Esta tesis doctoral contiene un índice que enlaza a cada uno de los capítulos de la misma.
Existen asimismo botones de retorno al índice al principio y final de cada uno de los capítulos.

Ir directamente al índice

Para una correcta visualización del texto es necesaria la versión de Adobe Acrobat Reader 7.0 o posteriores

Aquesta tesi doctoral conté un índex que enllaça a cadascun dels capítols. Existeixen així mateix botons de retorn a l'index al principi i final de cadascun deis capítols .

Anar directament a l'index

Per a una correcta visualització del text és necessària la versió d' Adobe Acrobat Reader 7.0 o posteriors.

UNIVERSIDAD DE ALICANTE

SINTAXIS Y CODIFICACIÓN EN MODELOS DE SISTEMAS COMPLEJOS

FRANCISCO VIVES MACIÁ

Memoria presentada para aspirar al grado de Doctor en Ciencias Matemáticas.

DEPARTAMENTO DE ANÁLISIS MATEMÁTICO
MATEMÁTICA APLICADA

YOLANDA VILLACAMPA ESTEVE, CATEDRÁTICA DEL DEPARTAMENTO DE ANÁLISIS MATEMÁTICO Y MATEMÁTICA APLICADA EN LA ESCUELA POLITÉCNICA SUPERIOR DE ALICANTE

CERTIFICA : Que D. Francisco Vives Maciá, licenciado en Ciencias Matemáticas, ha realizado bajo mi dirección la Memoria que lleva por título "Sintaxis y Codificación en Modelos de Sistemas Complejos" con el fin de que sea presentada como Tesis para aspirar al grado de doctor.

Lo que certifico para que conste según la ley vigente.

Alicante, 30 de Abril de 1999

Fdo. Yolanda Villacampa Esteve

AGRADECIMIENTOS

El presente trabajo ha sido realizado bajo la dirección de Dª Yolanda Villacampa Esteve, a quién va dirigido mi sincero agradecimiento por su confianza al proponerme el proyecto y por su trabajo incansable en la lectura, revisión, crítica, consejos e ideas aportadas durante todo el proceso de elaboración.

También quiero manifestar mi agradecimiento a todas aquellas personas que con su constante apoyo han hecho posible esta memoria.

INDICE

3. Gramáticas generativas y recognoscitivas en sistemas complejos

CAPÍTULO 1

IDEAS GENERALES

CAPÍTULO 1

IDEAS GENERALES

1.1. INTRODUCCIÓN

La introducción de las matemáticas en el estudio de las ciencias es un proceso que ha ido en aumento a través de los tiempos, sobre todo en el estudio de los sistemas naturales.

De forma general, el estudio de la estructura de un sistema complejo, así como su modelización y evolución en el tiempo presenta una gran dificultad debido a la complejidad de los sistemas.

El estudio de la estructura de un sistema complejo se ha realizado en primer lugar desde el estudio de diversas teorías en el contexto general de la teoría de los sistemas. Podemos destacar la teoría general de sistemas de Mesarovic y Takahara (1978) en (41).

Asimismo, en el planteamiento de un modelo de sistemas y la estabilidad de sistemas entrada-salida, podemos resaltar los estudios realizados en (37) por Lin Y. (1987). Analogías y estudios entre sistemas así como diversos conceptos de sistemas son reflejados por Lin Y. y Ma. Y. (1987, 1989) en (37) y (38) y Yang Z.B. (1989) en (60). Estudios concretos para sistemas vivos son llevados a cabo por Miller (1978) en (42) y diversas axiomatizaciones de los mismos son realizadas por Villacampa et al. (1997, 1999) en (57) y (58).

En la modelización de sistemas complejos se da, usualmente, mucha importancia a la metodología, ya que toda la ayuda computacional que podemos emplear en las modelizaciones está totalmente condicionada a ella. Así como cualquier metodología está condicionada por la teoría de sistemas en la que se basa.

Las modelizaciones de sistemas complejos son realizadas a través de la combinación de diversas metodologías, entre las que se pueden citar las de Forrester (1961,1966) en (18) y (19), Zeigler (1984) en (61), Jorgensen (1988) en (32) y Zhan (1990) en (62).

Sea cuál fuese la metodología utilizada en el estudio y modelización de sistemas complejos, esta debe describir los métodos para conseguir expresar determinados procesos en un lenguaje.

Son diversos los autores que han resaltado la importancia del estudio de los sistemas y sus modelizaciones desde el punto de vista de la matemática lingüística. Así, Patten (1997) en (46) señala que "todos los modelos tienen en común que encierran experiencia y siempre envuelven signos, señales, sintaxis, semántica y una habilidad para descifrar y obtener significados".

Un estudio reciente, desde el punto de vista de la matemática lingüística de sistemas semióticos ha sido realizado por Villacampa et al. (1999) en (58).

1.2. LINGÜÍSTICA MATEMÁTICA Y MODELIZACIÓN.

Supongamos un sistema dinámico modelizado por un conjunto de ecuaciones diferenciales ordinarias $\{y_i'\}_{i=1}^{n}$, llamadas ecuaciones de estado. Si $y_i' = g(x_{i1}, x_{i2}, ..., x_{in})$ a las variables $\{x_{ij}\}_j$ se les llaman variables de flujo. A su vez, cada variable de flujo depende de un conjunto de variables, es decir, $x_{ij} = F(\varphi_1, \varphi_2, ..., \varphi_n)$. A la ecuación $F(\varphi_1, \varphi_2, ..., \varphi_n)$, que expresa la dependencia de una variable de flujo respecto a estas otras variables del sistema se le llama ecuación de flujo.

En el estudio de un sistema complejo, como son los sistemas ecológicos, las variables de flujo, x_{ij}, y las variables de las que dependen, $\varphi_1, \varphi_2, ..., \varphi_n$, deberán ser modelizadas matemáticamente disponiendo de datos experimentales de las mismas.

Es, pues, necesario obtener expresiones analíticas que sirvan para este tipo de sistemas cuya estructura es muy compleja, así como la obtención de relaciones entre sus diversos elementos.

Estas expresiones analíticas que sirven para expresar matemáticamente estas variables pueden conseguirse, a partir de datos experimentales, de forma computacional mediante gramáticas que construiremos a lo largo del presente proyecto.

En este proyecto se realiza un estudio de la teoría de modelos de sistemas complejos desde el punto de vista de la matemática lingüística, la teoría de autómatas finitos, lenguajes regulares y la teoría de la codificación.

El estudio de la teoría de los sistemas complejos, como los ecológicos, agronómicos, socioeconómicos, etc. se analiza desde el punto de vista de la lingüística matemática, ya que cualquier metodología está asociada a un lenguaje y este a sus componentes sintácticos y semánticos.

Por otra parte, nos adentramos en la teoría de autómatas y más concretamente en los reconocedores de lenguajes, ya que se usan frecuentemente en los problemas que implican el análisis de cadenas de caracteres. De esta forma al construir un lenguaje, podremos construir un reconocedor finito del mismo. Estos estudios y los resultados obtenidos harán posible la construcción de lenguajes que pueden modelizar sistemas complejos, siendo posible la computerización de los mismos.

El estudio desde el punto de vista de la lingüística matemática y la teoría de autómatas se va a desarrollar de la siguiente manera:

El capítulo 2 comienza con la definición de autómata finito. La definición que se ofrece corresponde al modelo de máquina de Mealy, máquina en la que las salidas están asociadas a las transiciones. A continuación se expone el concepto de máquina de Moore y la equivalencia entre estos dos modelos de autómatas, por la comodidad que supone el trabajar con una máquina cuyas salidas están asociadas a los estados. Es por ello por lo que hemos desarrollado un programa que obtiene el autómata de Moore equivalente a un autómata de Mealy dado.

También estudiamos los autómatas como dispositivos que producen cadenas de símbolos a su salida en respuesta a cadenas de símbolos presentadas a la entrada, como introducción a la explicación del concepto de reconocedor finito y los lenguajes aceptados por reconocedores finitos.

Recordemos, también, que un autómata finito es, él mismo, un modelo de un procedimiento para reconocimiento de cadenas por medio de la expresión regular asociada y es sencillo traducir el autómata finito a un código en un lenguaje de programación.

Terminamos la primera parte del capítulo con la definición de expresión regular, como forma de expresar el lenguaje que puede ser reconocido por un reconocedor finito y el concepto de derivada de una expresión regular para tener un procedimiento que nos permitirá obtener el reconocedor finito de dicho lenguaje.

La segunda parte del capítulo está dedicada al estudio de los diferentes tipos de gramáticas y los lenguajes generados por las mismas para llegar a la conclusión de que los lenguajes generados por las gramáticas de tipo 3, los lenguajes regulares y los que pueden ser reconocidos por los reconocedores finitos coinciden. Hemos desarrollado lenguajes regulares por su doble importancia. Desde el punto de vista práctico porque pueden ser usados para especificar la construcción de analizadores léxicos (programas que analizan texto y extraen los lexemas que hay en el mismo). Desde el punto de vista teórico porque constituyen el menor conjunto de lenguajes sobre un alfabeto que es cerrado respecto a la

concatenación de cadenas, la cerradura de Kleene y la unión de lenguajes y además contiene el lenguaje vacío y los lenguajes unitarios.

Es este, pues, un capítulo teórico cuyo único objetivo es exponer todos los conceptos que se van a utilizar en los capítulos siguientes. Se han omitido ciertas demostraciones, ya que se salen de los objetivos que nos hemos marcado, aunque indicamos que pueden encontrarse en textos como el (28) de J. Hopcroft y J. Ullman (1979), el (34)de Dean Kelley (1998) o el (16) de G. Fernández y F. Sáez Vacas (1995).

En el capítulo 3 se analiza la construcción de gramáticas generativas y recognoscitivas que generan lenguajes que pueden ser utilizados en la modelización matemática de sistemas complejos.

Los primeros estudios que se hicieron sobre gramáticas generativas y recognoscitivas con aplicaciones se publican en el artículo (56), "Generative and recognoscitive grammars of Ecological Models with applications (Villacampa et al., 1999). En él son consideradas y aplicadas, al caso de modelización ecológica, gramáticas que generan un lenguaje y una selección de ecuaciones que modelizan unos procesos ecológicos. Resultan ser, además, un caso particular de los estudios más amplios y generales que se realizan en este capítulo 3. No obstante cuando se utilicen conceptos, resultados o ejemplos que hayan sido publicados en él lo reflejaremos convenientemente aludiendo a Villacampa et al., 1999.

Comienza el capítulo 3 con la introducción genérica de los sistemas a los que se van a aplicar los lenguajes. Se consideran los vocabularios de primer, segundo, ..., n-ésimo orden y se define el t-léxico a partir de los vocabularios

asociados a un conjunto de variables. A continuación se obtiene el léxico diferencial, d-léxico que junto con el t-léxico constituyen, de forma general, el léxico de un sistema complejo.

Es definida una relación de equivalencia en el conjunto de palabras escritas en un determinado vocabulario, llamada relación de similitud que divide al conjunto de palabras en clases de equivalencia, llamándose sinónimas aquellas palabras que pertenecen a una misma clase.

Se continua con la definición de gramática generativa de las funciones transformadas, comenzando por las de primer orden y llegando a generalizarlas para un cierto orden n. Estas gramáticas generan lenguajes que forman el vocabulario de las gramáticas generativas de las ecuaciones de flujo.

Las gramáticas consideradas originan lenguajes regulares pudiéndose encontrar, en cada caso un reconocedor finito del mismo.

Al obtener lenguajes que modelizan las ecuaciones de flujo, es necesario seleccionar una o más ecuaciones de acuerdo con ciertos criterios establecidos por el modelizador. En este proyecto se definen algunas gramáticas cuya misión es la de realizar esta selección, son las gramáticas recognoscitivas de las ecuaciones de flujo.

Termina el capítulo definiendo las gramáticas generativas y recognoscitivas de las ecuaciones de estado o ecuaciones que modelizan los procesos a estudiar.

La codificación de las ecuaciones de flujo obtenidas en los lenguajes desarrollados en el capítulo anterior para la modelización de los sistemas complejos se realiza en el capítulo 4.

En la modelización matemática, el modelizador deberá elegir, en cada caso, las ecuaciones que desee almacenar para su posterior utilización. Por este motivo, en el capítulo 3 se han desarrollado gramáticas recognoscitivas que permiten seleccionar un cierto tipo de ecuaciones matemáticas de acuerdo con ciertos criterios de recognoscibilidad, pudiendo ser estos muy diversos. Así pues, resulta interesante paliar, en la medida de lo posible los problemas de almacenamiento de ecuaciones para su posterior manipulación. En el capítulo 4 abordamos inicialmente los conceptos generales de la teoría de códigos, introduciendo aquellos conceptos y resultados que son interesantes para su posterior utilización.

Comenzamos el capítulo con el establecimiento de códigos para la codificación de las funciones transformadas, que formarán la base sobre la que se fundamentan los códigos de las ecuaciones de flujo.

Como es lógico suponer, son inmumerables los códigos que se pueden establecer. Por ello hemos optado por tres tipos que hemos denominado CLCT (código de longitud constante), CBCT (código compacto) y un tercero CGT (código Gödel) a título de curiosidad y como prueba de los inmumerables tipos de códigos que se pueden diseñar.

Se sigue, para terminar el tema, con la codificación de las ecuaciones de flujo, que nos permite almacenar de forma sencilla las ecuaciones, pudiendo,

Volver al índice/Tornar a l'índex

como es lógico, decodificarlas posteriormente, para expresarlas en su forma inicial, tal y como se obtuvieron en su proceso de modelización. Resaltamos la forma matricial de su representación.

Finalizamos este proyecto con la exposición de una aplicación concreta de los resultados obtenidos en los capítulos precedentes, algunos de los cuáles se encuentran publicados (artículo (56),Villacampa et al., 1999).

La aplicación consiste en el desarrollo de gramáticas que sirven para modelizar el Submodelo Reproductivo del Modelo Mariola. Hemos escogido un conjunto de p-símbolos y un conjunto de variables de flujo que nos han permitido definir vocabularios de primer y segundo orden para formar el t-léxico y así definir la gramática generativa de las funciones transformadas y las gramáticas generativa y recognoscitiva de las ecuaciones de flujo, utilizando el criterio de recognoscibilidad GRF_1 mediante el cuál hemos seleccionado siete ecuaciones de flujo que posteriormente hemos codificado para su almacenamiento. Finalmente, siguiendo el proceso de recognoscibilidad GRS_1 hemos formado las ecuaciones de estado que modelizan los procesos y cuya integración por métodos numéricos nos ha permitido realizar su validación, obteniendo los resultados que en el proyecto se exponen.

CAPÍTULO 2

AUTÓMATAS

Y

LENGUAJES

CAPÍTULO 2

AUTÓMATAS Y LENGUAJES

2.1. INTRODUCCIÓN.

Este capítulo se divide en dos partes. En la primera se expone de forma general la teoría de autómatas finitos y en la segunda se realiza un estudio de las gramáticas. Se presenta en primer lugar el modelo de autómata denominado ***Máquina de Mealy***, seguido del estudio que lleva a la conclusión de que a toda máquina de Mealy se le puede asociar una ***Máquina de Moore*** equivalente, con las ventajas que supone el disponer de un autómata en el que a cada estado se le puede asociar una salida y sólo una. Se concluye esta primera parte del tema ofreciendo el listado en Pascal de un programa informático que obtiene el automáta de Moore equivalente a un determinado autómata de Mealy dado.

A continuación se ofrece otra definición de autómata, a partir de la cuál, se estudian de forma matemática, los comportamientos **entrada-salida** y **entrada-estados** del autómata y obtener el monoide de un autómata cuya tabla nos permite ver rápidamente el comportamiento del autómata para cualquier cadena que se presente en su entrada. Se sigue con el estudio de la minimización de un autómata finito.

Termina la primera parte del tema con el estudio de un tipo particular de autómatas, los ***reconocedores finitos*** de lenguajes y las condiciones que debe reunir un lenguaje para ser aceptado por un reconocedor, con el consiguiente estudio sobre conjuntos y expresiones regulares, gramáticas y su clasificación y las propiedades de los lenguajes formales.

Toda esta teoría será utilizada posteriormente para construir lenguajes que sirvan para la modelización de sistemas complejos, lo que conlleva a la computerización de la modelización.

En la segunda parte del capítulo se van a considerar los conceptos de gramática, lenguajes generados por una gramática y gramáticas equivalentes. Veremos la clasificación de las gramáticas según las restricciones impuestas a sus reglas de producción y como esta clasificación da lugar a una jerarquía de lenguajes. A continuación veremos la manera de modificar una gramática para que el lenguaje contenga la cadena vacía sin cambiar de tipo.

En la última parte del capítulo planteamos el problema del reconocimiento de los lenguajes regulares desde el punto de vista material, ya que estudiamos la relación entre el lenguaje y la máquina capaz de reconocerlo. Veremos que la clase de lenguajes que puede ser reconocido por un autómata finito es precisamente la clase de lenguajes que pueden ser generados por una gramática regular.

Todos los conceptos desarrollados en el presente capítulo constituyen la base teórica y, por tanto, su conocimiento es imprescindible para poder abordar en el tema siguiente la construcción de gramáticas generativas de ecuaciones de flujo de los sistemas estructurales complejos y sus reconocedores correspondientes.

2.2. PRIMEROS CONCEPTOS.

La palabra "**autómata**", en el lenguaje ordinario, normalmente evoca algo que pretende imitar funciones propias de los seres vivos. Por ejemplo un robot dotado de capacidades autónomas de movimiento que le permite ejecutar las órdenes o seguir el programa establecido por un ser inteligente.

En el campo de la Informática lo fundamental no es la simulación de movimiento, sino la simulación de procesos que tratan **información**, y el ejemplo típico es el ordenador, que no es más que un dispositivo que maneja símbolos.

De acuerdo con esto, se entiende que este tratamiento o procesamiento de la información sólo tiene sentido para nosotros, que decidimos si una determinada cadena de símbolos significa tal cosa, es decir, que **codificamos la información** en cadenas de símbolos; el ordenador se limita a manipular esas cadenas, dando lugar a otras cadenas que nosotros decodificaremos.

