C O M U N I C A C I O N E S

APLICACION DEL HASH-CODING AL TRATAMIENTO AUTOMATICO DE GRAMATICAS

Por Roberto Moya Quiles e Isidro Ramos Salavert

I. INTRODUCCION

En el tratamiento automático de la sintaxis de los lenguajes de programación, se presentan nuevos problemas que requieren para su resolución una tecnología nueva, por ahora no estructurada, que llena las revistas especializadas de algoritmos que resuelven determinados aspectos de un problema.

En el trabajo que desarrollamos conjuntamente sobre el diseño de un metaensamblador (y análogamente podríamos hablar del diseño de un metacompilador, compilador de compiladores, en definitiva, de todo sistema metaprogramado) la descripción formalizada (en el aspecto sintáctico) de los lenguajes que intervienen en el proceso la hacemos en el metalenguaje FNB (forma normal de Backus) codificada de forma que permita un tratamiento cómodo al ser tomada como dato por nuestro sistema.

El tratamiento ulterior que sufren estas descripciones sintácticas (paso a forma determinista, ...) hace que necesitemos extraer determinada información, que en nuestro caso podemos resumir en:

1°) Tablas de referencias cruzadas para metavariables y caracteres terminales.
2°) Terminales que encabezan cada metavariable.
3°) Metavariables que empiezan por un terminal dado.

Exponemos en el punto II los algoritmos clásicos para la resolución de estos problemas basados en la teoría de relaciones, el punto III será la exposición de nuestro método y en el IV haremos una comparación de ambos métodos enumerando las ventajas e inconvenientes.

II. RELACIONES BINARIAS FUNDAMENTALES DEFINIDAS POR UNA GRAMATICA LIBRE DE CONTEXTO. APLICACIONES

Dada la gramática Tipo 2 $G = \{V_T, V_N, P, S\}$ podemos definir las siguientes relaciones binarias, impuestas por P, sobre los elementos de $V = V_T \cup V_N$ (conjunto finito) (1).

1) Dependencia simple (representada por R_S)

Diremos que $\Sigma \in V_N$ y $\alpha \in V$ cumplen: $\Sigma R_s \alpha$ si y solamente si α aparece en una parte derecha de la regla en la que Σ es la parte izquierda.

O sea:

$$\Sigma \rightarrow \sigma_1 \alpha \sigma_2 \quad , \quad \sigma_1 \in V^* \quad , \quad \sigma_2 \in V^*$$

En el grafo de dependencia simple habrá una arista que una Σ y α.

Representando mediante una matriz booleana esta relación binaria, las columnas de esta matriz nos proporcionan las tablas de referencias cruzadas que dábamos en el apartado 1) de la Introducción.

2) Dependencia a izquierdas (representada por R_I)

Diremos que $\Sigma \in V_n$ y $\alpha \in V$ cumplen $\Sigma R_I \alpha$ si y solamente si α aparece como primer símbolo (distinto de la cadena vacía) en una producción de la gramática en la que Σ es la parte izquierda.

O sea:

$$\exists\ \Sigma \rightarrow \Sigma_1 \Sigma_2 \ldots \Sigma_p \alpha \phi$$
$$p \geq 0$$
$$\Sigma_1, \Sigma_2 \ldots \Sigma_p \in V_N \quad , \quad \phi \in V^*$$
$$\Sigma_1, \overset{*}{\rightarrow} \phi$$
$$\Sigma_2, \overset{*}{\rightarrow} \phi$$
$$\vdots$$
$$\Sigma_p, \overset{*}{\rightarrow} \phi$$

Donde por $\overset{*}{\rightarrow}$ representamos el cierre transitivo de la relación $\rightarrow$ ("produce") que definen las producciones de una gramática.

Tomando esta relación binaria y representándola en forma de la matriz booleana A(n,n) siendo n=cardinal (V), obtenemos una representación del grafo de dependencia a izquierdas fácilmente representable en un ordenador.

