ANALIZADOR SINTACTICO POR EL METODO DE LAS MATRICES DE TRANSICION Y GENERACION DE UNA C-ESTRUCTURA PARA UN MINI-LENGUAJE.

Por Pedro Díaz Muñoz y Teodomiro García Vadillo

1.- INTRODUCCION

1.1. - Introducción a los analizadores sintácticos.

1.1.1. - Misiones del analizador sintáctico.- El analizador sintáctico debe reconocer una frase de input de acuerdo con la gramática del lenguaje, emitiendo errores en caso de que la frase no se ajuste a dicha gramática.

En la segunda fase, que puede hacerse paralelamente a la anterior, se puede generar una C-estructura (cuyo significado posteriormente se explicará) y que está relacionada con el orden en que se hacen las reducciones, en el proceso de reconocimiento. Es decir, con las precedencias, que en una reducción tienen unas partes de las sentencias a reconocer sobre otras.

La entrada para el analizador sintáctico consiste en la cadena de unidades sintácticas, obtenidas por el preprocesador de la cadena de símbolos originales del lenguaje fuente.

1.1.2. - Tipos de analizadores.- Básicamente se tienen dos métodos, analizadores Top-down y Bottom-up.

El Top-down construye el árbol sintáctico a partir de la raiz y continuándole según sea la sentencia. Es decir, prueba distintas producciones de la gramática aceptando las que se ajusten a la sentencia.

El método Bottom-up busca la parte de la sentencia que se ha generado la última (se supone una generación canónica), llamada asidero o bien frase primera, y es esta parte la que se reduce en primer lugar, sustituyéndola por el no terminal correspondiente (es decir el que la produjo). Se continúa el proceso reiteradamente hasta llegar a la raiz.

Para el método Top-down una gramática ideal sería una gramática recursiva, ya que en ese caso no existirían problemas de Back-up (vuelta atras). Estos problemas, debidos a que la gramática no es determinista, se pueden resolver, sin embargo, utilizando una pila, complicándose el programa en este caso bastante más.

Para el sistema Bottom-up existen distintos métodos, según la gramática que se emplee. Así tenemos el método de precedencia simple para gramáticas de precedencia simple, o de precedencia (m, n) para las gramáticas de precdencia (m, n).

También hay otro método para gramáticas de precedencia de -

operadores, otro para las de contexto acotado, etc.

En particular para el método de matrices de transición explicaremos en la sección 2.1. el tipo de gramática a emplear.

1.2. - Introducción al método de matrices de transición.

1.2.1. - Modo de trabajo.- Para este método se va a necesitar un Stack S y una variable auxiliar V, además de un vector R que contendrá la cadena de input dada por el preprocesador.

Se construirá una matriz en la que cada fila corresponderá a una cabeza (que acabe en un símbolo terminal) de una parte derecha de una producción. A cada columna le corresponde uno de los símbolos terminales, (estos terminales son precisamente las clases sintácticas que obtiene el preprocesador).

Cada elemento de la matriz será un número que corresponderá a una subrutina.

El Stack S va a contener cabezas de parte derecha que acaben en símbolo terminal.

El método trabaja de la siguiente forma:

En cada paso el elemento más alto de la pila nos da una fila, y el terminal de la cadena de input que estamos tratando en ese momento nos da una columna. Ambos juntos nos dan un elemento de la matriz que determinará una subrutina. Esta subrutina, atendiendo al contenido de la variable V, hará una reducción o meterá en la pila el símbolo actual de R y pasará al símbolo siguiente. En el caso de que se haya efectuado una reducción meterá en la variable V el no terminal al que se ha reducido.

La construcción de las rutinas se verá en la sección 3.2.

1.2.2. - Comparación con otros métodos.- Este método es bastante parecido al de precedencia de operadores, con la ventaja de que utiliza un solo Stack en lugar de dos.

Además en cada posición de esta pila puede haber cadenas de símbolos terminales o no, con las condiciones de ser cabezas de producción y de que acaben en un símbolo terminal.

La primera condición se justifica porque así podemos estar seguros de que todos los símbolos que están en una misma posición de la pila se reducen al mismo tiempo.

La segunda condición se ha de imponer en el caso de que la gramática sea de operadores, que es el tipo de gramática para el que, generalmente, se emplea este método. Esta condición viene justificada por el siguiente teorema:

Teorema.- En el método de matrices de transición para una

gramática de operadores, ninguna posición del Stack puede acabar en un no terminal.