Hablando en términos generales, podemos considerar a un autómata como un dispositivo que maneja cadenas de símbolos que se le presentan a su entrada, produciendo otras cadenas de símbolos como salida. Un ordenador es un ejemplo de autómata, pero también son autómatas dispositivos más sencillos, como sumadores, contadores, etc.. También pueden estudiarse como autómatas determinadas funciones de los seres vivos, e incluso **complejos sistemas ecológicos y socioeconómicos**.

Un autómata recibe los símbolos de entrada distribuidos en el tiempo, es decir, secuencialmente y, en general, el símbolo que nos presenta en su salida en un determinado instante no sólo depende del último símbolo recibido en la entrada, sino de toda la secuencia de símbolos que ha recibido hasta ese instante.

Así pues, un determinado símbolo de entrada puede producir en la salida del autómata diferentes símbolos de salida, dependiendo de la "**historia**" o secuencia de todos los símbolos de entrada anteriores.

Lo que acabamos de mencionar nos lleva a definir el concepto de estado de un autómata. **El estado de un autómata es toda la información necesaria en un momento dado para poder deducir, dando un símbolo de entrada en ese momento, cuál será el símbolo de salida.** Un autómata poseerá un determinado número de estados y se encontrará en uno u otro según sea la historia de símbolos que le han llegado; si encontrándose en un estado determinado, recibe un símbolo

también determinado, producirá un símbolo de salida y efectuará un cambio o transición a otro estado.

Un campo importante dentro de la Informática, **y en el que nos vamos a introducir en este trabajo**, está constituido por el estudio de los **lenguajes** y las **gramáticas** que los generan. Los elementos de un lenguaje son sentencias, palabras, etc.., formados a partir de un **alfabeto.** Establecidas unas reglas gramaticales, una cadena de símbolos pertenecerá al correspondiente lenguaje si tal cadena se ha formado obedeciendo esas reglas. Puede entonces pensarse en la posibilidad de construir un **autómata reconocedor** de ese lenguaje, tal que cuando reciba a su entrada una determinada secuencia de símbolos nos presente en su salida una determinada señal que nos indique si la secuencia es o no correcta.

2.3. AUTÓMATAS FINITOS.

2.3.1. DEFINICIÓN.

Un autómata puede definirse como una quíntupla:

$$A = \langle E, S, Q, f, g \rangle$$

donde:

E = conjunto finito símbolos de entradas o alfabeto de entrada.

S = conjunto finito de símbolos de salida o alfabeto de salida.

Q = conjunto de estados.

f = es una función, llamada función de transición o función de estado siguiente:

$$f : E \times Q \rightarrow Q$$

g = es una función llamada función de salida:

$$g : E \times Q \rightarrow S$$

Esta definición puede interpretarse como la descripción matemática de una máquina que, si en el instante t recibe una entrada $e \in E$ y se encuentra en el estado

$q \in Q$, entonces proporciona una salida $g(e, q)$, y pasa al estado $f(e, q)$ en el instante $t+1$. (Suponemos una escala discreta de tiempos $t = 1, 2, 3, ..., n, ...$).

Las funciones f y g pueden representarse mediante una tabla con tantas filas como estados y tantas columnas como entradas. Si la fila i corresponde al estado q_i y la columna j corresponde a la entrada e_j, en la intersección de ambas se escribirá q_{ij}/s_{ij}, donde $q_{ij} = f(e_j,q_i)$ y $s_{ij} = g(e_j,q_i)$. (Figura 2.1)

$q \setminus e$	e_1	.....	e_j	.....
q_1	q_{11}/s_{11}	.....	q_{1j}/s_{1j}	.....
q_2	q_{21}/s_{21}	.....	q_{2j}/s_{2j}	.....
.....	.....	.....	.....	.....
q_i	q_{i1}/s_{i1}	.....	q_{ij}/s_{ij}	.....
.....	.....	.....	.....	.....

figura 2.1

Otra forma de representar las funciones f y g es mediante un grafo orientado en el que cada nodo corresponde a un estado, y si $f(e_j, q_i) = q_k$ y $g(e_j, q_i) = s_k$ existe un arco dirigido del nodo correspondiente a q_i al correspondiente a q_k, sobre el que se pone la etiqueta e_j / s_k. Este grafo suele llamarse diagrama de transiciones o diagrama de Moore y puede verse en la figura 2.2.

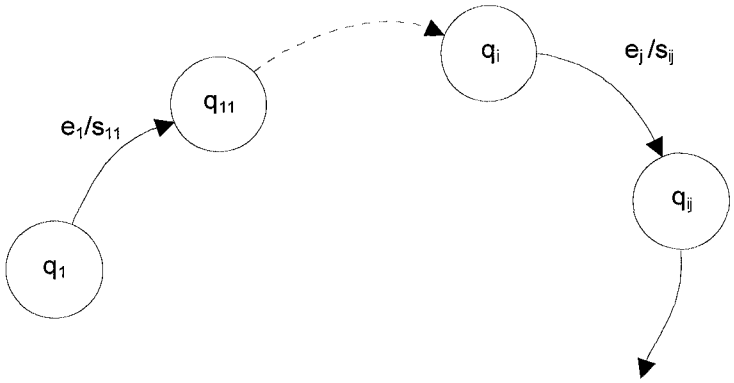


figura 2.2

2.3.2. AUTÓMATA DE MEALY Y AUTÓMATA DE MOORE.

El modelo de autómata que hemos presentado, realmente, se denomina ***Máquina de Mealy***. Las funciones g y f determinan la salida y el estado siguiente

cuando la máquina se encuentra en el estado $q \in Q$ y recibe una entrada $e \in E$. Ahora bien, es interesante considerar, además de los símbolos de E, un nuevo elemento, λ (elemento neutro); físicamente, el decir que la entrada es λ, es lo mismo que decir que no hay ninguna entrada.

Ampliando el dominio de f, que es ExQ, a $\{E \cup \{\lambda\}\}\ x\ Q$ y lo mismo, el dominio de g podemos responder a la pregunta: "¿ qué ocurre si, estando el autómata en el estado $q \in Q$, recibe como entrada λ? ".

La ampliación del dominio de f no plantea ningún problema, ya que podemos convenir que $f(\lambda,q) = q$ (es decir, físicamente, si no hay entrada no se cambia de estado). Pero no ocurre lo mismo con g ; y ello se ve fácilmente con el ejemplo anterior: si llegamos a q_1 ya sea de q_3 (por la entrada de b) o de q_1 (por la entrada de a) la salida es 0, por lo que podemos asociar la salida 0 al estado q_1 y decir que $g(\lambda,q_1) = 0$; sin embargo no podemos definir $g(\lambda,q_2)$, ya que si llegamos a q_2 desde q_1 la salida es 1, pero si llegamos desde el propio q_2 la salida es 0. Es evidente, que en general sólo puede definirse $g(\lambda,q)$ en el caso de que

$$[q = f(e_1,q_1) = f(e_2,q_2)] \rightarrow [g(e_1,q_1) = g(e_2,q_2)] \qquad (2.1)$$

es decir, que a q se le puede asociar una salida y sólo una.

Si esto ocurre para todo $q \in Q$ podemos definir una función inyectiva

$$h = Q \rightarrow S$$

tal que $g(e,q) = h[f(e,q)]$, $e \in \{E \cup \{\lambda\}\}$, $q \in Q$. En este caso, podemos decir que la salida solo depende del estado, y el autómata se llama máquina de Moore.

Expresando el tiempo de forma explícita

$$s(t) = g[e(t), q(t)] = h[q(t)] = h[f[e(t-1), q[t-1]]] \qquad (2.2)$$

En una máquina de Mealy las salidas están asociadas a las transiciones, mientras que en una máquina de Moore las salidas están asociadas a los estados. Puesto que toda máquina de Moore es una máquina de Mealy que cumple la condición (2.1) para todos los estados, parece en principio que las primeras son un subconjunto de las segundas. Sin embargo, se demuestra (Fundamentos de Informática. Gregorio Fernández, 1987) que siempre podemos encontrar una máquina de Moore equivalente a una máquina de Mealy dada.

Teorema: " Dada una máquina de Mealy, siempre se puede encontrar una máquina de Moore equivalente".

Debido a que nos parece más cómodo trabajar con máquinas de Moore hemos desarrollado un programa en Pascal que obtiene el autómata de Moore equivalente a un determinado autómata de Mealy dado y que a continuación se representa su listado.

2.3.3. PROGRAMA DE OBTENCIÓN DE UN AUTÓMATA DE MOORE EQUIVALENTE A UN AUTÓMATA DE MEALY.

```
Program moore;
uses crt;
type A=array[1..5] of char;
     A2=array[1..5] of integer;
     B=array[1..10,1..10] of integer;
     C=array[1..2] of integer;
var
m,n,i,j,k,x,y,fila,columna,numest,nument,numsal,contador:integer;
    entrada:A;
    salida:A2;
    f,g,ultima2:B;
    estado,ultima:array[1..10,1..10] of C;
    estfinal:array[1..10] of C;
    final:array[1..10]of integer;

(*******************************************************)
Procedure entradas_salidas;
begin
  writeln('Introduzca los identificadores de las entradas ');
  for n:=1 to nument do
    begin
      write('Entrada número ',n,' : ');
      readln(entrada[n]);
    end;
  writeln;
  writeln('Introduzca los identificadores de las salidas');
  for n:=1 to numsal do
    begin
      write('Salida número ',n,' : ');
      readln(salida[n]);
    end;
end;
(*******************************************************)
Procedure funcion_f;
begin
for n:=1 to numest do
  for m:=1 to nument do
    begin
```

```
      write('f(q',n,',',entrada[m],') = ');
      readln(f[n,m]);
    end;
end;
(*******************************************************)
Procedure funcion_g;
begin
for n:=1 to numest do
  for m:=1 to nument do
    begin
      write('g(q',n,',',entrada[m],') = ');
      readln(g[n,m]);
    end;
end;
(*******************************************************)
Procedure fusionar;
begin
  for n:=1 to numest do
    for m:=1 to nument do
      begin
       estado[n,m][1]:=f[n,m];
       estado[n,m][2]:=g[n,m];
      end;
end;
(*******************************************************)
Procedure estadosfinales;
begin
contador:=0;
  for j:=1 to numest do
   for i:=1 to numsal do
    for n:=1 to numest do
     for m:=1 to nument do
       if (estado[n,m][1]=j) and (estado[n,m][2]=salida[i]) then
          begin
 if (estfinal[contador][1]<>j) or (estfinal[contador][2]<>salida[i])
          then
           begin
           contador:=contador+1;
            estfinal[contador][1]:=j;
            estfinal[contador][2]:=salida[i];
            final[contador]:=contador;
           end;
          end;
end;
(*******************************************************)
procedure ultimo;
begin
```

```
  for i:=1 to contador do
    begin
      x:=estfinal[i][1];
      for m:=1 to nument do
        begin
          ultima[i,m][1]:=estado[x,m][1];
          ultima[i,m][2]:=estado[x,m][2];
        end;
    end;
  for i:=1 to contador do
    for m:=1 to nument do
      begin
         x:=ultima[i,m][1];
         y:=ultima[i,m][2];
            for j:=1 to contador do
               if(estfinal[j][1]=x) and (estfinal[j][2]=y)
                    then
                       ultima2[i,m]:=j;
       end;
end;
(*****************************************************)
begin
  clrscr;
  write('Número de estados del autómata de Mealy.......: ');
  readln(numest);
  write('Número de entradas del autómata ..............: ');
  readln(nument);
  write('Número de salidas del autómata ...............: ');
  readln(numsal);
  writeln;
  entradas_salidas;
  writeln;
  writeln('Función de transición de estados ');
  funcion_f;
  writeln;
  writeln('Función de salida ');
  funcion_g;
  fusionar;
  estadosfinales;
  ultimo;
  writeln;
  writeln('Tabla del autómata de Moore equivalente');
  writeln('=======================================');
  write('        Entradas -->    ');
  for m:=1 to nument do
     write(entrada[m],'        ');
     writeln;
  writeln('Estados/salidas');
```

```
  writeln('-------------------------------------------------------');
  for n:=1 to contador do
     begin
       write('    q',n,'/',estfinal[n][2],'                ');
       for m:=1 to nument do
         write('q',ultima2[n,m],'      ');
       writeln;
     end;
  readln
end.
```

Ejemplo de ejecución del programa moore.

Sea la máquina de Mealy dada mediante el diagrama de la figura 2.3.

Obtener la máquina de Moore equivalente.

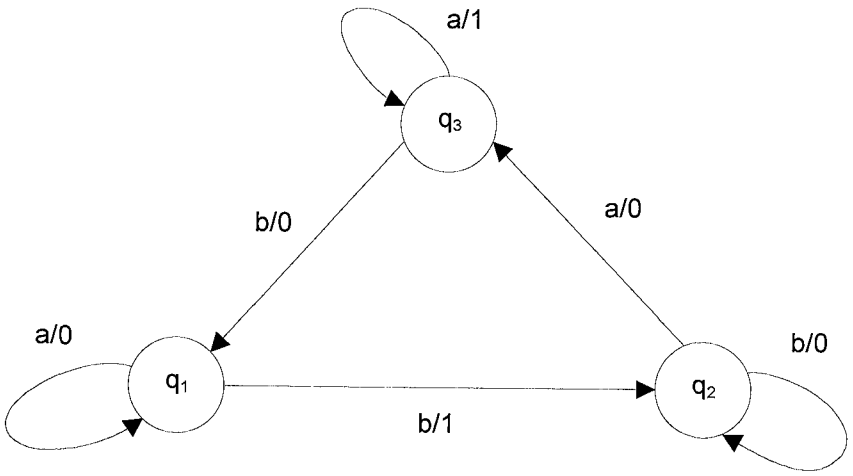


figura 2.3

```
Número de estados del autómata de Mealy .....: 3
Número de entradas del autómata .....: 2
Número de salidas del autómata .....: 2

Introduzca los identificadores de las entradas
Entrada número 1: a
Entrada número 2: b

Introduzca los identificadores de las salidas
Salida número 1: 0
Salida número 2: 1

Función de transición de estados
f(q1,a) = 1
f(q1,b) = 2
f(q2,a) = 3
f(q2,b) = 2
f(q3,a) = 3
f(q3,b) = 1

Función de salida
g(q1,a) = 0
g(q1,b) = 1
g(q2,a) = 0
```

```
g(q2,b) = 0
g(q3,a) = 1
g(q3,b) = 0
```

Tabla del autómata de Moore equivalente

Entradas -->	a	b
Estados/salidas		
q1/0	q1	q3
q2/0	q4	q2
q3/1	q4	q2
q4/0	q5	q1
q5/1	q5	q1

El diagrama correspondiente puede verse en la figura 2.4.

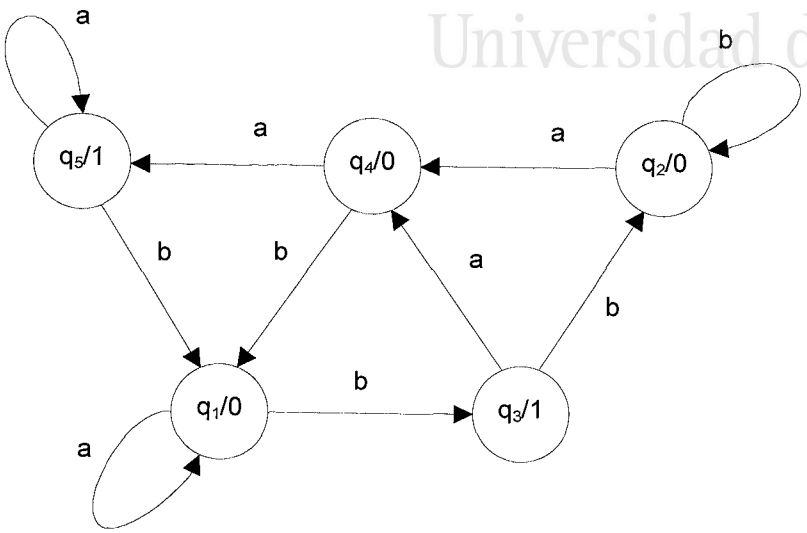


figura 2.4

2.4. OTRA DEFINICIÓN DE AUTÓMATA.

2.4.1. DEFINICIÓN

Al ser los autómatas unos dispositivos que producen cadenas de símbolos a la salida en respuesta a cadenas de símbolos presentadas a la entrada, podemos definir un autómata como una función

$$F^* : E^* \rightarrow S^*$$

que hace corresponder a cada cadena de entrada $x \in E^*$ una cadena de salida $F^*(x) = y \in S^*$.

Pero el autómata también queda perfectamente definido restringiendo el rango

de la función de S^* a S, es decir, podemos definir un autómata como una función

$$F^* : E^* \rightarrow S$$

que hace corresponder a cada cadena de entrada $x \in E^*$, el último símbolo obtenido como salida, $F(x) = s \in S$, ya que si $x = e_0e_1.....e_{n-1}$ tendremos como símbolos de salida

$F(e_0)$ *en el instante 1*

$F(e_0e_1)$ *en el instante 2*

.....

$F(e_0e_1.....e_{n-1})$ *en el instante n*

por consiguiente

$$F^*(x) = F^*(e_0e_1.....e_{n-1}) = F(e_0)F(e_0e_1).....F(e_0e_1.....e_{n-1})$$

lo que demuestra que F^* queda determinada conociendo F.

2.4.2. COMPORTAMIENTO ENTRADA-SALIDA DEL AUTÓMATA

En un autómata $A = \langle E, S, Q, f, g \rangle$ se define el comportamiento de entrada-salida de A inicializado en el estado q por la función

$$C_q : E^* \rightarrow S$$

que aplica a cada entrada $x \in E^*$ una salida $s = g(x,q)$.

Es decir, si en el instante t_0 el autómata se encuentra en el estado q e introducimos la cadena $x = e_1e_2.....e_n$ se obtendrán las salidas

$C_q(e_1)$ en $t = t_0$

$C_q(e_1e_2)$ en $t = t_0+1$

$C_q(e_1e_2e_3)$ en $t = t_0+2$

.....

