Cálculo de los conjuntos First y Follow para los símbolos de una gramática

Definiremos a continuación dos funciones asociadas a una gramática independiente del contexto, G que facilitan la construcción de un analizador sintáctico predictivo.

Las funciones first y follow permiten rellenar siempre que sea posible, las entradas de una tabla de análisis sintáctico predictivo para la gramática. También se pueden utilizar los conjuntos de componentes léxicos devueltos por la función follow como componentes de sincronización durante la recuperación de errores en modo pánico.

Definición: Si α es una cadena de símbolos gramaticales, se llama FIRST(α) al conjunto de símbolos terminales por los que comienzan las cadenas derivadas de α. Si $\alpha \Rightarrow^* \varepsilon$, entonces ε también pertenece a FIRST(α).

Definición: Se define FOLLOW(A), para el no terminal A como el conjunto de símbolos terminales que pueden aparecer inmediatamente a la derecha de A en alguna forma sentencial.

Esto es, el conjunto de terminales $a \in$ T tales que haya una derivación de la forma $S \Rightarrow^* \alpha A a ß$ para algunas cadenas α y ß.

Obsérvese que en algún momento de la derivación puede haber existido símbolos entre A y *a* pero derivaron en ε y desaparecieron.

Si A puede ser el símbolo situado más a la derecha en una forma sentencial, entonces $\$ \in$ FOLLOW(A) siendo $ es el símbolo que delimita la entrada por la derecha.

Cálculo de los first

Para calcular FIRST(X) para todos los símbolos gramaticales X, aplíquense las reglas siguientes hasta que no se puedan añadir más terminales o ε a ningún conjunto FIRST.

1. Si X es un terminal, entonces FIRST(X) = {X}
2. Si $X \rightarrow \varepsilon$, entonces FIRST(X) = FIRST(X) U {ε}
3. Si X es un no terminal y $X \rightarrow Y_1Y_2...Y_k$

Si existe un i tal que $a \in$ FIRST(Y_i) y además $Y_1Y_2...Y_{i-1} \Rightarrow^* \varepsilon$
esto es, si ε está en FIRST(Y_1)...FIRST(Y_{i-1}) entonces:

$$\text{FIRST}(X) = \text{FIRST}(X) \cup \{\mathbf{a}\}$$

Si $\varepsilon \in$ FIRST(Y_j) para j = 1, 2,..., k entonces:

$$\text{FIRST}(X) = \text{FIRST}(X) \cup \{\varepsilon\}$$

Por ejemplo, todo los símbolos de FIRST(Y_1) sin duda pertenecerán a FIRST(X).

Si Y_1 no deriva a ε, entonces no se añade nada más a FIRST(X), pero si $Y_1 \Rightarrow^* \varepsilon$, entonces se añaden los elementos de FIRST(Y_2) y así sucesivamente.

Se puede calcular el conjunto FIRST para cualquier cadena de la forma $X_1X_2...X_n$ del siguiente modo:

- Añadir a FIRST($X_1X_2...X_n$) todos los símbolos distintos de ε de FIRST(X_1).
- Si $\varepsilon \in$ FIRST(X_1), añadir también los símbolos distintos de ε de FIRST(X_2).
- Si ε está tanto en FIRST(X_1) como en FIRST(X_2) añadir también los símbolos distintos de ε de FIRST(X_3) y así sucesivamente.
- Por último, añadir ε a FIRST($X_1...X_n$) si para todo i FIRST(X_i) contiene a ε.

Cálculo de los follow

Para calcular FOLLOW(A) para todos los no terminales A, aplicar las siguientes reglas hasta que no se pueda añadir ningún elemento nuevo a ningún conjunto FOLLOW.