Demostración.- La definición de gramática de operadores - es aquella en la que ninguna producción puede ser de la forma - V : : = ... V W ... , donde V y W son no terminales. Supongamos que hubiera en una de las posiciones de Stack, 1º una cadena de símbolos acabada en un no terminal. Se irían haciendo reduccio-- nes y estas reducciones se introducirán en la variable U. Llega- ría un momento en que la posición más alta del Stack sería 1º. Al haber llegado a esta posición quitando elementos del Stack, - es decir, haciendo reducciones U sería no vacío y por tanto nos encontraremos con dos no terminales, ya que U debe ser siempre no terminal por ser parte izquierda de una producción de la gra- mática. Con esto llegaríamos a una contradicción con el hecho de que la gramática sea de operadores.

Evidentemente si la gramática no es de operadores y tiene alguna producción de la forma: U = AVWB, donde V y W son no ter- minales, entonces podrá formar parte del Stack la cadena A V. - Sin embargo nosotros nos limitaremos a gramáticas de operadores y la restrición anterior será válida.

Este método es más rápido que muchos otros debido a que no es necesaria la búsqueda en tablas.

Otra ventaja es que al ser cada subrutina independiente de las demás se puede dividir el trabajo entre varias personas sin que ninguna de ellas tenga necesidad de conocer el trabajo de - las otras.

Por último, y quizá sea la más importante, se tiene una - gran facilidad para tratar los distintos tipos de errores. Cada error de un tipo distinto viene dado por un distinto elemento de la matriz. En cada caso se puede llamar a una rutina que según - la importancia del error detenga el programa, pase a la senten-- cia siguiente, ignore el error, etc. ... En resumen, la recupe-- ración de errores se puede incorporar de una forma muy sencilla, sin ningún trabajo auxiliar de programación, como haría falta en otros métodos, sino simplemente aumentando el número de rutinas de la matriz.

Sin embargo, la gran desventaja es que la matriz de transi ción, si el lenguaje es muy amplio, ocupa mucha memoria. Se pue- de obviar teniendo en cuenta que muchas intersecciones de filas y columnas no pueden ocurrir nunca, con lo que utilizando una ta bla se reducen las posiciones ocupadas, aunque aumenta el tiempo de búsqueda.

1.3. - Generación de la C-estructura.- La generación de la C-estructura es un paso intermedio entre el análisis sintáctico y la generación de código.

Creando la C-estructura se ahorran sentencias en el código objeto generado.

Una C-estructura es cualquier estructura, árboles, n-uplas, que haga ver como se ha ido produciendo la sentencia de input, teniendo en cuenta el orden en que se han hecho las reducciones.

La C-estructura se puede generar paralelamente al análisis sintáctico, es decir, al mismo tiempo que se hace una reducción , esa reducción se incorpora a la C-estructura que se genera. También se puede generar después del análisis sintáctico. Nosotros lo hacemos en paralelo.

2. - GRAMATICA

2.1. - Gramática necesaria para este método.- Como dijimos anla gramática más utilizada para este método es una gramática de - operadores.

Definición: O.G. - Es aquella que no tiene ninguna producción de la forma : U: : = ... V W ..... tales que V y W son no terminales.

Sin embargo, este método se puede emplear para gramáticas - que no sean de precedencia de operadores, aun cuando aquí empleemos una gramática de operadores.

2.2. - Gramática

```
<PROGRA> ::=    <INSTRU>
                <PROGRA> ; <INSTRU>

<INSTRU> ::=    <DECLARA>
                <NODECLARA>
                <CONTROL>

<DECLARA> ::=   REAL <LETRA>
                INTERGER <LETRA>
                LOGICAL  <LETRA>

<<NODECLARA> ::= <ASSIGNA>
                <CON DIC>

<ASSIGNA> ::=   <LETRA> = <EXPRE>

<EXPRE> ::=     <EXPRE> .OR. <EXPRE 1>
                <EXPRE 1>

<EXPRE 1> ::=   <EXPRE 1> .AND. <EXPRE 2>
                <EXPRE 2>

<EXPRE 2> ::=   <EXPRE 2> .RO. <EXPRE 3>
                <EXPRE 3>

<EXPRE 3> ::=   <EXPRE 3> + <TERM>
                <EXPRE 3> - <TERM>
                <TERM>

<TERM> ::=      <TERM> * <FACTOR>
                <TERM> / <FACTOR>
                < FACTOR>
```

```
<FACTOR> ::=    ( <EXPRE> )
                <LETRA>
                .NOT. <FACTOR>
                <NUMERO>
<CON DIC> ::=   <IFCLAH> THEN <ASSIGNA> ELSE <NODECLARA>
<IFCLAH> ::=    IF <EXPRE>
<CONTROL> ::=   END
<LETRA> ::=     A
                B
                .
                .
                .
                .
                .
                Z
<NUMERO>::=     0/1/2/3/4/5/6/7/8/9/
```

3.- PROGRAMA

3.1. - Variables usadas.-

3.1.1. - Variables usadas para el analizador sintáctico.-

Tenemos:

a) Un Stack S con un pointer I.

b) Un vector R con un índice J, en el que está la cadena de input, y que tiene un delimitador ($) de principio y otro de - final.

c) Una matriz, MAT (S,R), que es la matriz de transición donde van las direcciones de las subrutinas. Asi MAT (S(I),R(J))= un n°que corresponde a la subrutina que se deberá llamar al tener S(I) como símbolo en lo alto de la pila y R(J) como símbolo de - input.

d) Una variable U

e) Un vector V(J) paralelo a R(J), R contendrá los códigos numéricos de los terminales y V los terminales.