$C_q(e_1e_2.....e_n)$ en $t = t_0+n-1$

Vemos así que para todo autómata pueden definirse tantas funciones como estados tiene el autómata.

La definición del comportamiento entrada-salida nos permite dar las siguientes definiciones:

1) ***Estados equivalentes:*** Dados dos autómatas con los mismos alfabetos de entrada y salida

$$A_1 = \langle E, S, Q_1, f_1, g_1 \rangle \qquad y \qquad A_2 = \langle E, S, Q_2, f_2, g_2 \rangle$$

$q_1 \in Q_1$ es equivalente a $q_2 \in Q_2$ si $C_{q1} = C_{q2}$.

Esta misma definición sirve para estados equivalentes dentro del mismo autómata: basta considerar $Q_1 = Q_2$, $f_1 = f_2$ y $g_1 = g_2$.

2) ***Forma mínima de un autómata:*** Un autómata está en forma mínima si

$$(C_{q1} = C_{q2}) \rightarrow (q_1 = q_2)$$

es decir, si no existen estados equivalentes.

Para comprobar que un autómata está en forma mínima hay que ver si el comportamiento es distinto en cada estado. Esto no es fácil, puesto que habría que ir ensayando con diferentes cadenas de entrada hasta que se observe una diferencia de comportamiento entre dos estados para la misma cadena. pero como E^* es infinito, si no encontramos tal diferencia después de un número finito de cadenas no podemos garantizar que el autómata finito esté en forma mínima. En el texto referenciado anteriormente, Fundamentos de Informática de Gregorio Fernández se ofrece una demostración y un algoritmo que nos permitirá ver si un autómata está en forma mínima.

2.5. CONCEPTOS ALGEBRAICOS.

2.5.1. INTRODUCCIÓN.

A continuación vamos a recordar algunos conceptos de álgebra que se utilizan en el proceso de minimización de un autómata finito.

2.5.2. MONOIDE DE TRANSFORMACIONES DE UN CONJUNTO.

Una **transformación** t en un conjunto C se define como una función de C en sí mismo: $t : C \rightarrow C$.

Teorema: Sea C un conjunto cualquiera y sea $C^C = \{ t: C \to C \}$ el conjunto de todas las transformaciones de C en C. Entonces $\langle C^C, o \rangle$, donde *"o"* representa la composición de funciones, es un monoide, llamado ***monoide de transformaciones*** de C.

2.5.3. HOMOMORFISMO ENTRE MONOIDES.

Si $\langle S_1, * \rangle$ y $\langle S_2, \bullet \rangle$ son dos semigrupos, una función $f: S_1 \to S_2$ se dice que es un homomorfismo entre ambos semigrupos si

$$(\forall a,b \in S_1) \qquad [f(a*b)c = f(a) \bullet f(b)]$$

Si f es biyectiva el homomorfismo recibe el nombre de isomorfismo.

Si S_1 es un monoide con elemento neutro e_1, S_2 un monoide con elemento neutro e_2, f un homomorfismo (isomorfismo) entre S_1 y S_2 y se cumple que $f(e_1) = e_2$ entonces f es un ***homomorfismo (isomorfismo) monoide.***

2.5.4. MONOIDE LIBRE Y HOMOMORFISMO.

Sea E un alfabeto, E^* es el conjunto de todas las cadenas, incluida λ, que pueden formarse a partir de E.

La concatenación de dos cadenas $x, y \in E^*$ es una ley de composición interna sobre E^*, es decir, una aplicación $E^* \times E^* \to E^*$ que consiste en formar la cadena xy

poniendo x delante de y.

La concatenación satisface las siguientes propiedades:

a) Es asociativa: $x(yz) = (xy)z = xyz$

b) No es conmutativa, en general: $xy \neq yx$

c) Tiene elemento neutro, λ: $\lambda x = x\lambda = x$.

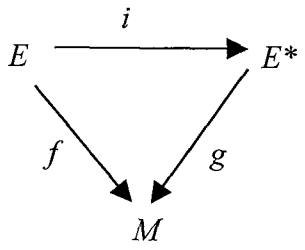
d) Ninguna cadena tiene inverso, salvo λ.

Por consiguiente $\langle E^*, \ \rangle$, es decir, E^* con la operación de concatenación es una estructura algebraica de tipo monoide, a la que se llama ***monoide libre*** generado por E.

Sea, ahora, i: $E \rightarrow E^*$ la función que aplica todo elemento de E en la correspondiente cadena de longitud unidad, es decir, $\forall a \in E,\ i(a) = a$ y sea f cualquier función de E en el conjunto de cualquier monoide $\langle M, * \rangle$. Entonces existe un único homomorfismo monoide

$$g: \langle E^*, \ \rangle \rightarrow \langle M, * \rangle$$

tal que $g \ o \ i = f$, es decir, tal que el diagrama



es conmutativo.

Este teorema nos permite extender el dominio de cualquier función $E \rightarrow M$ del alfabeto E en el conjunto de un monoide $\langle M, * \rangle$ a un homomorfismo monoide $\langle E^*, \rangle \rightarrow \langle M, * \rangle$, y nos será de utilidad más adelante.

2.5.5. RELACIONES DE CONGRUENCIA Y MONOIDE COCIENTE.

Dado un monoide $\langle M, * \rangle$ y una relación de equivalencia, R, en M, R es una relación de congruencia en $\langle M, * \rangle$ si $a\ R\ b$ implica que

$$(a*c)\ R\ (b*c) \quad \text{y} \quad (c*a)\ R\ (c*b)$$

para todo $c \in M$.

Si R es una relación de congruencia en el monoide $\langle M, * \rangle$ el conjunto cociente $M/R = \{[a] / a \in M\}$ con la operación $<< \cdot >>$ definida por

$$[a] \cdot [b] = [a*b]$$

es un monoide. Este monoide se llama ***monoide cociente*** de M por R.

2.6. COMPORTAMIENTO DE ENTRADA-ESTADOS.

Sea un autómata finito $A = \langle E, S, Q, f, g \rangle$.

Sabemos que dados un símbolo de entrada y un estado podemos obtener el estado siguiente mediante la función

$$f: E \times Q \to Q$$

Dado solamente un símbolo de entrada, $e \in E$, podemos definir una función que aplique a cada estado el siguiente bajo esa entrada determinada, es decir, una función de Q en Q, $k(e): Q \to Q$. Existirá entonces una función

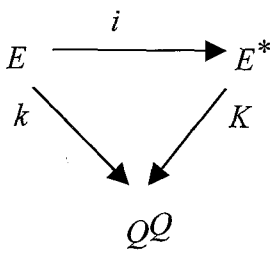
$$k : E \to Q^Q$$

siendo Q^Q el conjunto de las funciones de Q en Q, que sabemos que con la composición de funciones es un monoide. La función k se determina a partir de f del siguiente modo : para cada $e \in E$ y cada $q \in Q$, $[k(e)](q) = f(e,q)$.

Ahora bien, con las demostraciones anteriores, la función k puede extenderse a un homomorfismo entre monoides

$$K : \langle E^*, \rangle \to \langle Q^Q, o \rangle$$

con $K(e_1e_2.....e_n) = k(e_1)o\ k(e_2)o.....o\ k(e_n)$. Llamaremos a ***K comportamiento de entrada-estados*** del autómata.



2.7. RELACION EQUIRRESPUESTA Y MONOIDE DE UN AUTOMATA

Sea un autómata A con comportamiento de entrada-estados K.

La ***relación equirrespuesta*** de A, $\cong$, es una relación binaria de E* tal que

$$(\forall x,y \in E^*)\ [(x \cong y) \leftrightarrow (K(x) = K(y))]$$

Esta relación binaria, $\cong$, es una relación de equivalencia, ya que cumple:

1) $(\forall x \in E^*)\ [K(x) = K(x)]$
2) $(\forall x,y \in E^*)\ [(K(x) = K(y)) \rightarrow (K(y) = K(x))]$
3) $(\forall x,y,z \in E^*)\ [(K(x) = K(y)) \wedge (K(y) = K(z)) \rightarrow (K(x) = K(z))$

Además es una relación de congruencia en el monoide $\langle E^*, \rangle$. En efecto, si $x \cong y$, $K(x) = K(y)$, y $K(xz) = K(x)\ o\ K(z) = K(y)\ o\ K(z) = K(yz)$. Análogamente, $K(zx) = K(zy)$. Por tanto, según hemos visto anteriormente, $\langle E^*/\cong, \rangle$, es decir, el conjunto cociente $E^*/\cong$ con la operación de concatenación es un monoide.

Dado un autómata con un comportamiento entrada-estados, K, que origina una relación equirrespuesta $\cong$ en E^*, el monoide cociente $\langle E^*/\cong, \rangle$ se llama ***monoide del autómata.***

Si el número de estados es n, el monoide del autómata tendrá, como máximo n^n elementos.

En efecto, el número de transformaciones en el conjunto Q, es decir, de funciones diferentes $Q \rightarrow Q$ es n^n. El homomorfismo K aplica entonces un conjunto infinito E^* , en un conjunto Q^Q, que tiene n^n elementos. Si esta aplicación es sobreyectiva la relación equirrespuesta tendrá indice n^n , es decir, inducirá n^n clases de equivalencia en E^* (Si la aplicación no es sobreyectiva, el número de clases de equivalencia será menor). Por consiguiente, el número máximo de elementos de $E^*/\cong$ es n^n.

El monoide de un autómata refleja la capacidad de éste para responder de distinto modo a las cadenas de entrada.

Si dos cadenas x e y están en la misma clase, es decir, si $x \cong y$ entonces $K(x) = K(y)$, es decir, el homomorfismo K las aplica sobre el mismo elemento de Q^Q o lo que es lo mismo, ambas producen la misma transformación $Q \rightarrow Q$, y el autómata será incapaz de distinguir una de la otra.

2.8. MINIMIZACIÓN DE UN AUTÓMATA FINITO.

2.8.1. PLANTEAMIENTO DEL PROBLEMA.

Si disponemos de dos autómatas equivalentes que describen un sistema secuencial que debe realizarse físicamente escogeremos que el posea un menor número de estados, ya que el coste de la realización crecerá con el número de estos. Es, pues, importante saber si un autómata finito está en forma mínima, y, si no lo está, hallar un autómata finito en forma mínima equivalente a él. Comenzaremos por ver que este existe siempre; a continuación veremos que para detectar equivalencia entre estados no es preciso realizar infinitos ensayos con cadenas de entrada, y, finalmente, veremos un algoritmo para minimizar un autómata finito.

Ante todo, debemos señalar que la equivalencia entre estados de un autómata finito definida por

$$(\forall x \in E^*)\ [(q_1 = q_2) \leftrightarrow (Cq_1(x) = Cq_2(x))]$$

es una relación de equivalencia en el conjunto Q, pues, como es inmediato comprobar, es reflexiva, simétrica y transitiva. Por consiguiente, puede definirse un conjunto cociente, $Q/\equiv$, cuyos elementos serán clases de equivalencia : $[q]$ será la clase de equivalencia que contiene a q.

2.8.2. AUTÓMATA EN FORMA MÍNIMA

Se demuestra que dado un autómata finito $A = \langle E, S, Q, f, h \rangle$, el autómata definido por $A_M = \langle E, S, Q_M, f_M, h_M \rangle$, donde

$$Q_M = Q/\equiv$$

$$f_M(x,[q]) = [f(x,q)]$$

$$h_M([q]) = h(q)$$

es equivalente a A y está en forma mínima.

2.8.3. COMPROBACIÓN DE LA EQUIVALENCIA ENTRE ESTADOS.

Teorema de las particiones sucesivas: Si $P_0, P_1, P_2, \ldots\ldots$ es una secuencia infinita de particiones en un conjunto finito Q tal que, para todo k, se cumple:

a) P_{k+1} es un refinamiento de P_k, es decir, todo bloque de P_{k+1}, B^i_{k+1} está contenido en un bloque, B^j_k, de P_k : $B^i_{k+1} \subset B^j_k$.

b) $(P_{k+1} = P_k) \rightarrow (P_{k+2} = P_{k+1})$,

entonces existe un entero $k_0 < \text{card}(Q)$ tal que $(\forall k \geq k_0)\ (P_k = P_{k0})$.

Equivalencia de orden k entre estados: Dos estados de un autómata finito son equivalentes de orden k si y sólo si conducen a la misma salida para cadenas de entrada de longitud igual o inferior a k:

$$(q_1 \equiv_k q_2) \leftrightarrow [(\lg(x) \leq k) \rightarrow C_{q1}(x) = C_{q2}(x)]$$

Para comprobar la equivalencia de orden *k* ya no hay que hacer un número infinito de comprobaciones.

Teorema: Dado un autómata finito y $q_1, q_2 \in Q$ existe un $k_0 <$ card (Q) tal que q_1 y q_2 son equivalentes $(q_1 \equiv q_2)$ si y sólo si q_1 y q_2 son equivalentes de orden k_0 $(q_1 \equiv_{k0} q_2)$.

La teorema es importante ya que, sin necesidad de conocer k_0 , como $k_0 <$ n para comprobar si dos estados son equivalentes bastará con comprobar si son equivalentes de orden $n-1$.

2.8.4. ALGORITMO PARA LA MINIMIZACIÓN DE UN AUTÓMATA FINITO.

El algoritmo de minimización de un autómata finito se basa en los resultados anteriores:

Para ello se puede proceder del siguiente modo:

1) Si $\{q_1, q_2, \ldots\ldots, q_n\}$ son los estados de un determinado autómata, formamos con ellos una partición P_0 consistente en colocar en un mismo bloque

aquellos estados que tengan asociada la misma salida.

2) A continuación formamos las particiones P_{k+1} (k = 0,1, ...) teniendo en cuenta que dos estados estarán en el mismo bloque si para cada entrada los estados siguientes están en el mismo bloque que P_k.

3) El proceso termina cuando $P_{k+1} = P_k$. Si $P_{k+1} = \{Q_1, Q_2, ..., Q_m\}$ los estados del autómata en forma mínima son $q_1 = Q_1$, $q_2 = Q_2$,....., $q_m = Q_m$.

2.9. RECONOCEDOR FINITO.

Un lenguaje, L, sobre un alfabeto, A, es un subconjunto cualquiera de

$$A^* = \bigcup_{i=0}^{\infty} A^i$$

donde $A^0 = \{\lambda\}$ y A^i es el conjunto de cadenas de longitud i. A continuación vamos a ver que ciertos lenguajes pueden asociarse con autómatas finitos que sirven como reconocedores de las cadenas pertenecientes a tales lenguajes.

Un reconocedor finito de un lenguaje L es un autómata finito que sólo acepta las cadenas de dicho lenguaje, en el sentido de que, inicializado en un estado predeterminado, q_1 , si se le introduce una cadena de entrada $x_1 \in L$, da un símbolo final de salida que corresponde a "aceptación", mientras que para $x_1 \notin L$ produce una salida de no aceptación.

De acuerdo con esto, daremos la siguiente definición formal :

Un reconocedor finito es una quíntupla:

$$R = \langle E, Q, f, q_1, F \rangle$$

donde:

E = conjunto finito (alfabeto de entrada).

Q = conjunto finito (conjunto de estados).

f = es una función $f: E \times Q \rightarrow Q$ (función de transición).

$q_1 \in Q$ es un estado designado como estado inicial.

$F \subset Q$ es un conjunto de estados designados como estados finales.

Llamaremos $L(R)$ al conjunto de cadenas aceptadas por R, es decir,

$$L(R) = \{x \in E^* / f(x,q_1) \in F\}$$

(con el dominio de f ampliado a E^*).

Seguiremos el convenio de representar en los diagramas de Moore los estados de aceptación por un círculo con un grosor de línea superior a los demás, y en las tablas de transición encerrados en un círculo.

2.10. LENGUAJES ACEPTADOS POR RECONOCEDORES FINITOS.

Podemos preguntarnos si, dado un lenguaje cualquiera , $L \subset E^*$, podemos siempre encontrar un reconocedor finito, R, tal que $L(R) = L$.

Consideremos $E = \{0, 1\}$, y sea $L = \{1^{n^2} / n \geq 1\}$. No existe ningún reconocedor finito que acepte este lenguaje. Basta este ejemplo para demostrar que existen lenguajes a los que no corresponde ningún reconocedor finito.

Por tanto, vemos que ***los lenguajes aceptados por un reconocedor son un subconjunto de todos los lenguajes posibles sobre E^*.***

En el apartado 2.6. definiamos de manera general el comportamiento de entrada-estados de un autómata finito como un homomorfismo

$$K: \langle E^*, \rangle \rightarrow \langle Q^Q, o \rangle$$

tal que a cada $x \in E^*$ corresponde una transformación entre estados, $K(x): Q \rightarrow Q$ y esto nos permitió definir una relación equirrespuesta en E^* tal que $x \cong y$ si y sólo si $K(x) = K(y)$, es decir, las cadenas x e y estarán relacionadas si y sólo si producen la misma transformación entre estados:

$$(\forall q_i \in Q)\ [(x \cong y) \leftrightarrow f(x,q_i) = f(y,q_i)$$

Veíamos que esta relación es una relación de congruencia en $\langle E^*, \rangle$ y particiona E^* en un número finito de clases de equivalencia (es decir, es de índice finito, menor o igual a n^n , siendo n el número de estados).

En el caso del reconocedor finito, el estado inicial, q_1 es fijo, y el comportamiento de entrada-estados será más bien una función K_R: $E^* \rightarrow Q$ que aplica a cada $x \in E^*$ un $q \in Q$ de tal manera que $K_R(x) = f(x,q_1)$.

Podemos igualmente definir una relación equirrespuesta, $\cong_R$ en E^*:

$$(\forall x, y \in E^*)\ [(x \cong_R y) \leftrightarrow (f(x,q_1) = f(y, q_1))]$$

que es de equivalencia, como fácilmente se puede deducir. Además $(x \cong y) \rightarrow (x \cong_R y)$, pero no a la inversa, por lo que la partición de E^* inducida por $\cong_R$ tendrá un número igual o inferior de clases de equivalencia que en la inducida por $\cong$.