1. Incluya $ en FOLLOW(S), donde S es el símbolo inicial de la gramática y $ es el delimitador derecho de la entrada.
2. Si hay una producción $A \rightarrow \alpha B ß$, (con $\beta \neq \varepsilon$) entonces todo los símbolos de FIRST(ß) excepto ε se incluyen en FOLLOW(B).
3. Si hay una producción de la forma $A \rightarrow \alpha B$ o bien $A \rightarrow \alpha B ß$ donde $\varepsilon \in$ FIRST(ß), es decir, $ß \Rightarrow^{*} \varepsilon$ entonces todos los símbolos de FOLLOW(A) se incluyen en FOLLOW(B)

Nótese que en la segunda regla se trata de identificar (para calcular sus follow) símbolos no terminales que no aparezcan al final de las reglas de producción, mientras que en la tercera regla se han de localizar símbolos no terminales que estén al final de la regla de producción, o bien estén seguidos por cadenas ß que derivan en ε.

Ejemplo

Consideremos la siguiente gramática:

1. $S \rightarrow BCc \mid gDB$
2. $B \rightarrow bCDE \mid \varepsilon$
3. $C \rightarrow ca \mid DaB$
4. $D \rightarrow dD \mid \varepsilon$
5. $E \rightarrow gSf \mid c$

y procedamos en primer lugar al cálculo de los conjuntos First para cada uno de sus símbolos no terminales.

Del análisis de la primera producción de la regla 1 ($S \rightarrow BCc$) deducimos solamente que First(B)$\subseteq$First(S). Debido a la segunda producción de la primera regla ($S \rightarrow gDB$) determinamos que {g}$\subseteq$First(S), y como consecuencia de esta producción no se añadirá ningún símbolo adicional a First(S), puesto que First(gDB)={g}. En la primera producción de la segunda regla tenemos que First(bCDE)={b}, y teniendo en cuenta la otra producción para el no terminal B determinamos que First(B)={b, ε}. De modo análogo, deducimos en la cuarta y quinta reglas que First(D)={d, ε} y First(E)={g, c}. Analizando las producciones de la regla 3, determinamos que c$\in$First(C) y también que First(D)$\subseteq$First(C).

En este punto ya se tienen completamente calculados los conjuntos First de los símbolos B, D y E. Ahora hemos de completar el cálculo de los First de S y de C que no quedaron completamente calculados en la primera etapa.

	First
S	g b c d a
B	b ε
C	c d a
D	d ε
E	g c

Figura 4.1 Conjuntos First

Se había determinado que First(D)$\subseteq$First(C). Ya se tiene completamente calculado el conjunto First(D), así que completamos el conjunto First(C): First(C)={c} $\cup$ {d}. Dado que el símbolo D puede derivar en la cadena vacía, hemos de seguir analizando símbolos más a la derecha de D en la segunda producción de la tercera regla ($C \rightarrow DaB$). First(Dab)=First(D) $\cup${a}={d, a}. No analizamos más símbolos en las cadenas de la forma DaB porque hemos alcanzado el símbolo terminal a. Así pues completamos First(C)={c, d, a}. Se Había establecido que First(B)$\subseteq$First(S), y como consecuencia de ello incluiremos en First(S) el no terminal b: First(S) = {g} $\cup$ {b}. Dado que B puede derivar en vacío ($\varepsilon \in$ First(B)) hemos de seguir completando los First(S) incluyendo en este conjunto los First(C) (por el análisis de la primera regla de producción). Así pues First(S)={g, b}$\cup${c, d, a}={g,

b, c, d, a}. Con este paso finalizamos el cálculo de todos los conjuntos First, que quedan como se muestra en la Figura 4.1.

Para el cálculo de los conjuntos Follow de los símbolos no terminales comenzamos por inicializar Follow(S)={$}. Aplicando el segundo paso del algoritmo de cálculo de conjuntos Follow, localizamos símbolos no terminales que no aparezcan al final de las reglas de producción, es decir, producciones de la forma A $\rightarrow$ αBß. Para este tipo de producciones hemos de incluir First(ß)$\subseteq$Follow(B). Así en la primera producción S $\rightarrow$ BCc incluimos {c} en Follow(C) y como consecuencia del análisis de esta producción no incluiremos ningún otro símbolo en Follow de C. En esta misma producción incluiremos First(Cc)={c, d, a} en Follow(B). Análogamente, en la producción S $\rightarrow$ gDB incluimos First(B)={b} en los Follow de D. Volveremos posteriormente a revisar esta producción en el tercer paso del algoritmo de cálculo de conjuntos Follow puesto que como B $\rightarrow$ ε, las producciones S $\rightarrow$ gDB pueden tener la forma S $\rightarrow$ gD (es decir, como si el no terminal D estuviera al final de las frases).

