Simplificación y normalización de gramáticas incontextuales

Para esta práctica, se ha implementado una clase en C++ llamada *GIC*, para soportar en este lenguaje de programación la representación de gramáticas incontextuales, que se definen con la tupla *G=(N,T,P,S)*, donde N es el conjunto de los símbolos no terminales, T es el conjunto de símbolos terminales, P las producciones y S el axioma.

Todas las estructuras usadas para el soporte de conjuntos, vectores, listas están proporcionadas por las librerías de la STL (*Standar Template Library*).

Una vez tenemos el soporte para estas gramáticas, hay que implementar los algoritmos necesarios para que, tomando una gramática G1 libre de contexto, obtengamos una gramática G2 que cumpla las siguientes propiedades:

1. G2 es casi equivalente a G1, es decir, *L(G2) = L(G1) – {λ}*
2. G2 está simplificada
3. G2 está en Forma Normal de Chomsky.

Se ha dividido el trabajo en dos partes principales:

- A partir de una gramática libre de contexto, obtener una **gramática casi equivalente** y **simplificada**.
- A partir de una gramática simplificada obtener la **gramática en Forma Normal de Chomsky.**

La clase GIC

La clase *GIC* encapsula todos los datos y métodos relacionados con los autómatas finitos deterministas. Como ya se ha dicho anteriormente, las gramáticas incontextuales se definen con la tupla *G=(N,T,P,S)*.

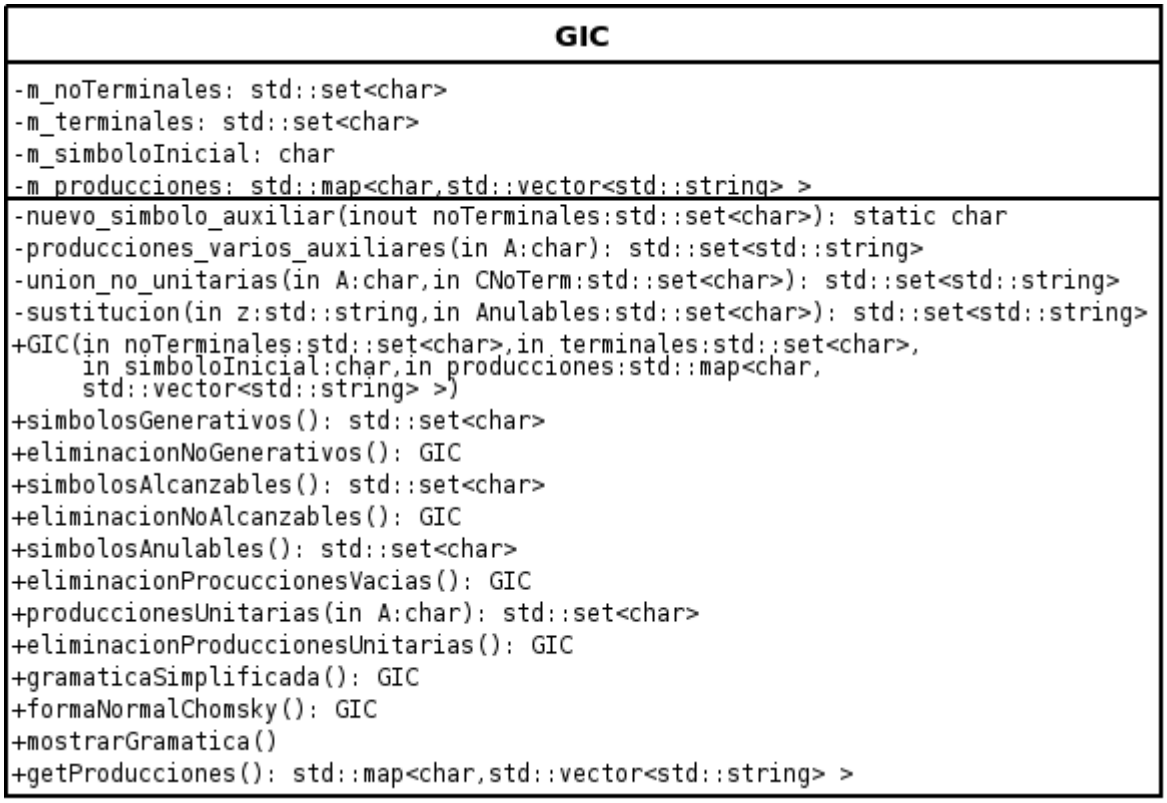
Vamos a ver como se ha definido dentro de la estructura de clase GIC cada uno de los elementos mencionados anteriormente:

- ***N***: representa el conjunto de símbolos no terminales, y se ha definido de forma que se corresponda con un conjunto de caracteres sin repeticiones.
- ***T***: este representa el conjunto de símbolos terminales de la gramática. Se ha representado de la misma forma que *N*, como un conjunto de caracteres que no admite repeticiones.
- ***P***: este conjunto representa las transiciones de la gramática incontextual. Se representa mediante un mapa, que asocia un carácter $A \in N$ un vector de cadenas de caracteres, siendo x una de estas cadenas tal que $x \in (N \cup T)^*$.
- ***S****:* el símbolo inicial de la gramática se almacena simplemente como un carácter.

Además de definir estos elementos dentro de la clase, se ha implementado un método llamado ***mostrarGramatica***, que nos ayuda a mostrar de forma legible la gramática contenida en la clase por la salida estándar.

También se han implementado todos los métodos necesarios para la simplificación de gramáticas y para la obtención de la Forma Normal de Chomsky, que serán comentados con mayor profundidad en sus apartados correspondientes.

Teniendo en cuenta esta descripción de la clase, y los métodos que se explicarán en las secciones posteriores, el diagrama UML asociado a la clase GIC sería el siguiente:



Obtención de la gramática simplificada

Para la obtención de la gramática simplificada, se implementarán nuevos métodos dentro de la clase GIC que se corresponderán con los algoritmos explicados en la práctica 5 de la asignatura de Teoría de Autómatas y Lenguajes.

En primer lugar, se ha implementado el método ***simbolosGenerativos***, que nos devuelve el conjunto de símbolos generativos, es decir, los símbolos que son capaces de generar en uno o más pasos de derivación cadenas formadas sólo por símbolos terminales.

El algoritmo implementado en este caso es el siguiente:

Entrada: *G=(N,T,P,S)*, la gramática que llama al método
Salida: *gen*, una lista de los símbolos generativos de la gramática
Método:

gen = { }
gen2 = *T*
Repetir
gen = { $A \in N : A \rightarrow x \in P, x \in gen2^*$}
gen = *gen* - *gen2*
gen2 = *gen* ∪ *gen2*
hasta *gen* = { }

gen = gen2 - T
finMétodo

Una vez implementado este método, el siguiente paso es la creación del método ***eliminacionNoGenerativos***, que nos devolverá una gramática incontextual equivalente que no tiene ningún símbolo que no sea generativo.

El algoritmo usado en la implementación es el siguiente:

Entrada: *G=(N,T,P,S),* gramática que llama al método.
Salida: *G1=(N1,T,P1,S)* sin símbolos no generativos, excepto posiblemente S.
Método:
gen = simbolosGenerativos()
N1 = gen ∪ { *S* }
P1 = { $A \rightarrow x \in P : A \in gen, x \in (gen \cup T)^*$}
finMétodo

Lo siguiente que interesa para la simplificación de gramáticas, es eliminar los símbolos que no sean alcanzables, es decir, aquellos que aparecen en cualquier derivación que se produce a partir del axioma. Para esto, primero implementaremos un método ***simbolosAlcanzables***, que nos devolverá una lista de los alcanzables.

El algoritmo implementado es el siguiente:

Entrada: *G=(N,T,P,S),* gramática que llama al método.
Salida: *alc*, una lista con los símbolos auxiliares y terminales alcanzables de la gramática
Método:
alc = {*S*}
aux1 = {*S*}
Repetir
aux2 = { $a \in N \cup T : \exists A \in aux1, \exists \alpha,\beta \in (N \cup T)^* : A \rightarrow \alpha\ a\ \beta \in P$}
aux1 = aux2 - (*alc* ∪ *T*)
alc = alc ∪ *aux2*
hasta *aux1* = { }
finMétodo

Teniendo ya este método, el siguiente será el encargado de devolver una gramática equivalente que no posea ningún símbolo no alcanzable. El método se llama ***eliminacionNoAlcanzables***, e implementa el siguiente algoritmo:

Entrada: *G=(N,T,P,S),* gramática que llama al método.
Salida: *G1=(N1,T1,P1,S),* sin símbolos no alcanzables.
Método:
alc = simbolosAlcanzables()
N1 = alc ∩ N
T1 = alc ∩ T
P1 = { $A \rightarrow \alpha \in P : A \in N1$ }
finMétodo

Después de implementar la eliminación de símbolos no alcanzables, pasamos a la eliminación de las producciones vacías ($A \rightarrow \lambda$). Para la eliminación de las producciones vacías se debe calcular en primer lugar el conjunto de símbolos anulables, es decir,

símbolos que, en uno o más pasos de derivación, son capaces de generar la cadena vacía. Para esto, se ha implementado el método ***simbolosAnulables***, que devuelve el conjunto de símbolos que son anulables.