Si bien $\cong$ es una relación de congruencia en $\langle E^*, \rangle$, $\cong_R$ sólo es una ***relación de congruencia derecha***. Esto quiere decir que $(x \cong_R y) \rightarrow (xz \cong_R yz)$, pero en general, no es cierto que $(zx \cong_R zy)$.

Así pues, a cada reconocedor finito corresponde una partición de E^* en clases de equivalencia tal que si dos cadenas están en la misma clase ambas cadenas conducen al mismo estado. Además, esta relación de equivalencia es de índice finito

menor o igual a n^n , siendo n el número de estados del reconocedor.

Veamos como pueden aplicarse estas conclusiones para caracterizar a los lenguajes aceptados por reconocedores finitos.

Dado un lenguaje $L \subset E^*$, definimos la relación de congruencia derecha inducida por L, $\cong_L$, así:

$$(\forall x,y,z \in E^*)\ [(x \cong_L y) \leftrightarrow (xz \in L \leftrightarrow yz \in L)]$$

Es evidente que se trata de una relación de equivalencia. Para ver que también es una congruencia derecha basta suponer que $z = z_1 z_2$, con lo que

$$(\forall x,y,z_1,z_2 \in E^*)\ [(x \cong_L y) \leftrightarrow (xz_1z_2 \in L \leftrightarrow yz_1z_2 \in L) \rightarrow (xz_1 \cong_L yz_1)]$$

La principal conclusión de este apartado es que:

$L \subset E^*$ es un lenguaje aceptado por un reconocedor finito si y sólo si la relación de congruencia derecha inducida por L tiene índice finito.

Supongamos ahora que L es un lenguaje que induce una relación de congruencia derecha en E^* que es de índice finito. Vamos a construir un reconocedor

finito, R_L, tal que $L(R_L) = L$. Llamemos $[x]$ a la clase de equivalencia de $E^*/\cong_L$ que contiene a $x \in E^*$. Entonces definimos

$$R_L = \langle E, Q_L, f_L, q_1, F_L \rangle$$

donde

$Q_L = E^*/\cong_L = \{[x]\}$

$(\forall y \in E^*)\ (f_L(y, [x]) = [xy]$

$q_1 = [\lambda]$

$F_L = \{[x] / x \in L\}$

El lenguaje aceptado por este reconocedor será:

$$L(R_L) = \{y \in E^* / f_L(y, q_1) \in F_L\} = \{y / f_L(y, [\lambda]) \in F_L\} =$$
$$= \{y / [y] \in F_L\} = \{y / y \in L\} = L$$

Además, R_L está en forma mínima. En efecto, si $q_{L1} = [x_1]$ fuera equivalente a $q_{L2} = [x_2]$, esto querría decir que

$$(\forall y \in E^*)\ ((f_L(y,[x_1]) \in L) \leftrightarrow (f_L(y,[x_2]) \in L))$$

y por la definición de f_L,

$$(\forall y \in E^*)\ (([x_1 y] \in L) \leftrightarrow ([x_2 y] \in L))$$

es decir, $x_1 \cong_L x_2$; x_1 y x_2 se encuentran en la misma clase de equivalencia, por lo que $q_{L1} = q_{L2}$.

2.11. CONJUNTOS REGULARES. EXPRESIONES REGULARES.

Acabamos de ver una condición necesaria y suficiente para que un lenguaje sea aceptado por un reconocedor finito.

En este apartado vamos a exponer una herramienta, ***las expresiones regulares***, especialmente creada para trabajar con los lenguajes aceptados por reconocedores finitos, y que nos permitirá construir algoritmos que resuelvan los problemas de análisis y síntesis de reconocedores.

2.11.1. CONJUNTOS REGULARES

Sea $\{L_1, L_2, ...\}$ el conjunto formado por los posibles subconjuntos de E^* (lenguajes sobre E). Definamos tres operaciones en dicho conjunto:

a) Unión: $L_1 \cup L_2 = \{x / x \in L_1 \vee x \in L_2\}$

b) Concatenación: $L_1L_2 = \{x_1x_2 / x_1 \in L_1 \wedge x_2 \in L_2\}$

c) Cierre: $L^* = \{\lambda\} \cup \{L\} \cup \{LL\} \cup \{LLL\}..... = \bigcup_{n=0}^{\infty} L^n$

Diremos que un subconjunto de E^*, L_R, es un conjunto regular si y sólo si:

a) ***L_R es un subconjunto finito de E* (puede ser el conjunto vacío) o bien,***

b) ***L_R puede obtenerse a partir de subconjuntos finitos de E* mediante un número finito de operaciones de unión, concatenación y cierre.***

2.11.2. EXPRESIONES REGULARES.

Las expresiones regulares se introducen para describir los conjuntos regulares. Para designar el conjunto descrito por la expresión regular α utilizaremos la notación $[\alpha]$.

Dado un alfabeto $E = \{e_1, e_2, \ldots, e_n\}$, vamos a definir, en el conjunto de las expresiones regulares sobre dicho alfabeto, las operaciones suma, concatenación y cierre de forma recursiva:

a) λ (cadena vacía y elemento neutro dentro de la concatenación de cadenas), ϕ (conjunto vacío y elemento neutro para la unión de conjuntos) y e_i $(i = 1,2,\ldots,n)$ son expresiones regulares tales que $|\lambda| = \{\lambda\}$; $|\phi| = \phi$ y $|e_i| = \{e_i\}$ $(i = 1,2,\ldots,n)$;

b) Si α y β son expresiones regulares, $\alpha+\beta$ es una expresión regular tal que $|\alpha+\beta| = |\alpha| \cup |\beta|$;

c) Si α y β son expresiones regulares, $\alpha\beta$ es una expresión regular tal que $|\alpha\beta| = |\alpha||\beta|$;

d) Si α es una expresión regular, α^* es una expresión regular tal que $|\alpha^*| = |\alpha|^*$.

Dos expresiones regulares son iguales si designan al mismo conjunto regular: $(\alpha = \beta) \rightarrow |\alpha| = |\beta|$.

Teniendo esto presente, es fácil demostrar las siguientes propiedades de las expresiones regulares:

1. La concatenación es asociativa: $\alpha(\beta\gamma) = (\alpha\beta)\gamma$.
2. La suma es distributiva: $\alpha\beta + \alpha\gamma = \alpha(\beta + \gamma)$.
3. ϕ es el elemento neutro para la suma : $\alpha + \phi = \phi + \alpha = \alpha$.
4. ϕ es el elemento cero para la concatenación: $\alpha\phi = \phi\alpha = \phi$.
5. λ es el elemento neutro para la concatenación: $\alpha\lambda = \lambda\alpha = \alpha$.
6. Propiedades de la operación cierre:
   a) $(\alpha+\beta)^* = (\alpha^*+\beta^*)^* = (\alpha^*\beta^*)^*$.
   b) $(\alpha+\lambda)^* = \alpha^* + \lambda = \alpha^*$
   c) $\alpha\alpha^* + \lambda = \alpha^*$
   d) $\lambda^* = \phi^* = \lambda$

Se demuestra (Fundamentos de Informática. Gregorio Fernández) que todos los lenguajes aceptados por reconocedores finitos son conjuntos regulares y que todo conjunto regular es un lenguaje aceptado por un reconocedor finito, lo que nos va a permitir resolver los problemas de análisis y síntesis, respectivamente.

2.12. RESOLUCIÓN DE PROBLEMAS DE ANÁLISIS Y SÍNTESIS DE RECONOCEDOS FINITOS.

2.12.1. ALGORITMO DE ANÁLISIS.

Supongamos que un reconocedor finito tiene n estados, $Q = \{q_1,.....,q_n\}$, con estado inicial q_1 y con estados finales $F = \{q_{f1}, q_{f2}, ..., q_{fr}\}$ $(n \geq r \geq 0)$. El lenguaje aceptado es:

$$L(R) = \{x / f(x,q_1) \in F\} = \cup \{R_{1j} / q_j \in F\}$$

con $R_{ij} = \{x / f(x,q_i) = q_j\}$ (conjunto de cadenas que nos llevan del estado q_i al estado q_j). Bastará con demostrar que los R_{ij} son conjuntos regulares.

Vamos a definir R^k_{ij} $(0 \leq k \leq n)$ como el conjunto de cadenas que llevan del estado q_i al estado q_j sin pasar por ningún q_l tal que $l > k$. En particular, R^0_{ij} serán las cadenas que llevan del estado q_i al estado q_j sin pasar por ningún otro estado, por lo que serán símbolos, es decir, es un subconjunto de $E \cup \{\lambda\}$ y, por tanto, es regular ya que es finito. Supongamos que R^{k-1}_{ij} es regular para todo i, j y $k \geq 1$ y demostremos que R^k_{ij} es regular. En efecto, basta con comprobar que

$$R^k_{ij} = R^{k-1}_{ij} \cup R^{k-1}_{ik}(R^{k-1}_{kk})^* R^{k-1}_{kj}$$

Entonces, R^k_{ij} es regular para todo k, y en general, $R_{ij} = R^n_{ij}$.

De la demostración anterior se desprende un procedimiento para obtener la expresión regular del lenguaje aceptado por un determinado reconocedor.

En efecto, llamemos α^k_{ij} a la expresión regular que designa al conjunto R^k_{ij}: $|\alpha^k_{ij}| = R^k_{ij}$; entonces, si q_1 es el estado inicial, la expresión regular de $L(R)$ será:

$$\begin{cases} \alpha_{1f_1} + \alpha_{1f2} + ... + \alpha_{1fr} & si \quad r > 0 \\ \phi & si \quad r = 0 \end{cases}$$

Los α_{ij} se calcularán recursivamente, teniendo en cuenta que

$$|\alpha^0_{ij}| = \{e \in E \cup \{\lambda\} / f(x,q_i) = q_j\}$$
$$\alpha^k_{ij} = \alpha^{k-1}_{ij} + \alpha^{k-1}_{ik}(\alpha^{k-1}_{kk})^* \alpha^{k-1}_{kj} \quad (0 < k \leq n)$$
$$\alpha_{ij} = \alpha^n_{ij}$$

2.12.2. ALGORITMO DE SINTESIS

Aunque hay diversas demostraciones de este teorema, como las de Rabin y Scott (1959), o la de Ott y Feinstein (1961). Sin embargo, preferimos basarnos en los trabajos de Brozozowski (1962,1965) que introduce un algoritmo de fácil manejo empleando el concepto de ***derivada de una expresión regular***.

Sea α la expresión regular que designa al conjunto regular L_R, $|\alpha| = L_R$ y consideremos el subconjunto de L_R formado por todas las cadena de L_R que empiezan por un determinado símbolo e. Definimos el cociente izquierdo de L_R por e, $L_R\backslash e$, como el conjunto resultante de suprimir e en todas estas cadena:

$$L_R\backslash e = \{x / xe \in L_R\}$$

y definimos la derivada de α respecto del símbolo e , $D_e(\alpha)$, como la expresión regular de $L_R\backslash e$.

Las siguientes propiedades de las expresiones regulares se deducen fácilmente a partir de la definición:

1) $D_{e1}(e_2) = \begin{cases} \lambda & si \quad e_1 = e_2 \\ \phi & si \quad e_1 \neq e_2 \end{cases}$

2) $D_e(\lambda) = D_e(\phi) = \phi$

3) $D_e(\alpha + \beta) = D_e(\alpha) + D_e(\beta)$

4) $D_e(\alpha\,\beta) = [D_e(\alpha)\beta] + \delta(\alpha)D_e(\beta)$ con $\delta(\alpha) = \begin{cases} \phi & si \quad \lambda \notin |\alpha| \\ \lambda & si \quad \lambda \in |\alpha| \end{cases}$

5) $D_e(\alpha^*) = [D_e(\alpha)]\alpha^*$

6) $D_{xe}(\alpha) = D_e[D_x(\alpha)]$

Para construir un reconocedor finito en forma mínima del lenguaje denotado por la expresión regular α, a partir de las derivadas se puede seguir el siguiente procedimiento:

1) **Calcular** $\{ D_x(\alpha) / x \in E^* \}$
2) **El estado inicial es** $q_1 = \alpha$ **; los otros son las diferentes** $D_x(\alpha)$
3) **La función de transición es** $f(e,\alpha) = D_e(\alpha)$; $f(e, D_x(\alpha)) = D_{xe}(\alpha)$
4) **El conjunto de estados finales es** $F = \{ D_x(\alpha) / \lambda \in | D_x(\alpha)| \}$

2.13. DEFINICIÓN DE GRAMÁTICA.

Una gramática es una cuádrupla

$$G = \langle E_T, E_A, P, S \rangle$$

donde

E_T es un conjunto finito llamado ***alfabeto principal o alfabeto de símbolos terminales.***

E_A es un conjunto finito de símbolos, llamado ***alfabeto auxiliar o alfabeto de variables.***

S es un símbolo destacado del alfabeto auxiliar llamado ***símbolo inicial*** .

P es un conjunto finito de pares ordenados (α, β), donde $\beta \in (E_T \cup E_A)^*$ y

$\alpha \in (E_T \cup E_A)^+$, donde $(E_T \cup E_A)^*$ es el conjunto de todas las cadenas (incluida la cadena vacía, λ) que pueden formarse a partir de $E_T \cup E_A$ y $(E_T \cup E_A)^+ = (E_T \cup E_A)^* - \{\lambda\}$.

Estos pares ordenados reciben el nombre de ***reglas de escritura o producciones,*** y generalmente se escriben con la notación $\alpha \rightarrow \beta$. α es el antecedente de la regla y β el consecuente. Si $\beta \in E^*_T$ se dice que la regla es una regla terminal.

Si $\alpha \rightarrow \beta \in P$ y $\gamma, \delta \in E^*$, $(E = E_T \cup E_A)$ las cadenas $\gamma\alpha\delta$ y $\gamma\beta\delta$ están en **relación de derivación directa** y escribiremos

$$\gamma\alpha\delta \underset{G}{\Rightarrow} \gamma\beta\delta$$

y diremos que la cadena $\gamma\beta\delta$ ***deriva directamente*** de la cadena $\gamma\alpha\delta$.

Si $\alpha_1, \alpha_m \in E^*$, se dice que están en ***relación de derivación*** en la gramática G si existen $\alpha_2, \alpha_3, \ldots\ldots, \alpha_{m-1}$, tales que

$$\alpha_1 \underset{G}{\Rightarrow} \alpha_2 \underset{G}{\Rightarrow} \alpha_3 \underset{G}{\Rightarrow} \ldots\ldots \underset{G}{\Rightarrow} \alpha_{m-1} \underset{G}{\Rightarrow} \alpha_m$$

Lo denotaremos por $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$ y diremos que α_m deriva de α_1.

Observemos que $\underset{G}{\Rightarrow}$ y $\overset{*}{\underset{G}{\Rightarrow}}$ no son relaciones de equivalencia, pues en general no se cumple la propiedad simétrica.

2.14. LENGUAJE GENERADO POR UNA GRAMÁTICA.

El conjunto de sentencias que pueden generarse de una gramática G se llama ***lenguaje generado por una gramática***, es decir,

$$L(G) = \{x / x \in E^*_T \text{ y } S \overset{*}{\underset{G}{\Rightarrow}} x\}$$

Dos gramáticas que generan el mismo lenguaje se dice que son equivalentes, es decir G_1 y G_2 son equivalentes si $L(G_1) = L(G_2)$

2.15. CLASIFICACIÓN DE LAS GRAMÁTICAS Y DE LOS LENGUAJES.

Las gramáticas pueden clasificarse dependiendo que la forma que tienen las reglas de producciones. Partiremos de una gramática donde las reglas de escritura no poseen restricción alguna e introduciendo restricciones adicionales a las reglas obtendremos sucesivamente gramáticas cada vez más restringidas. Pasemos a comentar la clasificación debida a Chomsky y aceptada universalmente.

2.15.1. GRAMÁTICAS DE TIPO 0 Ó NO RESTRINGIDAS.

La gramática definida de una forma general en el apartado 2.13. se llama ***gramática de tipo 0 ó no restringida.***

Recordemos que las reglas de escritura o producciones son de la forma $\alpha \rightarrow \beta$, con $\alpha \in E^+$ y $\beta \in E^*$, es decir, la única restricción es que no pueden haber reglas de la forma $\lambda \rightarrow \beta$.

2.15.2. GRAMÁTICAS DE TIPO 1 Ó SENSIBLES AL CONTEXTO.

Este tipo de gramáticas lo forman aquellas cuyas reglas de escritura son de la forma

$$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$$

con $A \in E_A$; $\alpha_1, \alpha_2 \in E^*$; $\beta \in E^+$.

Es decir A puede reemplazarse por β siempre que esté en el contexto de α_1 y α_2.

Una propiedad importante de las gramáticas de tipo 1 es que ***las cadenas que se van obteniendo en cualquier derivación son de longitud no decreciente***, ya que al ser $\beta \neq \lambda$ se tiene que $lg(A) = 1 \leq lg(\beta)$, y $lg(\alpha_1 A \alpha_2) \leq lg(\alpha_1 \beta \alpha_2)$.

2.15.3 GRAMÁTICAS DE TIPO 2 Ó LIBRES DE CONTEXTO.

Las gramáticas de tipo 2 son un caso particular de las gramáticas de tipo 1, con $\alpha_1 = \alpha_2 = \lambda$, es decir, las reglas son del tipo

$$A \rightarrow \beta$$

con $A \in E_A$ y $\beta \in E^+$.

Estas gramáticas también se llaman ***Gramáticas de Chomsky*** y juegan un papel muy importante tanto en la lingüística como en la teoría de lenguajes de programación.

2.15.4. GRAMÁTICAS DE TIPO 3 Ó REGULARES.

Las gramáticas de tipo 3, también llamadas ***Gramáticas de Kleene*** son un caso particular de las gramáticas de tipo 2, con reglas de la forma

$$A \rightarrow aB \quad \text{ó} \quad A \rightarrow a$$

con $A, B \in E_A$ y $a \in E_T$.

