a CFG, G is in GNF if all the rules are of the form :

$$A \longrightarrow a\alpha, a \in T, \alpha \in N^*$$

With the possibility of $S \longrightarrow \varepsilon, if\ \varepsilon \in L(G)$

The idea is left recursion elimination.

Theorem

For each CFG there exists an equivalent GNF.

Definition

We say that the non terminal symbol A in a CFG G=(N,T,S,P) is recursive

If $\Longrightarrow^{+} \alpha A\beta, IF\ \alpha = \varepsilon\ THEN\ A\ is\ left\ recursive$,
$IF\beta = \varepsilon\ THEN\ A\ is\ right\ recursive$.

We say that a CFG is recursive if it contains non terminal recursive symbols.

Remark:

The majority of algorithms of the compilers cannot be used if the grammar is left recursive.

Lemma:

Let G=(N,T,S,P) be a CFG , such that $A \longrightarrow A\alpha_1|A\alpha_2 \ldots A\alpha_m|\beta_1|\beta_2| \ldots |\beta_n \in P, such\ that\ \beta_i\ does\ not\ begin\ by\ A, i = 1, n.$

Let $G1 = (N \cup \{A1\}, T, S, P1)$

Such as P1 is P with replacing the above rules by :

$$A \rightarrow \beta_1|\beta_2| \ldots |\beta_n|\beta_1 A1|\beta_2 A1| \ldots \beta_n A1$$

$$A1 \rightarrow \alpha_1|\alpha_2| \ldots \alpha_m|\alpha_1 A1|\alpha_2 A1| \ldots \alpha_m A1$$

A1 is a new non terminal symbol THEN L(G)=L(G1)

Example:

$$P = (E \rightarrow E + T|T; T \rightarrow T * F|F; F \rightarrow (E)|a)$$

$$E \rightarrow T|TE1; E1 \rightarrow +T| + TE1$$

$$T \rightarrow F|FT1; T1 \rightarrow * F| * FT1$$

Algorithm LRE (*Left Recursion Elimination*)

Input :monotone CFG grammar G

Output: CFG grammar G1 without left recursion.

Method.

Let N ={A1,A2,…,Am}

We transform in the beginning G such that if
$Ai \rightarrow \alpha \in P, THEN\ \alpha\ begin\ by\ terminal\ symbol$

$$OR\ by\ non\ terminal\ symbol\ Aj,$$

$$j > i$$

For this

1-put i=1

2-let the rules $i \longrightarrow Ai\alpha_1|Ai\alpha_2 \ldots Ai\alpha_m|\beta_1|\beta_2| \ldots |\beta_p$, such as
$\beta_i\ not\ begin\ by\ Ak, k \leq i$

Replace these rules by :

$Ai \rightarrow \beta_1|\beta_2| \ldots |\beta_p|\beta_1 Ai'|\beta_2 Ai'| \ldots \beta_p Ai'$
$Ai' \rightarrow \alpha_1|\alpha_2| \ldots \alpha_m|\alpha_1 Ai'|\alpha_2 Ai'| \ldots \alpha_m Ai'$

Ai' is a new no terminal symbol.

Now nall the rule begin either by terminal or by no terminal Ak such as k>i.

3-if i=m then G1 is reached STOP.

Else i=i+1, j=1

4-replace each rule of the form $Ai \rightarrow Aj\alpha\ by\ Ai \rightarrow \beta_1\alpha|\beta_2\alpha| \ldots \beta_m\alpha$

Such as $Aj \rightarrow \beta_1|\beta_2| \ldots |\beta_m$

Now all rule begin either by terminal or by no terminal Ak; k>j

5-if j=i-1 goto (2) else j=j+1 goto (4).

Example

$$P = (A \rightarrow BC|a; B \rightarrow CA|Ab; C \rightarrow AB|CC|a)$$

1-A1=A,A2=B,A3=C i=1

2-no change

3-i=2;j=1

4-the rule $B \rightarrow Ab, by\ B \rightarrow BCb|ab$

5- GOTO 2 i=2;j=1

$$B \rightarrow BCb|ab|CA \quad replced\ by \quad B \rightarrow ab|CA|abB'|CAB';\ \ B' \rightarrow Cb|CbB'$$

3-i=3,j=1

4-$C \rightarrow AB\ Replaced\ by\ \ C \rightarrow BCB|aB$

4-I=3,J=2 $C \rightarrow BCB|aB|CC|a$ $replaced\ \ by : C \rightarrow abCB|CACB|abB'CB|CAB'CB|CC|a|aB$

2-i=3

$$C \rightarrow abCB|abB'CB|aB|a|abCBC'|abB'CBB'|aBC'|aC'$$

$$C' \rightarrow ACBC'|AB'CBC'|CC'|ACB|AB'CB|C$$

Rmark:

G is CGF without left recursion then exist linear order < such that

If $A \rightarrow B\alpha$ then A<B.

Algorithm GNF

Input : monotone grammar without left recursion

Input : CFG G1 of the from of Greibach such that L(G)=L(G1)

Method:

Order the symbolof N={A1,A2,…,Am} such that A1<A2<…<Am

2-put i=m-1

3-if i=0 goto (5) else replace each rule of the form $Ai \rightarrow Aj\alpha \;\; i < j \; by$:

$Ai \rightarrow \beta_1\alpha|\beta_2\alpha| \ldots |\beta_n\alpha$ where
$Aj \rightarrow \beta_1|\beta_2| \ldots |\beta_n \; and \;\; \beta_i \; begin \; by \; terminal \; symbol.$

4-i=i-1 goto (3)

5-in this stage all the rules except $S \rightarrow \varepsilon$, begin by terminal symbol.

For each rule $A \rightarrow aX1X2 \ldots Xk$ replace all terminal symbol Xj by Xj'(new non terminal).

6-for all non terminals Xj' added in step 5 add $Xj' \rightarrow Xj$.

Remark: see the Tutorial series for application