Si nos interesan los terminales que pueden encabezar una clase sintáctica dada o las clases sintácticas que pueden empezar por un terminal dado (puntos 2 y 3 de la Introducción) es suficiente con calcular el cierre transitivo de esta relación que representaremos por R_I^+ y que usando el teorema de Warshall calcularíamos mediante el siguiente algoritmo expresado en ALGOL (2)

```
Procedure WARS (A,N)
boolean array A[1:n,1:n];
begin integer i,j,k;
      for i: = 1 step 1 until n do
      for j: = 1 step 1 until n do
           if A(j,i) then
           for k: = 1 step 1 until n do
               A(j,k): = A(j,k) ∪ A(i,k);
end WARS
```

esto nos daría una nueva matriz $A^+(n,n)$ en la que, considerando las filas correspondientes a las metavariables, obtendríamos los terminales que pueden encabezarlas y considerando las columnas correspondientes a los terminales, las clases sintácticas que empiezan cada uno de ellos.

Veamos mediante un ejemplo sencillo la aplicación de estos métodos:

Dada la gramática $G\{V_T, U_N, P, S\}$

$V_T = \{a, b, (,), +, \dashv\}$

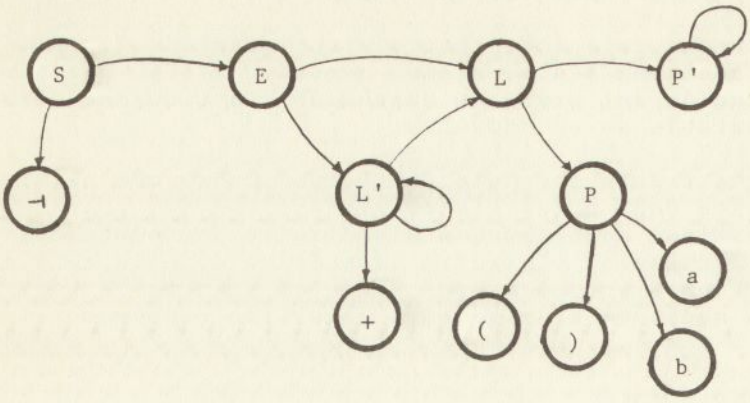
$V_N = \{E, L, L', P, P', S\}$

axioma: S

Producciones: P

$E \rightarrow LL'\dashv$	$P \rightarrow a$
$L' \rightarrow +LL'$	$P \rightarrow b$
$L' \rightarrow \phi$	$P \rightarrow (E)$
$L \rightarrow PP'$	$S \rightarrow E$
$P' \rightarrow PP'$	
$P' \rightarrow \phi$	

El grafo de dependencia simple sería:



que aceptaría la representación matricial:

V_N		S	E	L'	L	P'	P	a	b	(	)	+	⊣
V_N	S	0	1	0	0	0	0	0	0	0	0	0	1
	E	0	0	1	1	0	0	0	0	0	0	0	0
	L'	0	0	1	1	0	0	0	0	0	0	1	0
	L	0	0	0	0	1	1	0	0	0	0	0	0
	P'	0	0	0	0	1	1	0	0	0	0	0	0
	P	0	1	0	0	0	0	1	1	1	1	0	0
V_T	a	0	0	0	0	0	0	0	0	0	0	0	0
	b	0	0	0	0	0	0	0	0	0	0	0	0
	(	0	0	0	0	0	0	0	0	0	0	0	0
	)	0	0	0	0	0	0	0	0	0	0	0	0
	+	0	0	0	0	0	0	0	0	0	0	0	0
	⊣	0	0	0	0	0	0	0	0	0	0	0	0