Por el análisis de la producción B $\rightarrow$ bCDE incluimos First(E)={g, c} en el conjunto Follow(D). También en esta producción incluiremos First(DE)={d, g, c} en Follow(C). Analizando la producción C $\rightarrow$ DaB incluimos First(aB)={a} en el conjunto Follow(D), y por último incluimos {f} en los Follow(S) como consecuencia del análisis de la regla E $\rightarrow$ gSf.

Pasamos a continuación al tercer paso del algoritmo de cálculo de conjuntos Follow, localizando símbolos no terminales que se encuentren al final de alguna regla de producción (o que no estén al final pero vayan seguidos de cadenas que pueden derivar en vacío). Así encontramos en la primera regla la producción S $\rightarrow$ gDB y por ello incluiremos Follow(S) en Follow(B). Dado que B $\rightarrow$ ε también incluiremos Follow(S) en Follow(D). Por el análisis de B $\rightarrow$ bCDE incluimos Follow(B) en Follow(E). Por la producción C $\rightarrow$ DaB incluiremos Follow(C) en Follow(B). Una vez considerados todos estos cómputos hemos finalizado el cálculo de los conjuntos Follow, que queda como muestra la Figura 4.2.

	First	Follow
S	g b c d a	$ f
B	b ε	c d a f $ g
C	c d a	c d g
D	d ε	b g c a $ f
E	g c	c d a f $ g

Figura 4.2 Conjuntos First y Follow

Práctica 4.1. Cálculo de los conjuntos first y follow

El objetivo de la práctica consiste en leer una gramática libre de contexto desde un fichero de entrada y calcular los conjuntos First y Follow de todos los símbolos no terminales imprimiéndolos en pantalla.

Fichero de descripción de la gramática.

Los ficheros de texto que describen gramáticas tienen la extensión .gra y la siguiente estructura:

- Línea 1: Número total de símbolos (tanto terminales como no terminales) que posee la gramática.
- Línea 2: Lista de todos los símbolos terminales de la gramática, separados por espacios en blanco. Cada terminal se representa mediante un carácter ASCII.
- Línea 3: Lista de todos los símbolos no terminales de la gramática, separados por espacios en blanco y representados también mediante caracteres.
- A partir de la cuarta línea el fichero contendrá las reglas de producción de la gramática, con una línea para todas las reglas de producción de cada uno de los no terminales anteriores. Cada línea contendrá:
    - Carácter correspondiente al no terminal.
    - Flecha '->' (caracteres menos y mayor).

- Alternativas asociadas al no terminal anterior, formadas por una sucesión de caracteres correspondientes a símbolos terminales o no terminales, o 'ε' para indicar la cadena vacía. Las diferentes alternativas de una producción vendrán separadas entre sí por el símbolo '|'.

Se supondrá que el símbolo de arranque de la gramática será el primer no terminal que aparece en la tercera línea.

La rutina que lea el fichero de definición de la gramática deberá ignorar completamente todas las líneas que comiencen con el carácter '#'. Ese carácter se utilizará para poner comentarios a los ficheros de gramáticas indicando cualquier característica de los mismos que se desee.

Así para representar la siguiente gramática:

S -> S**ab**T | T**a** | ε | **b**
T -> ε | **b**

en la que los símbolos ***a*** y ***b*** son terminales, y {*S* , *T*} es el conjunto de símbolos no terminales, siendo *S* el símbolo de arranque de la gramática, el fichero tendrá el contenido que muestra la Figura 4.3.

```
4
a b
S T
S -> SabT | Ta | ε | b
T -> ε | b
```

Figura 4.3 Fichero de representación de una gramática

Formato de la salida.

La salida del programa puede redirigirse a un fichero y deberá escribir los conjuntos First y Follow de cada símbolo no terminal de la gramática. El carácter '$' se utilizará para representar el final de fichero.

