[FECHA]

INVESTIGACIÓN

TABLAS DE TRANSICIÓN, BUFFER DE ENTRADA, PAREJA, CENTINELA, FIRTS, FOLLOWS Y TERMINAL

MARISOL SOLIS LÓPEZ
COMPILADORES E INTÉRPRETES
INGENIERIA EN DESARROLLO DE SOFTWARE

Tabla de transición:

Es la manera más cercana de representar los autómatas, consta de filas que representan los estados y las columnas que representan los símbolos de entrada, adicionalmente se agregan dos columnas para representar los tokens y para indicar retrocesos.Es una tabla que muestra qué estado se moverá un autómata finito dado, basándose en el estado actual y otras entradas. Una tabla de estados es esencialmente una tabla de verdad en la cual algunas de las entradas son el estado actual, y las salidas incluyen el siguiente estado, junto con otras salidas.

Una tabla de estados es una de las muchas maneras de especificar una máquina de estados, otras formas son un diagrama de estados, y una ecuación característica.Cuando se trata de un autómata finito no determinista, entonces la tabla de transición muestra todos los estados que se moverá el autómata.

Formas comunes

- *Tablas de estados de una dimensión:*
  También llamadas tablas características, las tablas de estados de una dimensión son más como tablas de verdad que como las versiones de dos dimensiones. Las entradas son normalmente colocadas a la izquierda, y separadas de las salidas, las cuales están a la derecha. Las salidas representarán el siguiente estado de la máquina.

A	B	Estado Actual	Siguiente Estado	Salida
0	0	S_1	S_2	1
0	0	S_2	S_1	0
0	1	S_1	S_2	0
0	1	S_2	S_2	1
1	0	S_1	S_1	1
1	0	S_2	S_1	1
1	1	S_1	S_1	1
1	1	S_2	S_2	0

S_1 y S_2 representarían probablemente los bits individuales 0 y 1, dado que un simple bit solo tiene dos estados.

- *Tablas de estados de dos dimensiones:*
  Las tablas de transición de estados son normalmente tablas de dos dimensiones. Hay dos formas comunes para construirlas.
  - La dimensión vertical indica los Estados Actuales, la dimensión horizontal indica eventos, y las celdas (intersecciones fila/columna) de la tabla contienen el siguiente estado si ocurre un evento (y posiblemente la acción enlazada a esta transición de estados).

Events / State	E_1	E_2	...	E_n
S_1	-	A_y/S_j	...	-
S_2	-	-	...	A_x/S_i
...	...	...	...	...
S_m	A_z/S_k	-	...	-

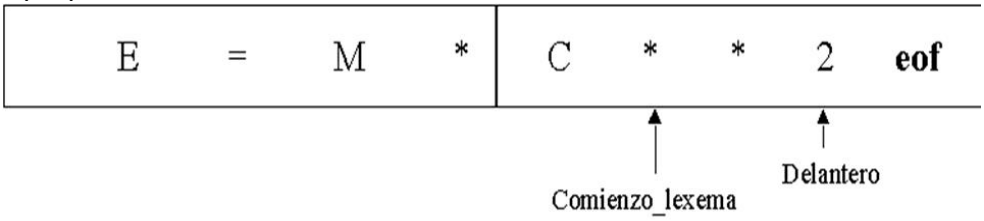
- La dimensión vertical indica los Estados Actuales, la dimensión horizontal indica los siguientes estados, y las intersecciones fila/columna contienen el evento el cual dirigirá al siguiente estado particular.

next / current	S_1	S_2	...	S_m
S_1	A_y/E_j	-	...	-
S_2	-	-	...	A_x/E_i
...	...	...	...	...
S_m	-	A_z/E_k	...	-

Buffer de entrada

Un buffer es un espacio de almacenamiento temporal de entrada o salida de datos cuando esta está en transferencia. En el análisis léxico se utiliza un buffer para realizar la compilación en menos tiempo y además para no consumir toda la memoria durante las comparaciones.