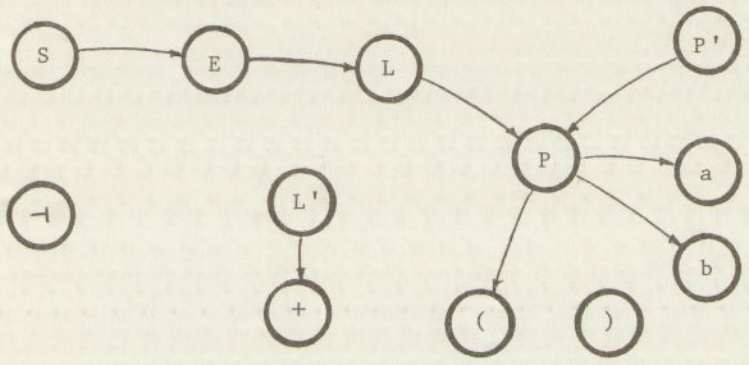
En la que vemos:

Escogiendo una columna cualquiera, hay un 1 en la regla donde aparece esta clase sintáctica o terminal (a la derecha) y un cero en caso contrario. Así:

Símbolo	Aparece en
E	S , P
L'	E , L'
⋮	⋮
a	P

Que son las tablas de referencia cruzadas a las que hacíamos referencia en I-1).

El grafo de dependencia a izquierdas sería:



que se puede representar matricialmente en la forma que se da en la primera tabla de la página siguiente.

Calculando el cierre transitivo de esta matriz usando el teorema de Warshall, llegaríamos a la matriz que se da en la segunda tabla de la página siguiente.

Escogiendo una fila cualquiera, hay un 1 en la columna cuyo símbolo puede encabezar la clase sintáctica asociada con la columna. Por ejemplo:

Símbolo	Puede empezar por
S	E , L , P , a , b , (
E	L , P , a , b , (
⋮	⋮

		V_N						V_T					
		S	E	L'	L	P'	P	a	b	(	)	+	⊣
V_N	S	0	1	0	0	0	0	0	0	0	0	0	0
	E	0	0	0	1	0	0	0	0	0	0	0	0
	L'	0	0	0	0	0	0	0	0	0	0	1	0
	L	0	0	0	0	0	1	0	0	0	0	0	0
	P'	0	0	0	0	0	1	0	0	0	0	0	0
	P	0	0	0	0	0	0	1	1	1	0	0	0
V_T	a	0	0	0	0	0	0	0	0	0	0	0	0
	b	0	0	0	0	0	0	0	0	0	0	0	0
	(	0	0	0	0	0	0	0	0	0	0	0	0
	)	0	0	0	0	0	0	0	0	0	0	0	0
	+	0	0	0	0	0	0	0	0	0	0	0	0
	⊣	0	0	0	0	0	0	0	0	0	0	0	0

		V_N						V_T					
		S	E	L'	L	P'	P	a	b	(	)	+	⊣
V_N	S	0	1	0	1	0	1	1	1	1	0	0	0
	E	0	0	0	1	0	1	1	1	1	0	0	0
	L'	0	0	0	0	0	0	0	0	0	0	1	0
	L	0	0	0	0	0	1	1	1	1	0	0	0
	P'	0	0	0	0	0	1	1	1	1	0	0	0
	P	0	0	0	0	0	0	1	1	1	0	0	0
V_T	a	0	0	0	0	0	0	0	0	0	0	0	0
	b	0	0	0	0	0	0	0	0	0	0	0	0
	(	0	0	0	0	0	0	0	0	0	0	0	0
	)	0	0	0	0	0	0	0	0	0	0	0	0
	+	0	0	0	0	0	0	0	0	0	0	0	0
	⊣	0	0	0	0	0	0	0	0	0	0	0	0

O sea, tenemos fácilmente los terminales que pueden encabezar una clase sintáctica (punto I-2)).