Veremos más adelante que ***los lenguajes generados por las gramáticas de tipo 3 son exactamente los lenguajes regulares.***

2.15.5. JERARQUÍA DE LOS LENGUAJES.

Según hemos definido las gramáticas, es evidente que toda gramática regular es libre de contexto, toda gramática libre de contexto es sensible al contexto, y toda

gramática sensible al contexto es de tipo 0. Por consiguiente, si llamamos $L(G_3)$, $L(G_2)$, $L(G_1)$ y $L(G_0)$ a los lenguajes generados por estas gramáticas tendremos que

$$\{L(G_3)\} \subset \{L(G_2)\} \subset \{L(G_1)\} \subset \{L(G_0)\} \subset P(E^*)$$

2.15.6. LENGUAJES CON LA CADENA VACÍA.

Tal y como hemos definido las gramáticas, la cadena vacía, λ, no puede figurar en ningún lenguaje de tipo 1, 2 ó 3. Para que la cadena vacía esté contenida en dichos lenguajes bastaría con agregar la regla

$$S \rightarrow \lambda$$

a las reglas de la gramática que describen al lenguaje. Ahora bien, si hemos impuesto la condición de que $\beta \neq \lambda$ para conseguir la propiedad del no decrecimiento, será preciso que el símbolo inicial S no aparezca en el consecuente de ninguna regla.

Este problema queda solucionado introduciendo en el lenguaje auxiliar de la gramática un nuevo símbolo S_1, que se convertirá en el símbolo inicial y una regla de la forma $S_1 \rightarrow \alpha$ por cada regla $S \rightarrow \alpha$ que exista en la gramática.

Es decir, si tenemos la gramática $G = (E_T, E_A, P, S)$ formamos la gramática

$$G_1 = \langle E_T, E_A \cup S_1, P_1, S_1 \rangle$$

donde $P_1 = P \cup \{ S_1 \to \alpha / (S \to \alpha) \in P \} \cup (S_1 \to \lambda)$.

Es fácil comprobar que $L(G) \cup \{\lambda\} = L(G_1)$.

2.16. LENGUAJES REGULARES Y AUTÓMATAS FINITOS.

En la primera parte del capítulo hemos estudiado los reconocedores finitos y se ha visto que los lenguajes que pueden reconocer son los lenguajes regulares. Aquí vamos a ver que esa clase de lenguajes coincide precisamente con los lenguajes que pueden ser generados por las gramáticas de tipo 3. Para ello necesitamos introducir un concepto nuevo: el ***autómata finito no determinista.*** Su necesidad se verá claramente cuando se establezcan las relaciones entre gramáticas de tipo 3 y los autómatas.

2.16.1. AUTÓMATA FINITO NO DETERMINISTA.

Un autómata finito no determinista es una quíntupla

$$AFND = \langle E, S, Q, f, g \rangle$$

donde todo coincide con el autómata finito excepto que el rango de la función de transición no es Q sino el conjunto de las partes de Q, $P(Q)$:

$$f: E \times Q \to P(Q)$$

Es decir, $f(e,q) = \{q_a, q_b, ..., q_m\} \subset Q$.(Obsérvese que puede ser $f(e,q) = \phi$).

El reconocedor finito no determinista se define siguiendo la misma línea que en el caso determinista

$$RFND = \langle E, Q, f, q_1, F \rangle$$

El dominio de la función de transición puede extenderse a $E^* \times Q$ haciendo

$$f(\lambda, q) = \{q\}; \quad f(x_1x_2, q) = \bigcup_{\substack{q_k en \\ f(x_1,q)}} f(x_2, q_k)$$

También puede extenderse el dominio a $E^* \times P(Q)$ haciendo

$$f(x, \phi) = \phi$$

$$f(x, \{q_a, q_b, ..., q_m\}) = \bigcup_{j=a}^{m} f(x, q_j)$$

2.16.2. LENGUAJE ACEPTADO POR UN RECONOCEDOR FINITO NO DETERMINISTA.

Se define el lenguaje aceptado por un reconocedor finito no determinista como el conjunto de cadenas para las cuáles la función de transición conduce a un subconjunto de Q dentro del cuál se encuentra, al menos, un estado final:

$$L(RFND) = \{x \in E^* / [f(x,q_1) = \{q_a, q_b, ..., q_m\}] \wedge [\exists q_i \ (i = a,b,...,m) \in F]\}$$

A continuación pasemos a ver que la clase de los lenguajes aceptados por reconocedores finitos no deterministas coincide con la de los aceptados por los reconocedores finitos deterministas. Para ello será necesario comentar dos cosas:

a) que, dado un reconocedor finito determinista, *RFD*, siempre existe un reconocedor finito no determinista, *RFND*, tal que $L(RFND) = L(RFD)$, y por tanto, $\{L(RFD) \subset L(RFND)\}$.

b) que, dado un reconocedor finito no determinista, *RFND*, siempre existe un reconocedor finito determinista, *RFD*, tal que $L(RFD) = L(RFND)$, y por tanto, $\{L(RFND) \subset L(RFD)\}$ con lo que se demostrará que $\{L(RFND)\} = \{L(RFD)\}$.

Lo primero es evidente, ya que los reconocedores finitos deterministas son un caso particular de los reconocedores finitos no deterministas. Lo segundo se demuestra en el libro citado anteriormente de Fundamentos de Informática.

Terminamos el capítulo diciendo que los lenguajes regulares, los lenguajes que reconocen los reconocedores finitos y los lenguajes generados por las gramáticas del tipo 3 coinciden.

CAPÍTULO 3

GRAMÁTICAS GENERATIVAS Y RECOGNOSCITIVAS EN SISTEMAS COMPLEJOS

CAPÍTULO 3

GRAMÁTICAS GENERATIVAS Y RECOGNOSCITIVAS EN SISTEMAS COMPLEJOS

3.1. INTRODUCCIÓN.

En este capítulo se construyen gramáticas generativas y recognoscitivas que generan lenguajes matemáticos que pueden ser utilizados en la modelización matemática de sistemas complejos. Quedará, además, garantizada la obtención computacional de los lenguajes generados por las mismas.

Un caso concreto de los lenguajes que se han obtenido constituye el primer estudio realizado sobre esta materia por el autor de este proyecto junto con el director del mismo. Además, el estudio y la aplicación a un caso concreto de modelización matemática, se publica en un artículo junto con un grupo de colaboradores (Véase (56),Villacampa et al, 1999) y se hará referencia al mismo para señalar lo que ya ha sido publicado.

Normalmente, cuando mencionamos modelos estructurales complejos, como los ecológicos o los socioeconómicos, damos mucha importancia a la metodología empleada.

Nos apoyamos en el siguiente argumento: " La ayuda del ordenador está condicionada por la metodología empleada para modelizar, y esta metodología está condicionada por la teoría de sistemas en la que se basa".

Pero una metodología es un conjunto de métodos organizados que nos permite avanzar en una investigación de forma disciplinada ((7)Checkland, 1981). La literatura sobre metodología es muy extensa. Hay diferentes metodologías alternativas en modelización, aunque todas ellas poseen restricciones ((22)Gelovany, 1985; (14)Davis and O´Keefe, 1989).

La metodología en la que nos basamos para la modelización de los sistemas complejos se integran diversas ideas. Entre ellas destacan las desarrolladas en (18) y (19) por Forrester (1961,1966), en (7) por Checkland (1981), en (43) por Morecroft (1982), en (2) por Balci (1986), en (32) por Jorgensen (1988), en (39) por Mathewson (1989) y en (62) por Zhang et al. (1990). Se incorporan además las desarrolladas en los últimos años por Villacampa et al.(1997-1999) en los artículos (56) y (57): "Generative and Recognoscitive Grammars of Ecologogical Models with Aplications" publicado en la revista Ecological Modelling en 1999 y "A population model of the reproductive behaviour of mediterranean shrubs: A case of Cistus Albidus" en Ecosystems and Sustainable Development (Peñíscola 1997).

En este trabajo iniciamos una teoría de modelos de sistemas estructurales complejos desde el punto de vista de la Lingüística Matemática, como una manera más, no la única, de abordar este área del conocimiento humano

ya que estamos convencidos que cualquier metodología está asociada a un lenguaje y a sus correspondientes aspectos lingüísticos, tales como los componentes sintácticos y semánticos.

3.2. PRIMEROS CONCEPTOS.

A continuación se exponen los primeros conceptos que constituirán la base a partir de la cuál se definirán de forma abstracta un conjunto de gramáticas generativas, mediante las cuáles se construyen los lenguajes matemáticos para la modelización de sistemas complejos. Estos conceptos han sido publicados en la revista Ecological Modelling (ver 56, "Generative and Recognoscitive Grammars of Ecologogical Models with Aplications". Villacampa et al.1999).

3.2.1. MODELO BASE.

Definición:

Definimos el **Modelo Base** (MB) que denotamos por BM = (M*, R*) como el sistema determinado por

M* = {todos los atributos medibles asociados a la realidad óntica }
(Patten,1997 en (46))

R* = { todas las relaciones posibles entre los atributos}

3.2.2. SISTEMA SEMIÓTICO.

Definición:

El **sistema semiótico** es el sistema formado por S = (M, R) donde M es el conjunto de todos los atributos medibles y R es el conjunto de relaciones binarias $R \subset P(MxM)$ tales que $\forall r \in R$ es

$$r = \{(x_1, y_1), (x_2, y_2), ..., (x_i, y_i) / (x_i, y_i) \in MxM\}$$

3.2.3. CAMPO ASOCIATIVO DE UN ATRIBUTO MEDIBLE.

Sea x un atributo medible.

Definición:

Definimos el **campo asociativo** de x, y lo denotamos por Φ_x, como el conjunto formado por todas las posibles funciones transformadas (símbolos) de x.

$$\Phi_x = \{\{T^0(x)\}, \{T^1(x)\}, \{T^2(x)\}, ..., \{T^n(x)\},.....\}$$

donde $\{T^n(x)\}$ se llama conjunto de las funciones transformadas de orden n, es decir:

$\{T^0(x)\} = x$

$\{T^1(x)\} = \{f_1(x), f_2(x), \ldots\ldots\ldots, f_m(x), \ldots\ldots\ldots / f: R \rightarrow R\}$

$\{T^2(x)\} = \{f_i(x) \otimes f_j(x) \ / \ i,j = 1,2,\ldots\ldots\ldots,m,\ldots\ldots \}$

....

....

$\{T^n(x)\} = \{ \overleftrightarrow{f_i(x) \otimes f_j(x) \otimes \ldots\ldots \otimes f_k(x)}^{n} \ / \ i,j,k = 1,2,\ldots\ldots\ldots,m,\ldots\ldots \}$

....

donde ⊗ es una operación cualquiera de entre un conjunto dado: suma, producto, composición etc.

El conjunto Φ_X será un conjunto numerable con card(Φ_X) = N. Pero debido a las limitaciones del conocimiento humano, en la práctica se utilizará un subconjunto F_X de Φ_X con cardinal finito.

Como ejemplo, puede verse la figura 3.1.

x	transformada de orden 0 , $T^0(x)$
x^2, e^x, *sen(x), arctg(x)*,.................	Transformadas de orden 1, $T^1(x)$
x^2+e^x, $e^x \cdot cos(x)$, *sen*(x^2), ...	Transformadas de orden 2, $T^2(x)$
....	....

figura 3.1. Campo asociativo del atributo medible x

3.2.4. VOCABULARIO DE PRIMER ORDEN.

Sea $\Omega = \{x_1, x_2, \ldots\ldots, x_w\}$ un conjunto de atributos medibles. Cada elemento de Ω es una variable o primitiva y le llamaremos **p-símbolo.**

Definición:

Llamaremos **Vocabulario de primer orden**, VPO, y lo denotaremos por V^1_x, del p-símbolo x a su campo asociativo finito $F_x \subset \Phi_x$. Los elementos de V^1_x serán llamados **t-símbolos** y los representaremos por ϕ^j_i , donde i representa un índice del símbolo y j el orden de la transformada.. Es obvio que los p-símbolos son un caso particular de los t-símbolos.

Como ejemplo, supongamos que el modelizador decide utilizar:

a) un determinado p-símbolo, x,
b) m transformadas de orden 1,
c) la utilización de transformadas hasta el orden n, y
d) como operación $\otimes$ la composición de funciones

Entonces, el cardinal del VPO es

$$\text{card}(V^1_x) = 1 + m + m^2 + \ldots\ldots\ldots + m^n = \frac{m^{n+1} - 1}{m - 1}$$

como se desprende, también de la figura 3.2.

conjunto de símbolos	cardinal del conjunto de símbolos
$\{T^0(x)\}$	1
$\{T^1(x)\}$	m
$\{T^2(x)\}$	m^2
.....	.....
$\{T^n(x)\}$	m^n

figura 3.2

4.2.5. t-ALFABETO DE ORDEN i.

Definición:

Definimos el t-alfabeto de orden i del atributo x, y lo denotaremos por A^i_x al conjunto de t-símbolos del mismo orden i.

$$A^0_x = \{\phi^0_1(x) = x\}$$

$$A^1_x = \{\phi^1_1(x),\ \phi^1_2(x),\ ....,.... \phi^1_m(x)\}$$

$$A^2_x = \{\phi^2_1(x),\ \phi^2_2(x),\ .....\}$$

.....

.....

$$A^n_x = \{\phi^n_1(x),\ \phi^n_2(x),\ .....\}$$

De lo que se deduce que

$$V^l{}_x = A^0{}_x \cup A^1{}_x \cup \ldots\ldots \cup A^n{}_x = \bigcup_{i=0}^{n} A^i_x$$

3.2.6. VOCABULARIO DE SEGUNDO ORDEN.

Definición:

Definimos el vocabulario de segundo orden, VSO, y lo denotamos por $V^2{}_{xy}$, al vocabulario formado por dos t-símbolos de vocabularios de primer orden $V^1{}_x$ y $V^1{}_y$ de la siguiente forma:

$$^2\Psi^{rs}{}_{ij}(x,y) = \phi^r{}_i(x) \oplus \phi^s{}_j(y)$$

donde $\phi^r{}_i(x) \in V^1{}_x$, $\phi^s{}_j(y) \in V^1{}_y$ y $\oplus$ es un operador aritmético.

Es decir,

$$V^2{}_{xy} = \{ ^2\Psi^{rs}{}_{ij}(x,y) = \phi^r{}_i(x) \oplus \phi^s{}_j(y) \}$$

con i = 1,...., card($V^1{}_x$) ;
j = 1,...., card($V^1{}_y$) ;
r,s = 0,1,...., n

3.2.7. VOCABULARIO DE ORDEN N.

De igual manera, podemos definir vocabularios de orden 3,4, ..., por lo que podremos definir el vocabulario de orden n, así:

Definición:

Se define el vocabulario de orden n, VON, y lo denotamos por $V^n_{x_1x_2\ldots x_n}$, al vocabulario formado por n t-símbolos de vocabularios de primer orden V^1_{x1} y V^1_{x2},....., V^1_{xn} de la siguiente forma:

$$^{n}\Psi^{rs\ldots w}_{ij\ldots t}(x_1x_2\ldots x_n) = \phi^{r}_{i}(x_1) \oplus \phi^{s}_{j}(x_2) \oplus \ldots\ldots\ldots \oplus \phi^{w}_{t}(x_n)$$

A los elementos de la forma $^{n}\Psi^{rs\ldots w}_{ij\ldots t}(x_1x_2\ldots x_n)$ se les llamará **palabras**. Estas palabras, escritas con esos vocabularios han sido utilizadas para la modelización matemática de las ecuaciones de flujo de la dinámica de sistemas (Forrester y otros autores nombrados en el apartado 3.1.). Estas ecuaciones nos expresan un determinado proceso definido por una variable en función de flujos de entrada y salida. La expresión matemática de esta variable nos conducirá a la modelización del mismo. Por otra parte, en sistemas complejos estas ecuaciones que quedan definidas a partir de otras (flujos de entrada y salida) son ecuaciones implícitas de variables de las que sólo se pueden conocer datos numéricos experimentales o simulación de los mismos.

Para un caso concreto de elección de vocabularios de primer orden se han realizado modelizaciones de sistemas ecológicos en (56) y (57) (Villacampa et al 1977, 1999).

3.2.8. t-LÉXICO.

Dados n p-símbolos x_1, x_2,....., x_n, llamaremos **t-léxico** y lo denotaremos por t-L, al conjunto de todos los vocabularios de cualquier orden, es decir,

$$t\text{-}L = \{V^1_{x1}, V^1_{x2}, ..., V^1_{xn}, V^2_{x1x2}, ..., V^2_{x1xn}, ..., V^n_{x1x2.....xn}, ...\}$$

Las combinaciones lineales de palabras de un t-Léxico se llaman **sentencias ó funciones de flujo**, las cuáles se presentan en los Sistemas Dinámicos.

3.2.9. LÉXICO PRIMARIO.

Al subconjunto T de t-L formado por los vocabularios de primer orden generados por los p-símbolos $x_1, x_2, ..., x_n$, se llamará **léxico primario** (LP) o alfabeto de las funciones transformadas

$$T = \{V^1_{x1}, V^1_{x2}, ..., V^1_{xn}\}$$

3.2.10. VOCABULARIO DIFERENCIAL O d-VOCABULARIO.

Definimos el **vocabulario diferencial o d-vocabulario** de un p-símbolo x y lo denotamos por V^{∂}_x al conjunto formado por todas las derivadas parciales de cualquier orden de x respecto a cualquier otro p-símbolo y el tiempo t.

El **vocabulario diferencial primario** $V^{1\partial}_x$ es el conjunto de todas las derivadas parciales de primer orden de x respecto a cualquier otro p-símbolo y el tiempo t.

$$V^{1\partial}_x = \left\{\frac{\partial x}{\partial t}, \frac{\partial x}{\partial y}, ...\right\}$$

También pueden definirse vocabularios diferenciales de segundo, tercer,.....etc., orden , aunque por facilidad de cálculo, en las modelizaciones llevadas acabo hasta la fecha, de sistemas complejos sólo se utiliza un subconjunto de cardinal 1 del conjunto $V^{1\partial}_{x}$ y éste es la derivada parcial del p-símbolo x respecto al tiempo.