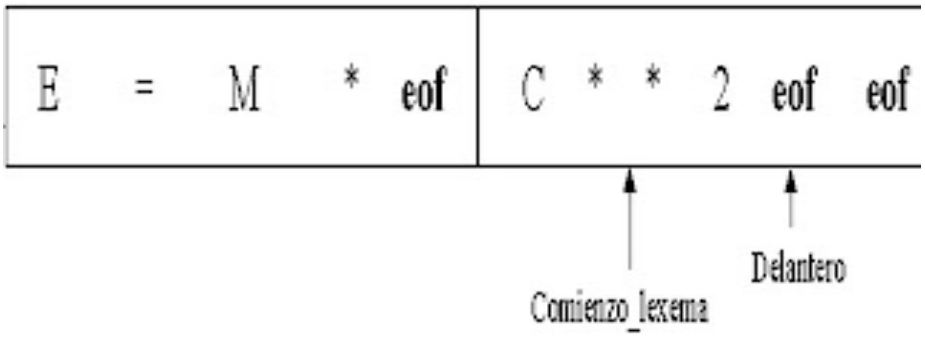
- ***Técnica de pareja del buffer:***
  Aquí se divide al buffer en 2 partes iguales, se tienen además 2 apuntadores que recorren el buffer, uno de ellos denominado delantero que va recorriendo el buffer posición por posición hasta encontrar una coincidencia con un patrón y otro llamado lexema que recorre el buffer procesando la información de los lexemas.
  La técnica consiste en que primero se llena una mitad del buffer, el apuntador delantero comienza a recorrer el buffer seguido por el apuntador lexema, hasta encontrar coincidencias con un patrón y así hasta llegar al final de la primera mitad, cuando se termina la primera mitad, se llena la segunda mitad del buffer y el apuntador es delantero y lexema se posicionan al inicio de la segunda mitad y realizan el mismo proceso que se describió anteriormente.
  Cuando se llega al final de la segunda mitad, se llena de nuevo la primera mitad y los apuntadores pasan al inicio de esta, este proceso se repite hasta encontrar el fin de archivo (eof).



"Buffer de entrada en dos mitades"

- ***Técnica del centinela:***
  El buffer se divide en 2 mitades, se usan 2 apuntadores a igual que en la técnica anterior. Funciona de manera similar al de pareja de buffer pero con una pequeña diferencia, aquí

cada que el apuntador delantero avanza se verifica que el buffer aun no termina, esto es si ha encontrado la señal de fin del buffer (eob), de ser así se llena la segunda mitad del buffer y los apuntadores pasan a la segunda mitad, con esta señal se reduce tiempo de verificar si se ha llegado al límite del buffer, utilizando 2 señales de terminación, una para cada buffer.



First:

Se define el conjunto *FIRST* de un símbolo, ***FIRST (X)*** ($X \in \{T \cup N\}$), como el **conjunto de los símbolos terminales que pueden aparecer como primer símbolo terminal** en las cadenas derivadas a partir de X.

- Si X es un terminal ($X = t$, con $t \in T$) entonces $FIRST(X) := \{t\}$.
- Si X es un no terminal ($X \in N$), para calcular $FIRST(X)$ hay que estudiar cada una de las reglas de X. Se ejecutarán los siguientes pasos hasta que no se puedan añadir más elementos al conjunto *FIRST*:
  1. Si existe la regla $X \rightarrow \lambda$, entonces añadir λ a $FIRST(X)$ (λ es la cadena vacía o elemento nulo)
  2. Para cada una de las restantes reglas gramaticales de X, $X \rightarrow Y_1\ Y_2 \ldots\ Y_k$, (donde $Y_i \in \{T \cup N\}$), se aplica el siguiente algoritmo hasta que no se pueda añadir nada nuevo al conjunto $FIRST(X)$:

```
Todos los elementos no nulos de FIRST(Y1) se añaden a FIRST(X)
if λ ∈ FIRST(Y1)
then Todos los elementos no nulos de FIRST(Y2) se añaden a FIRST(X)
     if λ ∈ FIRST(Y2)
     then Todos los elementos no nulos de FIRST(Y3) se añaden a FIRST(X)
         …y así sucesivamente…
         if λ ∈ FIRST(Yk-1)
         then Todos los elementos no nulos de FIRST(Yk) se añaden a FIRST(X)
                if λ ∈ FIRST(Yk)
                then Añadir λ a FIRST(X)
```

También se puede calcular el conjunto ***FIRST de una cadena α de símbolos*** (según lo indicado en el paso 2):

Si α es cualquier cadena de símbolos gramaticales ($\alpha \in \{T \cup N\}^*$), se define ***FIRST (α)*** como el conjunto formado por los símbolos terminales que encabezan las cadenas derivadas de α. Si $\alpha \overset{*}{\Rightarrow} \lambda$, entonces λ (la cadena vacía) también está en *FIRST* (α).