3.1.2. - Variables usadas por la C-estructura.

a) Variable U P, que es el valor de V anterior a la última reducción.

b) IU es un n°natural que se utiliza para distinguir no terminales correspondientes al mismo tipo. Ejemplo:

a+b y c-d ambas se reducen a EXPRE 3, pero son distintas IU valdrá distinto para cada una de estas expresiones.

c) LS es un vector alfanumérico que contiene todos los elementos que pueda tener S.

d) LU es análogo a LS para U.

e) JS es una nueva pila paralela a la S, es decir utiliza el mismo pointer, y hace la misma función que el IU. Es decir, a dos elementos de la pila del mismo tipo, pero que correspondan a cadenas de terminales distintas, les asocia números distintos.

f) En cada subrutina va un índice M, indicador de error, M=1 si no hay error. Si sí lo hay M=0.

3.2. - Explicación del Programa.

3.2.1. - Entrada y salida.

3.2.1.1. - Programa sintáctico. Como entrada tendremos los vectores R y V que nos serán proporcionados por el preprocesador (en nuestro caso se introducen como datos).

La matriz MAT(S,R) es introducida como dato fijo y no se variará en ningún momento, cualquiera que sea el programa a compilar.

Como salida se dará: Cada vez que se procesa una instrucción entera sin encontrar errores se escribe la instrucción alfanuméricamente y numéricamente.

Cada vez que se entre en una rutina se escribe: SITUACION - valor de S, valor de U, valor de V. En caso de error se escribe: Situación que produce error, es decir, valores de S, U y V que producen error y después la instrucción donde se encontró error alfanuméricamente y numéricamente.

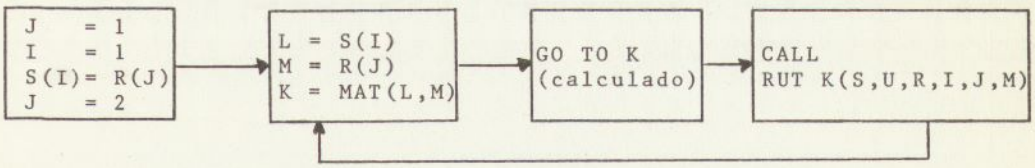
Al final de todo el proceso se da, en todo caso, un mensaje de análisis terminado.

3.2.1.2. - Salida y entrada de C-estructura.- Como entrada damos dos vectores LS t LU. Como salida se va generando un árbol de la forma siguiente:

Cada vez que se hace una reducción se escribe:

Los símbolos reducidos en línea y debajo el símbolo al que se han reducido.

3.2.2. - Explicación del programa.- El programa principal actúa de acuerdo con el siguiente organigrama:



Cada rutina actúa de la siguiente forma:

Primero mira si es compatible la situación S(I) U R(J), es decir si puede ocurrir de acuerdo con la gramática. Si no, hace M=0, y llama a la rutina de error. Si sí es compatible puede tomar las siguientes distintas acciones:

a) - Si S(I) U R(J) forman cabeza de una parte derecha de una producción se mete en el Stack en el lugar I, S(I) U R(J). Se hace U = $\emptyset$ y se hace J = J+1, avanzando al siguiente símbolo de la cadena de input. S(I) U R(J) deben poder obtenerse mediante $A \xrightarrow{*} S(I)\ U\ R(J)\ B$ tal que $B \neq \emptyset$, B cadena de símbolos.

b) - Si S(I) U R(J) corresponden a una parte derecha completa es decir $\exists A \xrightarrow{*} S(I)\ U\ R(J)$, se reduce, es decir, se hace U= a la parte izquierda de dicha producción, I = I-1, J = J+1.

c) - Si $\exists A \xrightarrow{*} U\ R(J)\ B$ con $B \neq \emptyset$ y $\exists A' \xrightarrow{*} S(I)\ U$, entonces si el último terminal de S(I) tiene precedencia sobre el símbolo de R(J), entonces se hace U = A' (a la parte izquierda de dicha producción), I = I - 1.

En caso contrario se hace I = I+1, se almacena en S(I) U R(J), se hace U = $\emptyset$ y J = J+1.

Si existe una sola de esas dos producciones se actúa como en el caso en que esta producción tiene precedencia.