3.2.11. LÉXICO DIFERENCIAL .

Sea x_1, x_2,....., x_n, un conjunto de p-símbolos. Definimos el **léxico diferencial**, d-L, al conjunto formado por los d-vocabularios generados por los p-símbolos

$$\text{d-L} = \{ V^{\partial}_{x1}, V^{\partial}_{x2},....., V^{\partial}_{xn}\}$$

3.2.12. LÉXICO DE UN SISTEMA COMPLEJO.

Definición:

El léxico de un sistema complejo, L, es la unión del t-léxico y el d-léxico

$$L = (\text{t-L}) \cup (\text{d-L})$$

3.2.13. PALABRAS SINÓNIMAS.

Se considera un Modelo Base y una variable del mismo, *y*, de la que conocemos una relación de dependencia respecto a un conjunto de variables

$\{x_j\}_{j=1}^{n}$: $y = f(x_1, x_2, ..., x_n)$ (pudiendo ocurrir que $\exists\ i\ /\ x_i = y$). Sean $V_{x_j}^1$ los vocabularios de primer orden de las variables $\{x_j\}_{j=1}^{n}$.

Para la variable *y*, se supone que mediante un determinado proceso de modelización se obtiene diferentes palabras Ψ_i, i = 1,2,.....,h que la modelizan matemáticamente. Estas palabras estarán escritas en los vocabularios $V_{\delta_1....\delta_k}^k$ tal que $k \leq n$ y $\delta_i \in \{x_j\}_{j=1}^{n}$, $\forall i = 1,....,k$. Diremos entonces que estas **palabras son sinónimas.**

Definición:

Podemos definir de forma general una **relación de similitud** de la siguiente manera:

Para un Modelo Base BM = (M*,R*) sea $X = \{x_1, x_2, ..., x_m\}$ el conjunto de atributos considerados y sean $V_{x_i}^1$ los vocabularios de primer orden correspondientes. Se define como $\mathcal{P}$ el conjunto de palabras escritas en los vocabularios $V_{\delta_1....\delta_k}^k$, $k \leq m$ y $\delta_i \in \{x_j\}_{j=1}^{n}$. En $\mathcal{P} = \{\Psi_i\}$ definimos la siguiente relación, S:

Ψ_i S Ψ_j sii son **palabras sinónimas**

Esta relación es una relación de equivalencia ya que satisface las propiedades siguientes:

a) Reflexiva : $\forall \Psi_i \in \mathcal{P}$ se verifica que Ψ_i S Ψ_i

b) Simétrica: $\forall \Psi_i , \Psi_j \in \mathcal{P}$ se verifica que $\Psi_i \, S \, \Psi_j \Rightarrow \Psi_j \, S \, \Psi_i$

c) Transitiva : $\forall \Psi_i , \Psi_j , \Psi_k \in \mathcal{P}$ se verifica que si

$$(\Psi_i \, S \, \Psi_j) \text{ y } (\Psi_j \, S \, \Psi_k) \Rightarrow \Psi_i \, S \, \Psi_k$$

Esta relación de equivalencia , S, divide el conjunto $\mathcal{P}$ en clases de equivalencia.

En el conjunto cociente $\mathcal{P}/S$ y en cada clase de equivalencia se van a considerar las palabras que llamaremos **sinónimas intrínsecas** y **sinónimas extrínsecas.**

Definición:

Si dos palabras Ψ_1 y Ψ_2 de un mismo proceso tienen el mismo número de elementos y sus símbolos respectivos pertenecen al mismo vocabulario de primer orden, es decir, si

$$\Psi_1 = \phi_1 \oplus \phi_2 \oplus \ldots\ldots \oplus \phi_n$$
$$\Psi_2 = \phi'_1 \oplus \phi'_2 \oplus \ldots\ldots \oplus \phi'_n$$

donde

$$\phi_1 \in V^1_{x1} , \phi_2 \in V^1_{x2} , \ldots\ldots\ldots , \phi_n \in V^1_{xn} \text{ y}$$
$$\phi'_1 \in V^1_{x1} , \phi'_2 \in V^1_{x2} , \ldots\ldots\ldots , \phi'_n \in V^1_{xn}$$

decimos que son **sinónimas intrínsecas** y lo expresamos mediante

$$\Psi_1 \, S_{int} \, \Psi_2$$

Si símbolos respectivos de ambas palabras pertenecen al mismo vocabulario de primer orden pero tienen diferente longitud entonces decimos que son **sinónimas casi-intrínsecas.**

Definición:

Si uno, al menos, de los símbolos de ambas palabras que definen el mismo proceso pertenecen a diferentes vocabularios de primer orden decimos que las palabras son **sinónimas extrínsecas**.

3.3. GRAMATICA GENERATIVA DE LAS FUNCIONES TRANSFORMADAS.

A continuación se va a obtener un método de generación de vocabularios que permitirán la construcción de palabras de un t-léxico cuyas combinaciones lineales nos determinan las funciones de flujo que modelizan procesos y quese presentan en los estudios de Sistemas Dinámicos.

Para ello se comenzará con la construcción de gramáticas generativas de las funciones transformadas. Dichas funciones van a constituir el alfabeto con el que posteriormente se calculan las ecuaciones matemáticas que modelizan los procesos considerados en un modelo de un sistema complejo.

Como caso particular, y utilizando únicamente la composición de funciones como operación, se obtendrán los vocabularios de orden n considerados al principio del capítulo y utilizados en modelizaciones ("A population model of the reproductive behaviour of mediterranean shrubs: A case of Cistus Albidus" en Ecosystems and Sustainable Development (Peñíscola 1997)).

Esta abstracción va a suponer, por una parte, la obtención de un mayor número de lenguajes y, por consiguiente, la disponibilidad de una herramienta más completa para la modelización. Por otra parte, la obtención de estas gramáticas y sus correspondientes lenguajes nos conduce directamente a la teoría de autómatas finitos. De esta manera seremos capaces de obtener un reconocedor finito de los lenguajes que se generan. Estos lenguajes son los utilizados posteriormente para la construcción de ecuaciones matemáticas.

3.3.1. GRAMATICA GENERATIVA DE LAS FUNCIONES TRANSFORMADAS DE PRIMER ORDEN.

Sea x una variable real (atributo medible) y $f_1, f_2, \ldots, f_m$, m funciones reales de variable real,

$$f_i : R \rightarrow R, \qquad i = 1, 2, \ldots, m$$

Llamaremos gramática generativa de las funciones transformadas, G_{T1}, a la cuádrupla

$$G_{T1} = \langle V_T, V_A, S, P \rangle$$

donde

V_T es el alfabeto principal o vocabulario de símbolos terminales y está formado por los elementos del conjunto

$$V_T = \{ f_1, f_2, ..., f_m, (,), x \}$$

V_A es el alfabeto auxiliar o vocabulario de símbolos auxiliares y está formado por los elementos del conjunto

$$V_A = \{ S, R_1, R_2, R_3 \}$$

$S \in V_A$ es un símbolo destacado llamado símbolo inicial

P es un conjunto finito de pares ordenados (α, β) ó reglas $\alpha \rightarrow \beta$ donde $\alpha \in V_A$ y β es de la forma $\beta = a \in V_T$ ó $\beta = ab$ con $a \in V_T$ y $b = \lambda$ ó $b \in V_A$ dadas por

$$P = \begin{cases} S \rightarrow x \\ S \rightarrow f_i R_1 \\ R_1 \rightarrow (R_2 \\ R_2 \rightarrow x R_3 \\ R_3 \rightarrow) \end{cases}$$

Con estas reglas generamos un lenguaje, $L(G_{T1})$ cuyas cadenas o palabras forman los t-albabetos de orden 0 y de orden 1 como puede observarse en el siguiente árbol

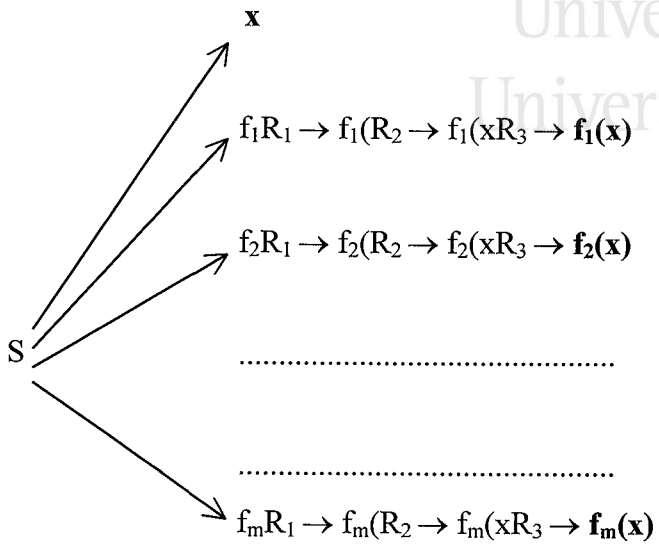


figura 3.4

y en el que podemos poner que

$$A^0{}_x = \{\phi^0{}_1(x) = x\}$$

$$A^1{}_x = \{\phi^1{}_1(x),\ \phi^1{}_2(x),\ \ldots,\ldots\ \phi^1_m(x)\} = \{f_1(x), f_2(x),\ \ldots\ldots\ldots, f_m(x)\}$$

$$L(G_{T1}) = A^0{}_x \cup A^1{}_x$$

La gramática que acabamos de definir, G_{T1}, es una gramática de tipo 3 o gramática de Kleen, pues, tal y como vimos en la sección 3.4.4 sus reglas son de la forma

$$A \rightarrow aB \quad \text{ó} \quad A \rightarrow a$$

con $A, B \in V_A$ y $a \in V_T$.

De ahí que, el lenguaje $L(G_{T1})$ sea un lenguaje regular. La expresión regular que representa las cadenas de dicho lenguaje es

$$\alpha = x + [f_1 + f_2 + ...+ f_m](x)$$

y derivando esta expresión regular podemos encontrar el reconocedor finito de este lenguaje:

Derivadas primeras:

$D_x(\alpha) = \lambda$

$D_{f1}(\alpha) = D_{f2}(\alpha) = ... = D_{fm}(\alpha) = (x)$

Derivadas segundas:

$D_{f1(}(\alpha) = D_{(}[D_{f1}(\alpha)] = x)$

$D_{f2(}(\alpha) = D_{(}[D_{f2}(\alpha)] = x)$

.....

$D_{fm(}(\alpha) = D_{(}[D_{fm}(\alpha)] = x)$

las restantes derivadas segundas son ϕ

Derivadas terceras:

$D_{fi(x}(\alpha) = Dx[D_{(}[D_{fi}(\alpha)]] =)$ $\quad i = 1,2,.....,m$

las restantes derivadas terceras son ϕ.

Derivadas cuartas:

$D_{fi(x)}(\alpha) = D_{)}[\, D_{fi(x}(\alpha)] = \lambda$

Las restantes derivas cuartas son ϕ.

Podemos, así, dibujar el diagrama de Moore (figura 3.5) del reconocedor finito, donde llamaremos

$q_1 = S$

$q_2 = D_x(\alpha)$

$q_3 = D_{f1}(\alpha) = D_{f2}(\alpha) = \ldots\ldots\ldots = D_{fm}(\alpha)$

$q_4 = D_{fi(}(\alpha)$

$q_5 = D_{fi(x}(\alpha)$

$q_6 = D_{fi(x)}(\alpha)$ $\quad i = 1,2,\ldots..,m$

siendo los estados finales q_2 y q_6 que son los que contienen la cadena vacía.

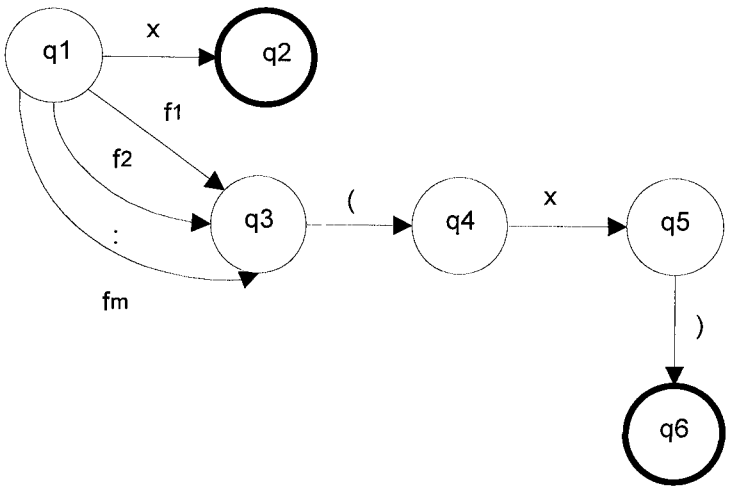


figura 3.5

3.3.2. GRAMATICA GENERATIVA DE LAS FUNCIONES TRANSFORMADAS DE SEGUNDO ORDEN

Llamaremos gramática generativa de las funciones transformadas, G_{T2}, a la cuádrupla

$$G_{T2} = \langle V_T, V_A, S, P \rangle$$

donde

V_T es el alfabeto principal o vocabulario de símbolos terminales y está formado la unión de los conjuntos

$$A^0{}_x = \{\phi^0{}_1(x) = x\}$$
$$A^1{}_x = \{\phi^1{}_1(x),\ \phi^1{}_2(x),\ \ldots,\ldots\ \phi^1_m(x)\}$$

que expresaremos para simplificar, por $\phi_0,\ \phi_1,\ldots\ldots,\ \phi_m$

$$\otimes = \{\otimes_1, \otimes_2,\ldots,\otimes_k\}$$

(los elementos $\otimes_i$, i = 1,2,...,k son operadores matemáticos)
y los paréntesis (y).

Es decir,

$$V_T = \{\ \phi_0,\ \phi_1,\ldots\ldots,\ \phi_m,\ \otimes_1, \otimes_2,\ldots,\otimes_n, (,)\}$$

V_A es el alfabeto auxiliar o vocabulario de símbolos auxiliares y está formado por los elementos del conjunto

$$V_T = \{ S, R_1, R_2, R_3, R_4 \}$$

$S \in V_A$ es el símbolo inicial

P es un conjunto finito de pares ordenados (α,β) ó reglas $\alpha \rightarrow \beta$ donde $\alpha \in V_A$ y β es de la forma $\beta = a \in V_T$ ó $\beta = ab$ con $a \in V_T$ y $b = \lambda$ ó $b \in V_A$ dadas por

$$P = \begin{cases} S \rightarrow (R_1 \\ R_1 \rightarrow \phi_i R_2 \\ R_2 \rightarrow \otimes_k R_3 \\ R_3 \rightarrow \phi_j R_4 \\ R_4 \rightarrow) \end{cases}$$

Con estas reglas generamos un lenguaje, $L(G_{T2})$ cuyas cadenas o palabras forman los t-albabetos de orden 2, como puede observarse en el árbol que representamos a continuación

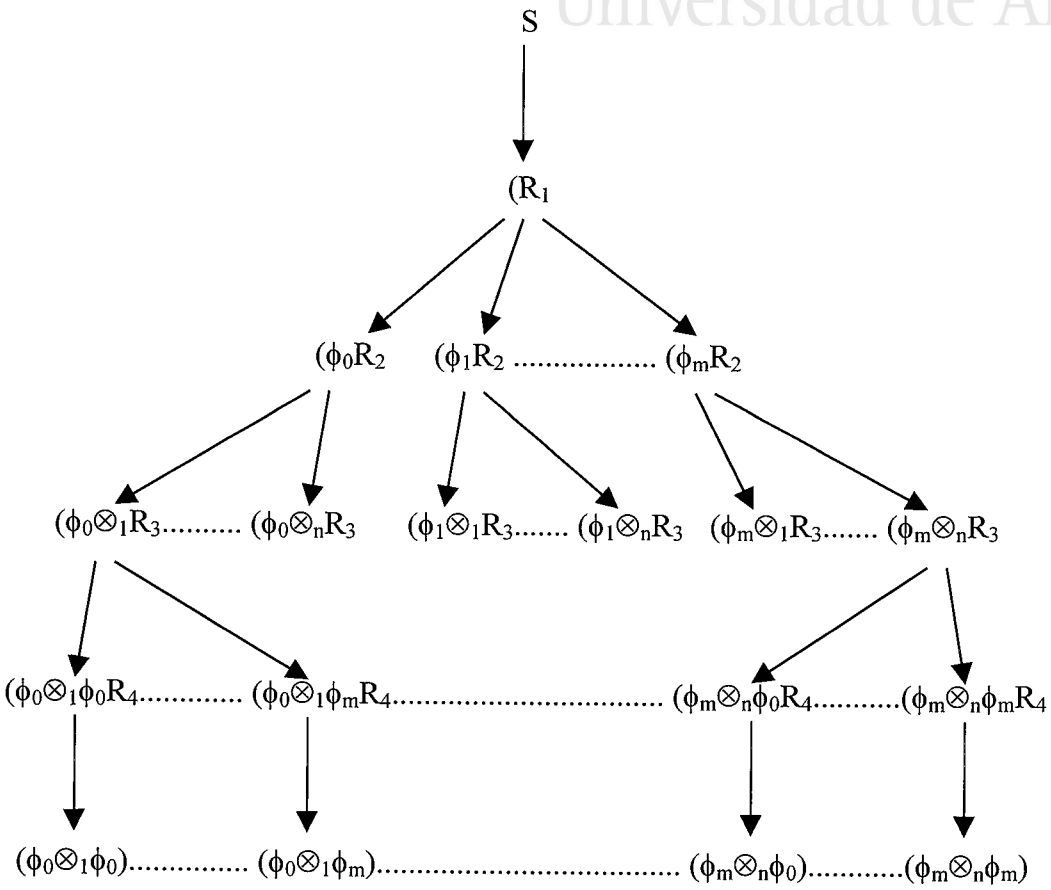


figura 3.6

El lenguaje generado por G_{T2}, es decir, $L(G_{T2})$ coincide con $L(G_{T2}) = A^2_x$.