Sea $\alpha = Y_1\ Y_2 \ldots\ Y_k$, donde $Y_i \in \{T \cup N\}$. Para calcular *FIRST* (α):

```
Todos los elementos no nulos de FIRST(Y1) se añaden a FIRST(α)
if λ ∈ FIRST(Y1)
then Todos los elementos no nulos de FIRST(Y2) se añaden a FIRST(α)
     if λ ∈ FIRST(Y2)
     then Todos los elementos no nulos de FIRST(Y3) se añaden a FIRST(α)
         …y así sucesivamente…
         if λ ∈ FIRST(Yk-1)
         then Todos los elementos no nulos de FIRST(Yk) se añaden a FIRST(α)
                if λ ∈ FIRST(Yk)
                then Añadir λ a FIRST(α)
```

Follows:

Se define ***FOLLOW (A)***, para el no terminal A $(A \in N)$, como el **conjunto de terminales** t $(t \in T)$ **que pueden aparecer inmediatamente a la derecha de** ***A*** en alguna forma sentencial, es decir, el conjunto de terminales t tales que haya una derivación de la forma $S \overset{*}{\Rightarrow} \alpha A t \beta$ (siendo $\alpha, \beta \in (N \cup T)^*$). Si A puede ser el símbolo de más a la derecha en alguna forma sentencial, entonces $ está en *FOLLOW (A)*.

Para calcular *FOLLOW (A)* se aplican las siguientes reglas hasta que no se pueda añadir nada más al conjunto *FOLLOW (A)*:

1. Si A es el axioma de la gramática, añadir $ a *FOLLOW (A)*.
2. Si existe una regla $B \rightarrow \alpha A \beta$, entonces todos los elementos no nulos de *FIRST* (β) se añaden a *FOLLOW (A)*.
3. Si existe una regla $B \rightarrow \alpha A \beta$ y $\lambda \in$ *FIRST* (β) (es decir, $\beta \overset{*}{\Rightarrow} \lambda$), o bien si existe una regla $B \rightarrow \alpha A$, entonces todo lo que esté en *FOLLOW (B)* se añade a *FOLLOW (A)*.

Terminal:

Una gramática describe de forma natural la estructura jerárquica de muchas construcciones de los lenguajes de programación. Las gramáticas libres de contexto permiten describir la mayoría de los lenguajes de programación, de hecho, la síntaxis de la mayoría de lenguajes de programación está definida mediante gramáticas libres de contexto. Por otro lado, estas gramáticas son suficientemente simples como para permitir el diseño de eficientes algoritmos de análisis sintáctico que, para una cadena de caracteres dada determinen como puede ser generada desde la gramática

Consta de:

• ***TERMINALES***. Símbolos básicos con que se forman las cadenas. Para un lenguaje de programación, cada palabra clave/reservada es un terminal.

• NO TERMINALES. Son variables sintácticas que denotan conjuntos de cadenas (identificadotes o variables). Los no terminales definen conjuntos de cadenas que ayudan a definir el lenguaje generado por la gramática. Imponen una estructura jerárquica sobre el lenguaje que es útil tanto para el análisis sintáctico como para la traducción.

• UN SÍMBOLO INICIAL. En una gramática, es un no terminal que representa un conjunto de cadenas.

• PRODUCCIONES. Especifican cómo se pueden combinar los terminales y no terminales para formar cadenas. Cada producción consta de un No terminal (símbolo inicial), seguido por una flecha o símbolo de asignación, seguida por una cadena de no terminales y terminales.

Referencias

7Mo 1, C., & Perfil, V. T. M. (s. f.). *UNIDAD 3: ANÁLISIS SINTÁCTICO*. http://bloggcompliladores7mo1.blogspot.com/2010/11/unidad-3_20.html

Julián, A. E. C. J. H. E. I. (s. f.). *UNIDAD II.-Analizador léxico.* http://cursocompiladoresuaeh.blogspot.com/2010/11/unidad-ii-analizador-lexico.html

Villalta, P. A. (s. f.). *Compiladores, Analisis Lexico, Tabla de Transiciones*. https://es.slideshare.net/pavillalta/analisis-lexico-tabla-de-transiciones-p