Si, por el contrario, escogemos una columna cualquiera (correspondiente a un terminal) la presencia de un 1 en ella indica que la clase sintáctica asociada con la fila en la que está puede empezar por él, con lo que fácilmente podemos construir la siguiente tabla:

Símbolo	Clases sintácticas que pueden encabezarlo
a	S , E , L , P', P
b	S , E , L , P', P
(	S , E , L , P', P
:	:
:	:

con lo que tenemos resuelto el punto I-3) y con ello la forma tradicional de resolver este problema. La memoria ocupada por la implementación de estos algoritmos (al margen de la procedure ALGOL, del algoritmo de definición de los grafos) sería:

cardinal (V) x cardinal (V)

que en nuestro sencillo ejemplo es 12 x 12 = 144 posiciones, o sea, depende de: n^2 siendo n=cardinal (V). Podemos pensar que si el lenguaje tratado es uno cualquiera de los habituales, este método es prohibitivo en memoria.

Se podría usar menos memoria a base de

1) Representar las matrices booleanas usando un bit por entrada. Pero esto aumenta mucho el tiempo de tratamiento de las tablas (3).
2) Como son matrices muy vacías hacer una representación sobre listas circulares ortogonales (4). Pero esto dificultaría también la implementación del algoritmo de WARSHALL.

Veamos a continuación nuestro método.

III. HASH-CODING Y GRAMATICAS

El procedimiento que hemos utilizado para resolver nuestro problema hace uso de una tabla hash, en la que las colisiones se representan sobre listas lineales (en la terminología de Knuth) implementadas sobre vectores.

Veamos paso a paso nuestro algoritmo.

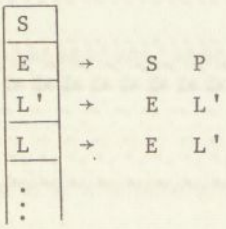
1) Obtención de las tablas I-1 y I-2 (tablas de referencias cruzadas).

Barremos la gramática de forma que obtenemos la representación en forma de diccionario de siguientes del grafo de dependencia simple:

Diccionario de siguientes (en R_s)

S	E , ⊣
E	L , L'
L'	+ , L , L'
L	P , P'
:	:
:	:

Tomando como clave, uno a uno, los siguientes de cada elemento, metemos en las listas de overflow los antecedentes asociados con cada clave. Al final del proceso tenemos, en forma de tabla hash las tablas de referencias cruzadas



vector Hash — listas de overflow

Este punto lo resolveremos en forma análoga al procedimiento clásico, la única diferencia es que tomamos como representación del grafo el diccionario de siguientes en vez de la matriz booleana.

Para resolver el problema siguiente: terminales por los que empieza cada clase sintáctica y clases sintácticas que empiezan por un terminal dado, utilizamos el siguiente algoritmo: recolectemos, analizando la gramática, las cabeceras de cada parte derecha de las producciones construyendo así el diccionario de siguientes del grafo de dependencia a izquierdas:

Diccionario de siguientes (en R_I)

símbolo	siguiente
S	E
E	L
L	P
L'	+
P'	P
P	a, b, (

y a partir del diccionario de siguientes buscamos el diccionario de antecedentes, inicialmente sólo para los terminales. Esto nos da:

Diccionario de antecedentes

símbolo	antecedente
+	L'
a	P
b	P
c	P
(	P

Buscando en la tabla de antecedentes anterior, si encontramos una clase sintáctica (por ejemplo en a\|P, la P) con ella buscamos en la tabla de siguientes cuál es su símbolo antecedente (en nuestro caso P : P' y L) y añadimos estos antecedentes en el diccionario de antecedentes y en la fila del terminal donde encontramos la clase sintáctica. Esto lo haremos en todos los sitios donde aparezca esta clase sintáctica, lo que nos dará:

+	L'
a	P, P', L
b	P, P', L
c	P, P', L
(	P, P', L

Nuevamente (y ya sin considerar la clase sintáctica ya usada) aplicamos el mismo criterio para todas las demás clases sintácticas que aparecen como antecedentes en la nueva tabla, lo que da al final la tabla de clases sintácticas que empiezan por un terminal dado.