Esta gramática que acabamos de definir, G_{T2}, también es una gramática de tipo 3 o gramática de Kleen, pues sus reglas son de la forma

$$A \rightarrow aB \quad \text{ó} \quad A \rightarrow a$$

con $A, B \in V_A$ y $a \in V_T$.

De ahí que, el lenguaje $L(G_{T2})$ sea un lenguaje regular. La expresión regular que representa las cadenas de dicho lenguaje es

$$\alpha = ([\phi_0 + ... + \phi_m][\otimes_1 + ... + \otimes_n] [\phi_0 + ... + \phi_m])$$

que en forma simplificada podemos expresar mediante

$$\alpha = (\phi_i \otimes_k \phi_j)$$

y derivando esta expresión regular podemos encontrar el reconocedor finito de este lenguaje:

Derivadas primeras:

$D_{(}[\alpha] = \phi_i \otimes_k \phi_j)$

Las primeras derivadas respecto a los restantes símbolos es la cadena vacía ϕ.

Derivadas segundas:

$D_{\phi i}[\phi_i \otimes_k \phi_j)] = \otimes_k \phi_j)$

las restantes derivadas segundas son ϕ.

Derivadas terceras:

$D_{\otimes k}[\otimes_k \phi_j)] = \phi_j)$

las restantes derivadas terceras son ϕ.

Derivadas cuartas:

$D_{\phi j}[\ \phi_j)] =)$

Las restantes derivadas cuartas son ϕ.

Finalmente, la única derivada quinta no nula es

$D_{)}\ [)] = \lambda$

y el diagrama de Moore correspondiente al reconocedor finito de las cadenas del lenguaje $L(G_{T2})$ será, llamando

$q_1 = \alpha$
$q_2 = D_{(}[\alpha]$
$q_3 = D_{(\phi i}[\alpha]$
$q_4 = D_{(\phi i \otimes k}[\alpha]$
$q_5 = D_{(\phi i \otimes k \phi j}[\alpha]$
$q_6 = D_{(\phi i \otimes k \phi j)}[\alpha] = \lambda$. Este estado es el estado final.

Los elementos de este lenguaje los podemos expresar mediante

$$\phi_{ij}^{k}$$

donde k indica la operación entre las funciones y los subíndices i y j son los índices de las funciones.

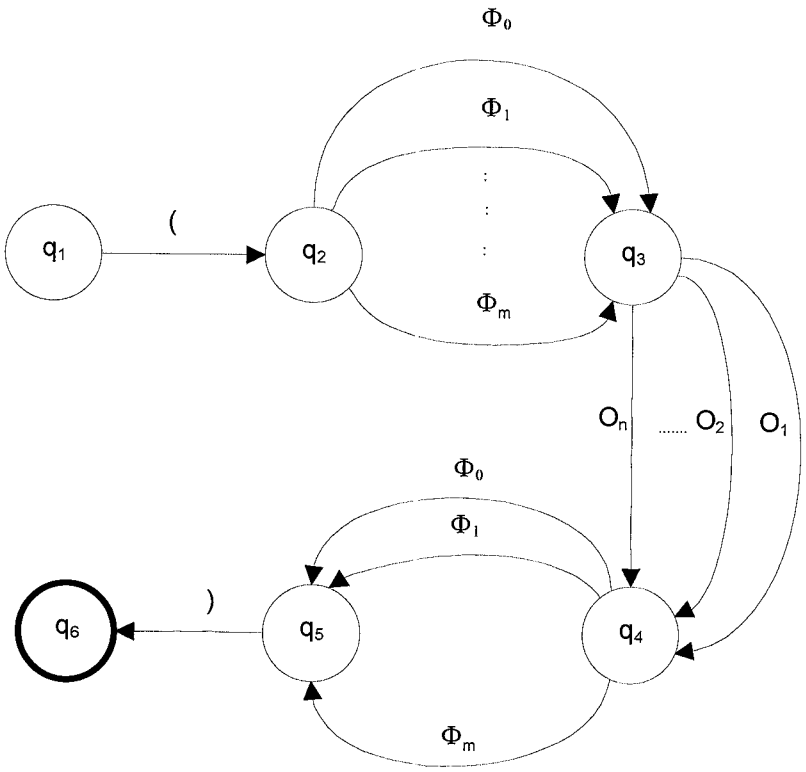


figura 3.7

3.3.3. GRAMATICA

GENERATIVA DE LAS FUNCIONES TRANSFORMADAS DE TERCER ORDEN.

Siguiendo un proceso similar a los apartados anteriores para la creación de las gramáticas G_{T1} y G_{T2}, vamos a diseñar una gramática que genere todas las funciones transformadas de orden tres, es decir, todas las funciones transformadas en las que aparezcan tres funciones transformadas de primer orden con las mismas operaciones $\otimes_k$ y utilizando como vocabulario terminal las cadenas obtenidas en las gramáticas anteriores.

Por tanto, llamaremos gramática generativa de las funciones transformadas, G_{T3}, a la cuádrupla

$$G_{T3} = \langle V_T, V_A, S, P \rangle$$

donde

V_T es el alfabeto principal o vocabulario de símbolos terminales y está formado por la unión de los conjuntos

$$V_T = L(G_{T1}) \cup L(G_{T2}) \cup \{\otimes_k\} \cup \{(,)\}$$

donde los elementos de $L(G_{T1})$ los representaremos por ϕ_i y los elementos de $L(G_{T2})$ por ϕ^s_{ij} .

V_A es el alfabeto auxiliar o vocabulario de símbolos auxiliares y está formado por los elementos del conjunto

$$V_T = \{ S, R_1, R_2, R_3, R_4, R_5\}$$

$S \in V_A$ es el símbolo inicial

P es un conjunto finito de reglas dadas por

$$P = \begin{cases} S \rightarrow (R_1 \\ R_1 \rightarrow \phi_i R_2 \\ R_1 \rightarrow \phi_{ij}^s R_5 \\ R_2 \rightarrow \otimes_k R_3 \\ R_3 \rightarrow \phi_{ij}^s R_4 \\ R_4 \rightarrow) \\ R_5 \rightarrow \otimes_k R_6 \\ R_6 \rightarrow \phi_i R_4 \end{cases}$$

Con estas reglas generamos un lenguaje, $L(G_{T3})$ cuyas cadenas o palabras forman los t-albabetos de orden 3 tienen la forma

$$(\phi_i \otimes_m (\phi_j \otimes_n \phi_k)) \quad \text{ó} \quad ((\phi_i \otimes_m \phi_j) \otimes_n \phi_k)$$

como puede observarse en el siguiente árbol

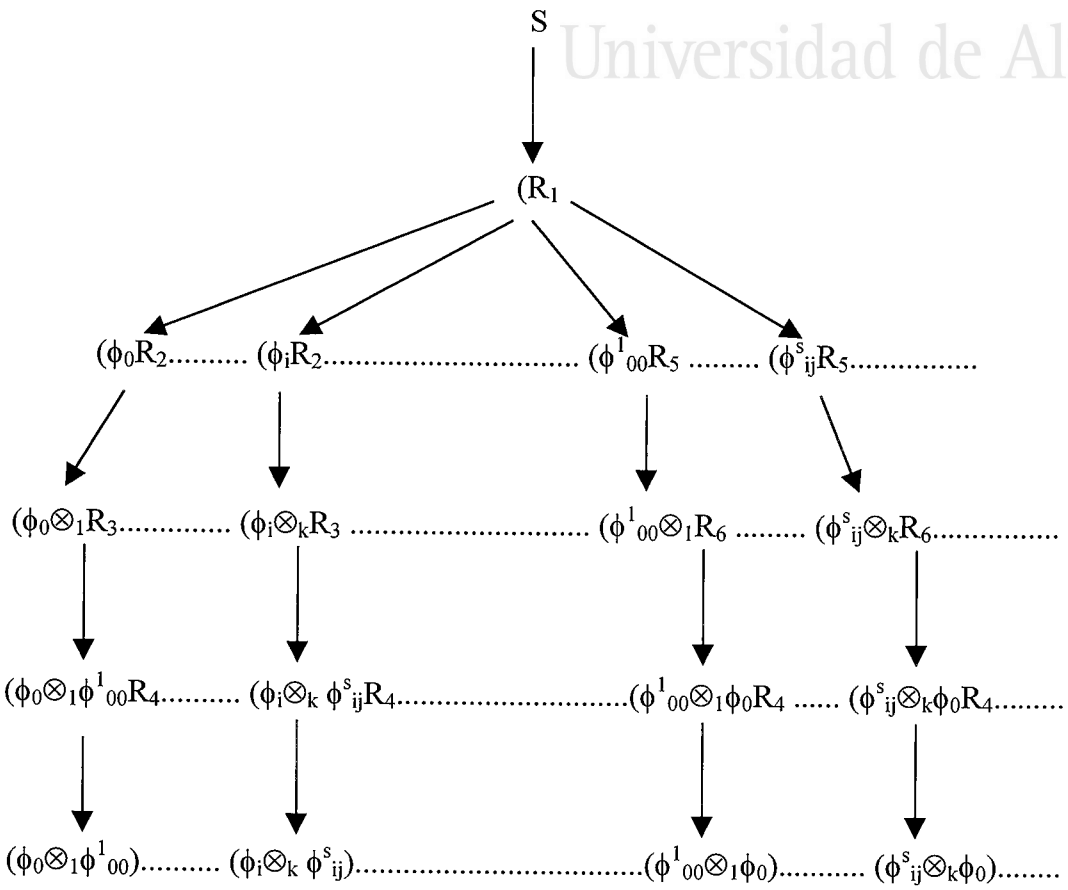


figura 3.8

El lenguaje generado por G_{T3}, es decir, $L(G_{T3})$ coincide con $L(G_{T3}) = A^3_x$.

Esta gramática que acabamos de definir, G_{T3}, también es una gramática de tipo 3. De ahí que, el lenguaje $L(G_{T3})$ sea un lenguaje regular. La expresión regular que representa las cadenas de dicho lenguaje es

$$\alpha = (\phi_i \otimes_m \phi^r_{jk}) + (\phi^s_{ij} \otimes_n \phi_k)$$

cuyas derivadas nos proporcionarán los estados del reconocedor de las cadenas del lenguaje $L(G_{T3})$.

Las derivadas no nulas las ofrecemos a continuación:

$\alpha = (\phi_i \otimes_m \phi^r_{jk}) + (\phi^s_{ij} \otimes_n \phi_k) = q_1$

Derivadas primeras:

$D_{(}[\alpha] = \phi_i \otimes_m \phi^r_{jk}) + \phi^s_{ij} \otimes_n \phi_k) = q_2$

Derivadas segundas:

$D_{\phi i}[D_{(}[\alpha]] = \otimes_m \phi^r_{jk}) = q_3$

$D_{\phi^s_{ij}}[D_{(}[\alpha]] = \otimes_n \phi_k) = q_6$

Derivadas terceras

$D_{\otimes_k}[D_{(\phi_i}[\alpha]] = \phi^k_{ij}) = q_4$

$D_{\otimes_k}[D_{(\phi^k_{ij}}[\alpha]] = \phi_i) = q_7$

Derivadas cuartas:

$D_{\phi^s_{ij}}[D_{(\phi_i \otimes_k}[\alpha]] =) = q_5$

Derivadas quintas:

$D_{)}[D_{(\phi_i \otimes_k \phi^k_{ij}}[\alpha]] = \lambda = q_8$

y como consecuencia de ellas obtendremos el diagrama de Moore del reconocedor finito de las cadenas de dicho lenguaje, que se representa de forma simplificada en la figura 3.9

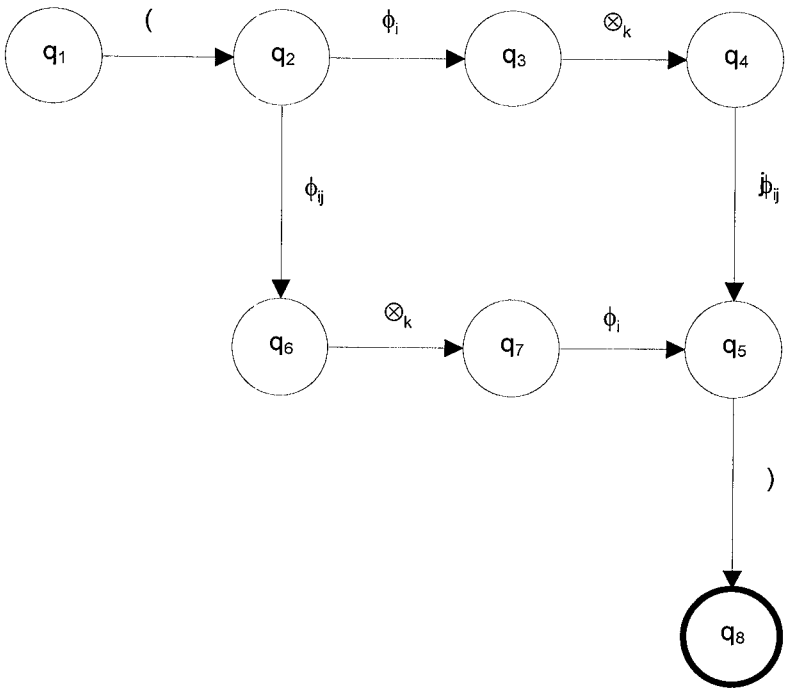


figura 3.9

3.3.4. GRAMATICAS GENERATIVAS DE LAS FUNCIONES TRANSFORMADAS DE ORDEN n

La obtención de gramáticas de orden superior a tres, siguiendo la misma estrategia de los casos anteriores, no entraña mayores dificultades, en cuanto a conceptos teóricos se refiere, y únicamente los problemas surgen a la hora de tener que expresar de forma general todas las combinaciones posibles entre las n funciones empleadas y las m operaciones que pueden existir entre las funciones.

Por ello, y a modo de ejemplo, veremos el proceso de obtención de una gramática generativa de orden 4, utilizando cinco funciones y la suma y la composición como operaciones que intervienen en las ecuaciones o palabras del lenguaje.

Funciones : f_1, f_2, f_3, f_4, f_5

Operaciones: +, o

Orden de las transformadas: 4

Teniendo en cuenta que en cada transforma intervienen cuatro funciones, habrá un número de transformadas igual a 845 obtenidas de la siguiente manera:

Con las 5 funciones se pueden formar $C_{5,4} = \binom{5}{4} = 5$ grupos, conteniendo cada uno de ellos 4 funciones

$\{f_1, f_2, f_3, f_4\}$, $\{f_1, f_2, f_3, f_5\}$, $\{f_1, f_2, f_4, f_5\}$, $\{f_1, f_3, f_4, f_5\}\{f_2, f_3, f_4, f_5\}$.

Con cada uno de estos grupos se pueden formar $P_4 = 4! = 24$ ordenaciones diferentes. Al tener que incluir tres operaciones entre las funciones, estas se

pueden ordenar de $VR_{2,3} = 2^3 = 8$ formas diferentes de las cuáles una será la misma, por ser la suma conmutativa, por ejemplo $f_1+f_2+f_3+f_4 = f_1+f_3+f_4+f_2$, por lo que sólo debemos considerarla una sola vez.

Por lo tanto el número de transformadas diferentes será $5(23\cdot7+8) = 845$, con la forma general $\alpha = f_i \otimes_r f_j \otimes_s f_k \otimes_t f_l$.

En este ejemplo, debido a las operaciones elegidas, no son necesarios los paréntesis.

3.4. UN CASO PARTICULAR DE GRAMÁTICA GENERATIVA DE FUNCIONES TRANSFORMADAS.

En este apartado vamos a considerar una gramática cuya principal característica es la de contener una única operación, la composición de funciones.

La gramática obtiene las transformadas de orden m de n funciones reales de variable real.

Hemos desarrollado la gramática tomando como vocabulario terminal, V_T, un conjunto finito formado por la unión de los conjuntos V_1 y V_2, donde V_1 está formado por n funciones reales de variable real f_i con i = 1, 2, ..., n :

$$V_1 = \{f_1, f_2, f_3, ..., f_n\}$$

y V_2 es el conjunto de símbolos

$$V_2 = \{(,), o, x\}$$

El vocabulario auxiliar, V_A, es un conjunto finito, formado por los símbolos T, R_1, R_2, R_3, R_4, R_5, R_6, R_7 es decir

$$V_A = \{T, R_1, R_2, R_3, R_4, R_5, R_6, R_7\}$$

donde T es el símbolo inicial.

El conjunto de reglas es el siguiente:

$T \rightarrow f_i R_1 \qquad i = 1,2,.....,n$

$R_1 \rightarrow (R_2$

$R_2 \rightarrow xR_3$

$R_3 \rightarrow)$

$T \rightarrow (R_4$

$R_4 \rightarrow f_i R_5$

$R_5 \rightarrow oR_6$

$R_6 \rightarrow f_i R_5$

$R_6 \rightarrow f_i R_7$

$R_7 \rightarrow)R_1$

Al conjunto de las funciones f_1, f_2, f_3,....., f_n le llamaremos conjunto de las funciones transformadas de orden 1, y lo denotamos por T^1.

Con las funciones anteriores y la aplicación sucesiva de las reglas de formación formamos todas las composiciones de orden 2, orden 3, ..., orden m,.... que llamamos respectivamente, transformadas de orden 1, transformadas de orden 2, ..., transformadas de orden m, ...y que denotamos por T^1, T^2, ..., T^m,....., es decir

$$T^2 = \{f_1 o f_1,\ f_1 o f_2,\ \ldots\ldots,\ f_n o f_n\}$$

$$T^3 = \{f_1 o f_1 o f_1,\ f_1 o f_1 o f_2,\ \ldots\ldots,\ f_n o f_n o f_n\}$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$\leftarrow\ m\ \rightarrow\ \leftarrow\ m\ \rightarrow \qquad\qquad \leftarrow\ m\ \rightarrow$$

$$T^m = \{f_1 o f_1 o \ldots\ldots o f_1,\ f_1 o f_1 o \ldots\ldots o f_2,\ \ldots\ldots,\ f_n o f_n o \ldots\ldots o f_n\}$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

El número total de palabras que forman el lenguaje generado por la gramática G_T formada a partir de n funciones transformadas de orden 1 y utilizando transformadas hasta el orden m es

$$1 + n + n^2 + \ldots\ldots + n^m = \sum_{j=0}^{m} n^j = \frac{n^{m+1} - 1}{n - 1}$$

Atendiendo a la clasificación de las gramáticas expuestas an apartados anteriores podemos observar que la gramática G_T es una gramática del tipo 3 o regular.

