Simplificación de una gramática independiente del contexto

El objetivo de esta práctica es diseñar un programa en C que sirva para simplificar una gramática independiente del contexto. Para este cometido se puede optar por diseñar un único programa o bien diferentes módulos que realicen las siguientes funciones:

1. Eliminación de producciones vacías
2. Eliminación de producciones unitarias
3. Eliminación de símbolos y variables inútiles
4. Conversión en forma normal de Chomsky

Normalmente para simplificar al máximo una gramática ejecutaremos sobre la gramática los pasos 1–2–3. Se mejora la eficiencia del proceso si se ejecutan los pasos: 3–1–2–3.

El paso 4 (conversión en forma normal de Chomsky) no produce una simplificación propiamente dicha, sino la escritura de la gramática en una forma normalizada.

Tanto si se implementa un único programa ejecutable como si se eligen diferentes módulos, la entrada del programa será un fichero que represente una gramática independiente del contexto representada del modo que se ha utilizado en prácticas anteriores, y la salida del programa será otro fichero correspondiente a la gramática transformada. Los ficheros que representan gramáticas admitirán líneas de comentarios (que comienzan con el símbolo #) que han de ser ignorados al leer la gramática.

A continuación se describen a través de pseudocódigo los diferentes algoritmos que se han de implementar.

Eliminación de símbolos y variables inútiles en una gramática independiente del contexto.

Dada una gramática independiente del contexto (de ahora en adelante, una gramática), $G \equiv (V, \Sigma, S, P)$, un símbolo $X \in (V \cup \Sigma)$ se dice que es útil si y sólo si existe una derivación de la forma:

$S \Rightarrow^{+} \alpha X \beta \Rightarrow^{+} \omega \in \Sigma^{*}$

es decir, se han de dar dos condiciones:

a) que a partir del símbolo X se pueda derivar una cadena de símbolos exclusivamente terminales.

b) que el símbolo X aparezca en una forma sentencial.

Una regla de producción de una gramática se dice útil si lo son todos sus símbolos.

A continuación se describe un algoritmo para eliminar símbolos y variables inútiles en una gramática.

El algoritmo se divide en dos etapas: la primera de ellas elimina los símbolos que no cumplen (a) y la segunda elimina los símbolos que no cumplen (b):

<u>Etapa 1</u>

V' = $\emptyset$

Para cada producción de la forma A $\rightarrow$ w **hacer**

  V' = V' $\cup$ {A}

  while (V' cambie) **do**

  Para cada producción de la forma B $\rightarrow$ a **hacer**

    if (todas las variables de a pertenecen a V')

      V' = V' $\cup$ {B}

- Eliminar todas las variables ($C \in V$) que estén en V y no en V'
- Eliminar todas las producciones donde aparezca una variable de las eliminadas en el paso anterior

<u>Etapa 2</u>

V" y J son conjuntos de variables (de V)
J son las variables pendientes de analizar
T' es un conjunto de símbolos terminales (de S)
J = {S}
V" = {S}
T' = ∅
while (J ≠ ∅) **do**
    Extraer un no terminal A de J: J = J – {A}
    Para cada producción de la forma A → a
        begin
        Para cada no–terminal B en a
            if (B ∉ V") then
                begin
                J = J ∪ {B}
                V" = V" ∪ {B}
                end
        Poner todos los terminales de a en T'
        end

- Eliminar todas las variables que no estén en V" y todos los terminales que no estén en T'.
- Eliminar todas las producciones donde aparezca un símbolo o variable de los eliminados.

Eliminación de producciones vacías

Un símbolo $A \in V$ se dice que es anulable si existe una derivación que haga $S \Rightarrow^* \varepsilon$ es decir, si el símbolo puede derivar en la cadena vacía. A continuación se presenta un pseudocódigo para un algoritmo que calcula todas las variables anulables en una gramática:

Cálculo del conjunto de variables anulables, H:

H = ∅
Para cada producción de la forma $A \to \varepsilon$
    H = H ∪ {A}
while (H cambie) **do**
    Para cada producción $B \to A_1A_2 \ldots A_n$ donde $A_i \in H\ \forall\ i = 1, 2, \ldots n$ **hacer**
        H = H ∪ {B}

El siguiente pseudocódigo corresponde a un algoritmo que elimina de la gramática todas las producciones vacías (producciones de la forma $A \to \varepsilon$):

Eliminación de producciones vacías:

Eliminar todas las producciones vacías ($A \to \varepsilon$)
Para cada producción de la forma $A \to \alpha_1\alpha_2 \ldots \alpha_n$ ($\alpha_i \in (\Sigma \cup V)$)
    Eliminar la producción $A \to \alpha_1\alpha_2 \ldots \alpha_n$
    Añadir todas las producciones $A \to \beta_1\beta_2 \ldots \beta_n$
    donde: $\beta_i = \alpha_i$ si α_i no es anulable
        $(\beta_i = \alpha_i) \vee (\beta_i = \varepsilon)$ si α_i es anulable
        β_i no es $\varepsilon\ \forall$ i (no se introducen producciones nulas)

Eliminación de producciones unitarias

Se denominan producciones unitarias a las que tienen la forma $A \rightarrow B$ con $A, B \in V$. Este tipo de producciones hacen que la gramática sea innecesariamente compleja. A continuación presentamos el pseudocódigo de un algoritmo que elimina este tipo de producciones:

H será el conjunto de parejas (A, B) tales que $A \Rightarrow B$
$H = \emptyset$
Para cada producción de la forma $A \rightarrow B$
    $H = H \cup \{(A, B)\}$
while (H cambie) **do**
    Para cada par de parejas de la forma (A, B) (B, C)
        if ($(A, C) \notin H$) then
            $H = H \cup \{(A, C)\}$
Eliminar todas las producciones unitarias
Para cada producción $B \rightarrow \alpha$
    Para cada pareja $(A, B) \in H$
        Añadir a la gramática una producción $A \rightarrow \alpha$

Conversión en forma normal de Chomsky

Una gramática se dice que está en forma normal de Chomsky si y sólo si todas sus producciones tienen una de las siguientes formas:

1. $A \rightarrow BC$
2. $A \rightarrow a$
3. $S \rightarrow \varepsilon$

El siguiente pseudocódigo corresponde a un algoritmo para convertir una gramática en forma normal de Chomsky:

Para cada producción de la forma $A \rightarrow \alpha_1\alpha_2 \ldots \alpha_n$ ($\alpha_i \in (\Sigma \cup V)$, $n \geq 2$)
    Para cada α_i si $\alpha_i = a \in \Sigma$
        Se añade la producción $C_a \rightarrow a$
        Se cambia α_i por C_a en la producción $A \rightarrow \alpha_1\alpha_2 \ldots \alpha_n$
Para cada producción de la forma $A \rightarrow B_1B_2 \ldots B_m$ ($m \geq 3$)
    Añadir (m−2) variables $D_1, D_2, \ldots, D_{m-2}$ (distintas para cada producción)
    Reemplazar la producción $A \rightarrow B_1B_2 \ldots B_m$ por las producciones
        $\mathbf{A \rightarrow B_1D_1}$, $\mathbf{D_1 \rightarrow B_2D_2}$, ..., $\mathbf{D_{m-2} \rightarrow B_{m-1}B_m}$