El siguiente algoritmo es el que sigue el método comentado anteriormente:

Entrada: *G=(N,T,P,S),* gramática que llama al método.
Salida: *Anulables*, una lista con los símbolos anulables de la gramática.
Método:
    Anulables = { }
    Repetir
        aux = { $A \in N - Anulables : \exists\ x \in Anulables^*, (A \rightarrow x) \in P$ }
        Anulables = *Anulables* ∪ *aux*
    hasta *aux* = { }
finMétodo

Para poder aplicar la eliminación de las producciones vacías, previamente se ha definido un método privado ***sustitucion***, que implementa la siguiente función:

Sea la sustitución **g: (N ∪ T)*** $\rightarrow$ $\mathbf{2^{(N \cup T)^*}}$ definida como sigue:

- Si $x \in (N \cup T) - Anulables$, $g(x) = \{x\}$
- Si $x \in Anulables$, $g(x) = \{ x, \lambda \}$

Por ejemplo si z=AaAB y únicamente A es anulable, entonces:

$g(z) = g(A)\ g(a)\ g(A)\ g(B) = \{A, \lambda\}\ \{ a \}\ \{ A, \lambda\}\ \{ B \} = \{ AaAB, AaB, aAB, aB \}$

Definimos para toda cadena $z \in (N \cup T)^*$, $f(z) = g(z) - \{ \lambda \}$

Para una implementación más cómoda del método, se ha definido una función externa a la clase, llamada ***concatena***, que dados dos conjuntos de cadenas, devuelve el conjunto de cadenas fruto de la concatenación de las cadenas de cada conjunto.

El siguiente paso ha sido desarrollar el método ***eliminacionProcuccionesVacias*** que devuelve una gramática casi equivalente a la inicial, y que no contiene símbolos anulables. El algoritmo que sigue es el siguiente:

Entrada: *G=(N,T,P,S),* gramática que llama al método.
Salida: *G1=(N,T,P1,S)* sin producciones vacías, con $L(G1) = L(G) - \{ \lambda \}$.
Método:
    Anulables = simbolosAnulables()
    P1 = { $A \rightarrow x : \exists\ (A \rightarrow z) \in P, x \in$ sustitucion(z, Anulables) }
finMétodo

Los siguientes métodos a implementar son los necesarios para la eliminación de las producciones unitarias. En primer lugar definimos el método ***produccionesUnitarias***, que recibe como argumento un símbolo no terminal, y devuelve una lista de símbolos alcanzables a partir de este no terminal mediante producciones unitarias.

El algoritmo implementado es el siguiente:

Entrada: *G=(N,T,P,S),* gramática que llama al método y el símbolo A ∈ N
Salida: C(A), una lista con los símbolos alcanzables a partir de A mediante producciones unitarias
Método:

C(A) = {*A*}
aux1 = {*A*}
Repetir
aux2 = $\cup_{B \in aux1}$ { $C \in N : (B \rightarrow C) \in P$ }
aux1 = *aux2* - *C(A)*
C(A) = *C(A)* ∪ *aux1*
hasta *aux1* = { }

finMétodo

Antes de pasar a la eliminación de las producciones unitarias, necesitamos definir una serie de métodos auxiliares que nos facilitarán la implementación del último algoritmo. En primer lugar implementamos ***producciones_varios_auxiliares***, que implementa el siguiente algoritmo:

Entrada: *G=(N,T,P,S),* gramática que llama al método y el símbolo A ∈ N
Salida: *pva*, Partes derechas de producciones.
Método:

pva = { $x : A \rightarrow x \in P$, *x no es un único símbolo auxiliar* }

finMétodo

También implementamos el método auxiliar ***union_no_unitarias***, que implementa el siguiente algoritmo:

Entrada: *G=(N,T,P,S),* gramática que llama al método y el símbolo A ∈ N, y C(A)
Salida: *unu*, Unión de partes derechas de producciones.
Método:

unu = { $\cup_{B \in C(A)}$ *producciones_varios_auxiliares(B)* }

finMétodo

Una vez hemos definido todos los métodos auxiliares, el siguiente paso es definir el método ***eliminacionProduccionesUnitarias***, que implementa el siguiente algoritmo:
Entrada: *G=(N,T,P,S),* gramática que llama al método.
Salida: *G1=(N,T,P1,S)* sin producciones unitarias, con *L(G1) = L(G).*
Método:

P1 = { $A \rightarrow x : x \in$ *union_no_unitarias(A, produccionesUnitarias(A))* }

finMétodo

Ahora tenemos todos los métodos necesarios para implementar el método ***gramaticaSimplificada***, que devuelve una gramática casi equivalente a la que llama al método y simplificada.