Para la gramática anterior, la salida del programa es:

```
First(S) = { a b ε }    Follow(S) = { a $ }
First(T) = { b ε }      Follow(T) = { a $ }
```

Detalles de implementación.

Para facilitar la realización de la práctica se propone a continuación un posible análisis y una estructura de datos que lo soporta. El lector es libre de utilizar estas ideas o bien las suyas propias.

Asignaremos un número entero a cada símbolo de la gramática, comenzando en 0 hasta el número de terminales menos uno (*num_term*-1) para todos los terminales, y desde del número de terminales (*num_term*) hasta el número de terminales más el número de no terminales menos uno (*num_term*+*num_no_term*-1) para los no terminales. La relación entre dicho número y su carácter correspondiente se encuentra descrita en el vector *alfabeto*, de modo que para el ejemplo anterior tendremos:

```
*alfabeto[0] = 'a'
*alfabeto[1] = 'b'
*alfabeto[2] = 'S'
*alfabeto[3] = 'T'
```

Para almacenar la reglas se utilizarán las estructuras que se muestran en la Figura 4.4.

```
typedef int simbolo;           /* Símbolos (T y NT) */
typedef simbolo *alternativa;  /* Alternativa */
typedef alternativa *regla;    /* Varias alternativas */
                               /* asociadas a un mismo NT (reglas) */
```

Figura 4.4 Estructuras para almacenar las reglas de la gramática.

Las variables de tipo *simbolo* almacenarán los números asociados a los símbolos de la gramática. Las variables de tipo *alternativa* almacenarán un vector de símbolos terminados con la constante FINALT que representará una alternativa. Las variables de tipo *regla* almacenarán un vector de alternativas (punteros a *simbolo*) asociadas a un mismo no terminal y acabado en NULL. Teniendo un vector con

tantos elementos de tipo *regla* como no terminales existan en la gramática tendremos almacenada toda la gramática.

La Figura 4.5 muestra gráficamente la estructura creada para almacenar las diferentes alternativas del símbolo no terminal S del ejemplo anterior y su contenido.

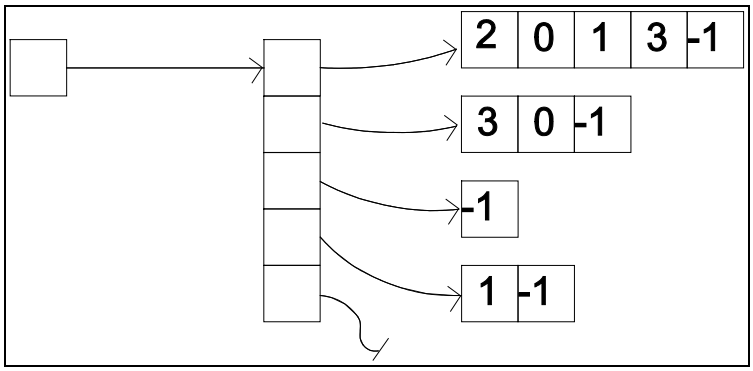

Figura 4.5 Estructura que almacena la producción S→SabT|Ta|ε|b

En el servidor ftp del CSI se encuentra disponible el fichero

ftp://ftp.csi.ull.es/pub/asignas/AUTOMALF/p04_fyf/nucleo.c

en el que se encuentran estas definiciones, así como el código de una función, *LeeGram()* que se encarga de leer el fichero de definición de la gramática y crear en memoria la estructura de datos necesaria. Esta función *LeeGram()* no contempla la funcionalidad requerida de ignorar las líneas de comentarios (las que comienzan con el carácter '#'): el alumno deberá incorporarle esta funcionalidad.

Si asignáramos la salida de la función *LeeGram* a la variable *gram*, para acceder al segundo elemento de la segunda alternativa asociada al primer no terminal se podría utilizar:

gram[0][1][1]

que para el ejemplo sería el símbolo *a* de la cadena *Ta*.

Las alternativas para *T* en el ejemplo estarían apuntadas por *gram*[1].