+	L'
a	P, P', L, E, S
b	P, P', L, E, S
c	P, P', L, E, S
(	P, P', L, E, S

El proceso de acceso a esta tabla de antecedentes se hace usando como clave el terminal que estemos considerando y usando como función de "mapping" el {AND} lógico con '3F' (hexadecimal) del propio terminal, lo que nos da un acceso directo al vector Hash, en donde encontramos una referencia a su lista de overflow y empezamos una búsqueda secuencial hasta encontrar una clase sintáctica no tratada todavía.

Al final disponemos de una tabla que nos da para cada símbolo terminal las clases sintácticas que pueden empezar por él. Sólo nos queda obtener los terminales que encabezan cada clase sintáctica; esta tabla la obtenemos sin más que buscar en las listas de overflow cada clase sintáctica y añadir los terminales que las encabezan en la entrada correspondiente a cada clase sintáctica.

IV. CONCLUSIONES

El método que hemos descrito lo hemos implementado sobre UNIVAC-9300. En nuestro caso y dadas las limitaciones de memoria, el tomar como representación del grafo el diccionario de siguientes (y/o antecedentes) ha significado un considerable ahorro de memoria, unido a una fácil implementación.

El proceso ha sido aplicado a la gramática del lenguaje 70/13(5), cuyo traductor estamos metaprogramando.

Nuestros requisitos de memoria son considerablemente inferiores a los del método clásico, aunque el tiempo de tratamiento sea mayor.

Damos en el apéndice los resultados obtenidos en forma de output directo de la máquina.

Bibliografía

1) BOUSSARD, J.C.- Analyse syntaxique et compilation. IMAG. Grenoble, 1968-69.
2) GOMEZ BUENO, L., RAMOS SALAVERT, I.- Cierre transitivo de una relación binaria entre los elementos de un conjunto finito: Teorema de Warshall.- Nota interna del Instituto de Informática. Madrid, 1971.
3) DUBY, J.J., MALING, G.- A set-theoretic application of IBM 7094 computer instructions.- Report n° 6, IBM. 1964.
4) KNUTH, D.E.- The art of computer programming.- Vol. 1. Addison Wesley, 1968.
5) FERNANDEZ-FLOREZ, I., GARCIA CAMARERO, E., RAMOS SALAVERT, I. A basic language oriented to secondary schools.- IFIP 71. Ljubljana. August 1971.

APENDICE

DICCIONARIO DE SIGUIENTES (GRAFO DE DEPENDENCIA A IZQUIERDAS)

```
<CU      >     U      I
<CONTINS>     <CT       >   <CN     >
<CT      >     (
<CN      >     =
    .
    .
    .
```

CLASES SINTACTICAS QUE PUEDEN EMPEZAR POR CADA TERMINAL

```
A   <CR      ><CARACT ><CADENA ><CK      >
B   <CARACT ><CADENA ><CK      >
C   <CARACT ><CADENA ><CK      >
D   <MDAR    ><CARACT ><INSTRUC><CADENA ><CK     >
E   <ESTNST ><MDARCON><CARACT ><INSTRUC><CADENA><CK>
F   <CTINST ><CARACT ><INSTRUC><CADENA ><CK     >
G   <CARACT ><CADENA ><CK      >
H   <CTINST ><CARACT ><INSTRUC><CADENA ><CK     >
I   <MDARCON><CU      ><CARACT ><CADENA ><CK     >
    .
    .
    .
```

TERMINALES QUE ENCABEZAN CADA CLASE SINTACTICA

```
<INSTRUC>DEFHLMPRST
<ESINST >EL
<CK      > ABCDEFGHIJKLMNOPQRSTUVWXYZ
<MDAR    >D
<MDARCON>EI
<MDINST >T
    .
    .
    .
```