El algoritmo implementado es el siguiente:

Entrada: *G=(N,T,P,S),* gramática que llama al método.
Salida: *G4=(N',T,P',S)* sin producciones unitarias, con *L(G4) = L(G) – { λ }.*
Método:

G1 = eliminacionProcuccionesVacias()
G2 = G1.eliminacionProduccionesUnitarias()

G3 = G2.eliminacionNoGenerativos()
G4 = G3.eliminacionNoAlcanzables()
finMétodo

Forma Normal de Chomsky

En la siguiente parte de la práctica tememos que implementar los mecanismos necesarios para, dada una gramática simplificada (mediante el método anteriormente descrito), obtener una gramática que este en Forma Normal de Chomsky.

Diremos que una gramática incontextual *G=(N,T,P,S)* que no genera la cadena vacía, está en Forma Normal de Chomsky cuando todas sus reglas son de la forma:

- $A \rightarrow BC$ *con* $A,B,C \in N$
- $A \rightarrow a$, *con* $A,B \in N$ *y* $a \in T$

Para realizar esta conversión se ha implementado el método ***formaNormalChomsky***, que se corresponde con el siguiente algoritmo para la obtención de la Forma Normal de Chomsky:

Entrada: *G=(N,T,P,S),* gramática que llama al método, previamente simplificada.
Salida: *G=(N'',T,P'',S)*.
Método:

/* PASO 1 */
N'=N; P'=∅;
Para toda regla ($A \rightarrow \alpha$) de P **hacer**
Si $|\alpha|=1$ **entonces**
añadir la regla a P' /* Ya esta en FNC */
Sino
Sea $\alpha=X_1, X_2...X_m$ con m > 1
Para i=1 hasta m hacer
Si $X_i=a \in \Sigma$ **entonces**
Se añade a N' un nuevo no terminal C_a y se añade a P' una nueva regla ($C_a \rightarrow a$).
finSi
finPara
Se añade a P' una regla ($A \rightarrow X'_1, X'_2...X'_m$) con:
$X'_i=Xi$ si $X_i \in N$
$X'_i=Ca$ si $X_i = a \in \Sigma$
finSi
finPara

/*Al finalizar el PASO 1 todas las reglas de la gramática resultante G'=(N',T,P',S) presentarán la forma:
A→a
A → B1B2...Bm
con A ∈ N', Bi ∈ N' 1<= i<= m, a ∈Σ.
Diremos que esta gramática G' está en Forma Normal de Chomsky intermedia.*/

/* PASO 2 */
N''=N'; P''=∅;

Para toda regla (A → α) de P' **hacer**
Si |α| < 3 **entonces**
Añadir la regla a P" /* Ya esta en FNC */
Sino
Sea α = B_1, B_2 ... B_m con m > 2
Añadir a N' los no terminales { D1, D_2, ..., D_{m-2}};
Añadir a P" el siguiente conjunto de reglas:
A → B_1D_1
D_1 → B_2D_2
...
D_{m-3} → $B_{m-2}D_{m-2}$
D_{m-2} → $B_{m-1}D_m$;
finSi
finPara
La gramática resultado es G"=(N",T,P",S).
finMétodo

Algunos ejemplos de ejecución

Gramática de **entrada**:

A → *bAA* | *aC* | *B*
B → *aSS* | *BC*
C → *CC* | *λ*
S → *SS* | *CA*

Tras eliminar las producciones vacías:

A → *B* | *a* | *aC* | *bAA*
B → *B* | *BC* | *aSS*
C → *C* | *CC*
S → *A* | *CA* | *SS*

Después de eliminar las transiciones unitarias:

A → *BC* | *a* | *aC* | *aSS* | *bAA*
B → *BC* | *aSS*
C → *CC*
S → *BC* | *CA* | *SS* | *a* | *aC* | *aSS* | *bAA*

Gramática sin símbolos no generativos:

A → *a* | *aSS* | *bAA*
B → *aSS*
S → *SS* | *a* | *aSS* | *bAA*

Gramática finalmente simplificada, después de eliminar los símbolos no alcanzables:

A → *a* | *aSS* | *bAA*
S → *SS* | *a* | *aSS* | *bAA*