Pasos a implementar

Para la correcta realización de la práctica se propone dividirla en tres etapas:

1. Identificación de todos los símbolos no terminales anulables.
2. Cálculo del conjunto First de cada no terminal.
3. Cálculo del conjunto Follow de cada no terminal.

Símbolos anulables

Un símbolo no terminal es anulable si puede derivar en la cadena vacía en uno o más pasos. Para el cálculo de los símbolos anulables se propone el algoritmo de la Figura 4.6, que está basado en la obtención mediante búsqueda en anchura de los nodos alcanzables dentro de un grafo partiendo de un conjunto de vértices.

```
Marcar como anulables todos los no terminales que
  posean alguna alternativa ε
repeat
  for (todos los no terminales con una alternativa formada
        por no terminales anulables) do
    marcar dicho no terminal como anulable
until no se marquen no terminales en el paso anterior
```

Figura 4.6 Cálculo de símbolos anulables

Conjunto First de un símbolo

Para este cálculo se propone un algoritmo de búsqueda de vértices alcanzables en un grafo desde un vértice dado. En nuestro caso los vértices del grafo serán los distintos símbolos de la gramática, y existirá un arco entre un símbolo S y otro T siempre que exista una alternativa para S de la forma:

$$S \rightarrow \alpha T \beta$$

siendo α una cadena de cero o más símbolos anulables.

Por lo tanto los elementos del conjunto First de un determinado símbolo se hallarán buscando los terminales alcanzables desde el nodo asociado a dicho símbolo. Dicho algoritmo puede ser optimizado al utilizar los First de otros no terminales que ya hayan sido calculados previamente.

Follow

Puede implementarse un algoritmo que recoja en cada alternativa en que aparezca un determinado no terminal los elementos de First del conjunto de símbolos que le siguen en dicha alternativa, y si dicho conjunto de símbolos es anulable puede asociársele además el no terminal asociado a dicha alternativa. El pseudocódigo se presenta en la Figura 4.7.

```
for (todas las producciones de la gramática) do
  for (todos los símbolos S no terminales dentro de la producción) do
    sea la producción de la forma T-> αSβ
    añadir First(β) (excepto ε) a Follow(S)
    if β es anulable then
      añadir el no terminal T a Follow(S)
```

Figura 4.7 Cálculo del conjunto Follow

Un paso posterior del algoritmo se encargaría de transformar los no terminales asociados a los Follow en terminales. Para ello basta con un algoritmo que recoja los terminales Follow de los no terminales alcanzables a través de la lista de no terminales asociados como Follow, basado en el conjunto de vértices alcanzables en un grafo desde uno dado.

Esto es, suponer un grafo cuyos vértices son los no terminales, y en el que existirá una arista de S a T si T esta incluido dentro de Follow de S. Calculando desde un vértice dado el conjunto de nodos alcanzables, y luego hallando la unión de cada uno de los Follow de dichos nodos que sean terminales tendremos la solución para dicho vértice. Este algoritmo puede optimizarse utilizando los Follow que ya sean definitivos.

Notas finales

Cuando se habla de grafos en los algoritmos anteriores, ello no significa que sea necesario crear físicamente en memoria dichos grafos.

En el servidor ftp del Centro Superior de Informática se pueden obtener los programas

ftp://ftp.csi.ull.es/pub/asignas/AUTOMALF/p04_fyf/fyf

ftp://ftp.csi.ull.es/pub/asignas/AUTOMALF/p04_fyf/fyf.exe

que tienen el comportamiento deseado. También se pueden obtener del servidor ftp los ficheros

ftp://ftp.csi.ull.es/pub/asignas/AUTOMALF/p04_fyf/test*.gra

que son una serie de ejemplos de ficheros de definición de gramáticas con los que el alumno puede comprobar el buen funcionamiento de su programa.

Para realizar la práctica se puede utilizar cualquiera de las librerías de funciones que se han utilizado anteriormente, y como bibliografía se recomienda la obra de Aho [Aho90].

Bibliografía

[Aho90] Aho, A., Sethi, R., y Ullman, J. *Compiladores. Principios, Técnicas y Herramientas*. Addison-Wesley Iberoamericana, 1990. ISBN: 0-201-62903-8