Es fácil observar que las cadenas generadas por dicha gramática pueden expresarse mediante la expresión regular :

$$\alpha = f_{i1}(x) + (f_{i2} o (f_{i3} o)^* f_{i4})(x)$$

Por derivación de la expresión regular obtendremos el autómata finito reconocedor del lenguaje $L(G_T)$ generado por la gramática G_T.

Por comodidad, para la obtención de las derivadas, llamaremos

$f_i = a$

$(= b$

$x = c$

$) = d$

$o = e$

con lo que la expresión regular quedaría como $\alpha = abcd + bae(ae)^*adbcd$

$D_a(\alpha) = bcd = q_6$

$D_b(\alpha) = ae(ae)^*adbcd = q_2$

$D_c(\alpha) = D_d(\alpha) = D_e(\alpha) = \phi$

$D_{ab}(\alpha) = D_b[D_a(\alpha)] = cd = q_7$

$D_{aa}(\alpha) = D_{ac}(\alpha) = D_{ad}(\alpha) = D_{ae}(\alpha) = \phi$

$D_{ba}(\alpha) = D_a[D_b(\alpha)] = e(ae)^*adbcd = q_3$

$D_{bb}(\alpha) = D_{bc}(\alpha) = D_{bd}(\alpha) = D_{be}(\alpha) = \phi$

$D_{abc}(\alpha) = d = q_8$

$$D_{aba}(\alpha) = D_{abb}(\alpha) = D_{abd}(\alpha) = D_{abe}(\alpha) = \phi$$

$$D_{bae}(\alpha) = (ae)^*adbcd = q_4$$

$$D_{baa}(\alpha) = D_{bab}(\alpha) = D_{bac}(\alpha) = D_{bad}(\alpha) = \phi$$

$$D_{abcd}(\alpha) = \lambda = q_9$$

$$D_{abca}(\alpha) = D_{abcb}(\alpha) = D_{abcc}(\alpha) = D_{abce}(\alpha) = \phi$$

$$D_{baea}(\alpha) = D_a[(ae)^*adbcd] = D_a[(ae)^*]adbcd + \delta((ae)^*)D_a(adbcd) =$$

$$= [D_a(ae)](ae)^*adbcd + \delta((ae)^*)D_a(adbcd) =$$

$$= e(ae)^*adbcd + \lambda dbcd = e(ae)^*adbcd + dbcd = q_5$$

$$D_{baead}(\alpha) = D_d[D_{baea}(\alpha)] = D_d[e(ae)^*adbcd + dbcd] = bcd = D_a(\alpha)$$

$$D_{baeae}(\alpha) = D_e[D_{baea}(\alpha)] = D_e[e(ae)^*adbcd + dbcd] = D_{bae}(\alpha)$$

Una vez obtenidas todas las derivadas no nulas de la expresión regular podemos dibujar el diagrama de Moore del autómata que reconoce el lenguaje $L(G_T)$:

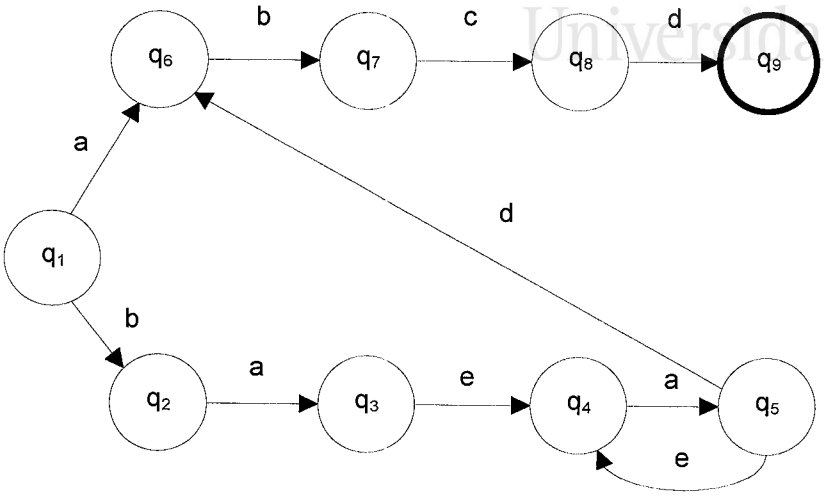


figura 3.10

Comprobación :

$$\alpha = abcd + baea(ea)^*dbcd$$

que es equivalente a la expresión regular de la que partimos.

Deshaciendo el cambio de símbolos queda :

$$\alpha = f_i(x) + (f_i of_i(of_i)^*)(x)$$

Ejemplo práctico:

En el apartado anterior hemos propuesto una gramática generativa de funciones transformadas general, es decir, sin fijar el orden máximo de las funciones transformadas a emplear.

En este ejemplo estudiaremos las modificaciones que debemos realizar en las reglas de escritura o producciones de la gramática para el caso particular de que queramos fijar de antemano un determinado orden máximo de funciones transformadas.

El estudio lo realizaremos sobre el caso particular de transformadas de orden 3, aunque con su desarrollo veremos que para cualquier otro orden de transformadas el estudio se realiza de forma idéntica.

La expresión regular toma en este caso la forma :

$$\alpha = f_{i1}(x) + (f_{i2} o\ f_{i3})(x) + (f_{i4} o\ f_{i5} o f_{i6})(x)$$

Teniendo en cuenta el código empleado en el apartado anterior, la expresión regular toma la forma :

$$\alpha = abcd + baeadbcd + baeaeadbcd$$

Derivamos sucesivamente la expresión regular para obtener los estados del autómata :

$D_a(\alpha) = bcd = q_8$

$D_b(\alpha) = aeadbcd + aeaeadbcd = q_2$

$D_{ab}(\alpha) = cd = q_9$

$D_{ba}(\alpha) = eadbcd + eaeadbcd = q_3$

$D_{abc}(\alpha) = d = q_{10}$

$D_{bae}(\alpha) = adbcd + aeadbcd = q_4$

$D_{abcd}(\alpha) = \lambda = q_{11}$

$D_{baea}(\alpha) = dbcd + eadbcd = q_5$

$D_{baead}(\alpha) = bcd = D_a(\alpha)$

$D_{baeae}(\alpha) = adbcd = q_6$

$D_{baeaea}(\alpha) = dbcd = q_7$

$D_{baeaead}(\alpha) = bcd = D_a(\alpha)$

El diagrama de Moore correspondiente es:

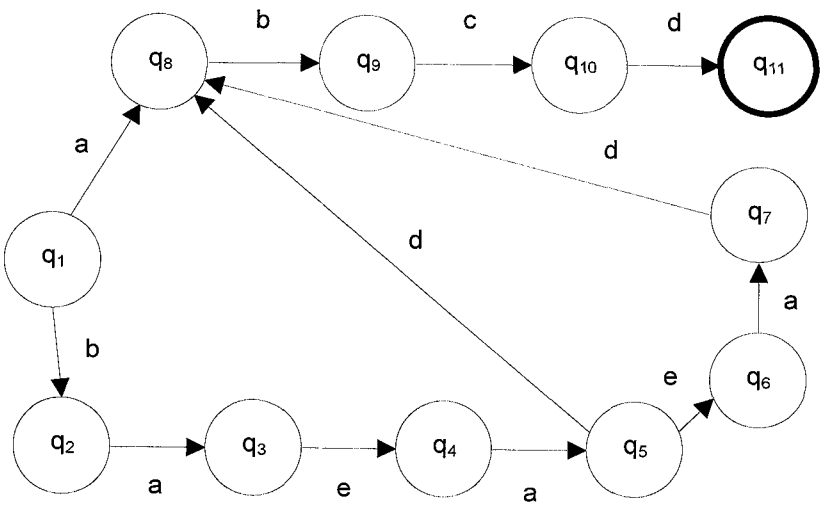


figura 3.11

3.5. GRAMÁTICA GENERATIVA DE LAS ECUACIONES DE FLUJO, G_F.

Las ecuaciones de flujo son ecuaciones matemáticas que se van a construir a partir de los lenguajes generados por las gramáticas funciones transformadas. Estas ecuaciones de flujo son las que se utilizarán en la modelización matemática de las variables consideradas en un Modelo Base.

Sean n variables o p-símbolos $x_1, x_2, \ldots\ldots, x_n$ y m funciones reales de variable real o transformadas de orden uno, $f_1, f_2, \ldots\ldots, f_m$.

Una vez elegidas las operaciones $\otimes = \{\otimes_1, \otimes_2, \ldots\ldots, \otimes_k\}$ y el máximo orden, p, de las transformadas a emplear en un determinado proceso de modelización, formamos, con las gramáticas generativas construidas en los apartados precedentes, los t-alfabetos de orden 0, 1, 2, ..., p de todos los p-símbolos, es decir, construimos los conjuntos

$$A^0{}_{xi} = \{\phi^0{}_1(x_i) = x_i\}$$

$$A^1{}_{xi} = \{\phi^1{}_1(x_i) = f_1(x_i),\ \phi^1{}_2(x) = f_2(x_i), \ldots\ldots,\ \phi^1{}_m(x_i) = f_m(x_i)\}$$

$$A^2{}_{xi} = \{\phi^2{}_1(x_i) = f_1(x_i) \otimes_1 f_1(x_i),\ \phi^2{}_2(x_i) = f_1(x_i) \otimes_1 f_2(x_i), \ldots\ldots\ldots\ldots\ldots\}$$

....

....

$$A^p{}_{xi} = \{\phi_1^p(x_i) = \overbrace{f_1(x_i) \otimes_1 \ldots\ldots \otimes_1 f_1(x_i)}^{p},\ \phi_2^p(x_i) = \overbrace{f_1(x_i) \otimes_1 \ldots\ldots \otimes_1 f_2(x_i)}^{p}, \ldots\}$$

con lo que podemos definir el vocabulario de primer orden de cada p-símbolo como la unión de estos alfabetos, es decir,

$$V_{xi}^{1} = \bigcup_{j=0}^{p} A_{xi}^{j}$$

Si designamos por t_{xi}^{j} a un elemento cualquiera, j, de V_{xi}^{1}, es decir, $t_{xi}^{j} \in V_{xi}^{1}$, llamaremos ecuación de flujo, Ψ, a toda ecuación de la forma

$$\Psi = A_0 \oplus_{q1} A_1 t_{x1}^{j1} \oplus_{q2} A_2 t_{x2}^{j2} \oplus_{q_3} ..... \oplus_{n-1} A_n t_{xn}^{jn}$$

donde $\oplus_q$ es un elemento de un determinado conjunto de operadores aritméticos y las A_i son números reales no todos nulos. Llamaremoa A = { A_0, A_1, ..., A_n}

Una gramática generativa de las funciones de flujo es la cuádrupla

$$GF = \langle VF_T, VF_A, T, P \rangle$$

donde el vocabulario de símbolos terminales es

$$VF_T = A \cup V_{x1}^{1} \cup V_{x2}^{1} \cup ..... \cup V_{xn}^{1} \cup \oplus$$

VF_A es el alfabeto auxiliar formado por el conjunto de símbolos

$$VF_A = \{T, R_1, R_2, ..., R_{4n}\}$$

donde T es el símbolo inicial.

Las reglas de escritura o formación de cadenas son :

$$T \rightarrow A_0 R_1$$

$$R_{4i-3} \rightarrow \oplus_{qi} R_{4i-2}$$

$$R_{4i-2} \rightarrow A_i R_{4i-1}$$

$$R_{4i-1} \rightarrow \cdot R_{4i}$$

$$R_{4i} \rightarrow t_{xi}^{ji} R_{4i+1}$$

$$R_{4n-3} \rightarrow \oplus_{qn} R_{4n-2}$$

$$R_{4n-2} \rightarrow A_n R_{4n-1}$$

$$R_{4n-1} \rightarrow \cdot R_{4n}$$

$$R_{4n} \rightarrow t_{in}^{jn}$$

para i = 1, 2,....., n-1

El diagrama de Moore correspondiente es, por tanto:

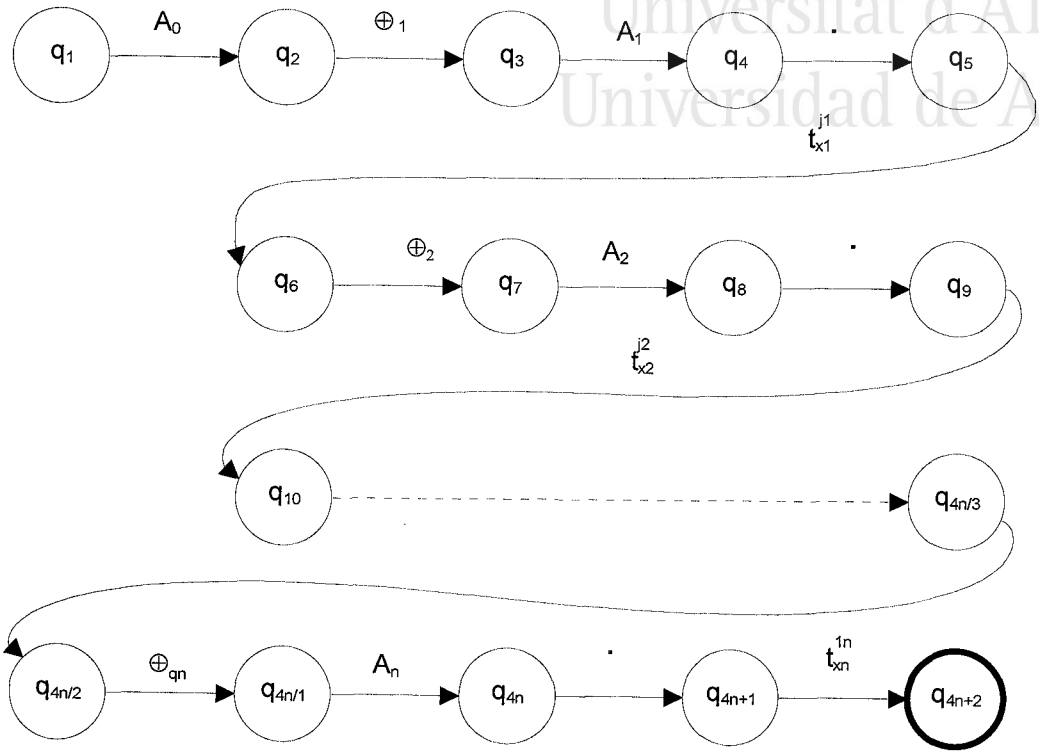


figura 3.12

3.6. GRAMATICAS RECOGNOSCITIVAS DE ECUACIONES DE FLUJO.

Una vez elegidas las gramáticas de las funciones transformadas, con los lenguajes generados por las mismas, y un conjunto de operadores aritméticos, previamente seleccionados formamos las gramáticas generativas de ecuaciones de flujo, las cuáles generan un lenguaje que denotamos por $L(G_F)$, cuyas sentencias (frases) o ecuaciones de flujo se producen todas ellas con la misma probabilidad, es decir, son equiprobables.

Una vez construidas las sentencias y dependiendo de los datos experimentales y de los criterios seguidos por el modelizador deberá elegirse una ecuación o un conjunto de ellas, es decir, deberá seguirse un determinado proceso de recognoscibilidad, que nos permita, de entre todas las sentencias, elegir la o las que consideraremos "correctas", siendo el resto rechazadas.

Serán, pues, múltiples las opciones que se pueden seguir en esta elección, opciones que llamaremos **criterios de recognoscibilidad.**

En los sistemas complejos, como son los ecológicos, las variables del modelo base que se desean escribir como ecuaciones de flujo son variables de las que tan sólo se conocen datos numéricos (datos experimentales) y demás variables de las que dependen. Estos datos experimentales (o en ocasiones, simulaciones de los mismos) son los que, por norma, van a determinar ciertos criterios de recognoscibilidad de los que a continuación se dan tres.

Primer criterio de recognoscibilidad:

Un primer criterio de recognoscibilidad, que denotaremos por GRF_1 y podemos llamar "criterio de máximo ajuste" está basado en el coeficiente estadístico de determinación (el cuadrado del coeficiente de correlación en el contexto de la regresión múltiple) (Usó-Domenech et al., 1997 en (52)).

Supongamos una variable de un Modelo Base y unos datos experimentales de la misma, así como datos de las variables de las que depende. Mediante un proceso iterativo son leídas todas las posibles ecuaciones de flujo que modelizan la variable, calculando en cada caso el coeficiente de correlación múltiple.

Un primer criterio de recognoscibilidad sería el seleccionar la ecuación de flujo cuyo coeficiente de correlación sea mayor. Este criterio ya ha sido utilizado en las modelizaciones realizadas por Villacampa et al. durante 1997 y1999 en (56) y (57).

Con este criterio seleccionaremos una sola ecuación de flujo: aquella cuyo coeficiente de correlación sea mayor.

La figura 3.13 muestra el proceso.

Segundo criterio de recognoscibilidad:

El segundo criterio de recognoscibilidad, GRF_2, consiste en fijar dos coeficientes de correlación r_1 y r_2 y considerar "correctas" todas las ecuaciones de flujo cuyo coeficiente decorrelación r_i esté comprendido entre ambos valores, es decir, $r_i \in [r_1, r_2]$ creando un conjunto de ecuaciones de flujo.

El diagrama de la figura 3.14 muestra el proceso.

Tercer criterio de recognoscibilidad:

Para la aplicación de este criterio, que denotamos por GRF_3, se elegirán funciones transformadas que se puedan normalizar en relación con los datos experimentales, desechando aquellas no no satisfagan este requerimiento. A continuación se elige uno de los dos criterios anteriores.

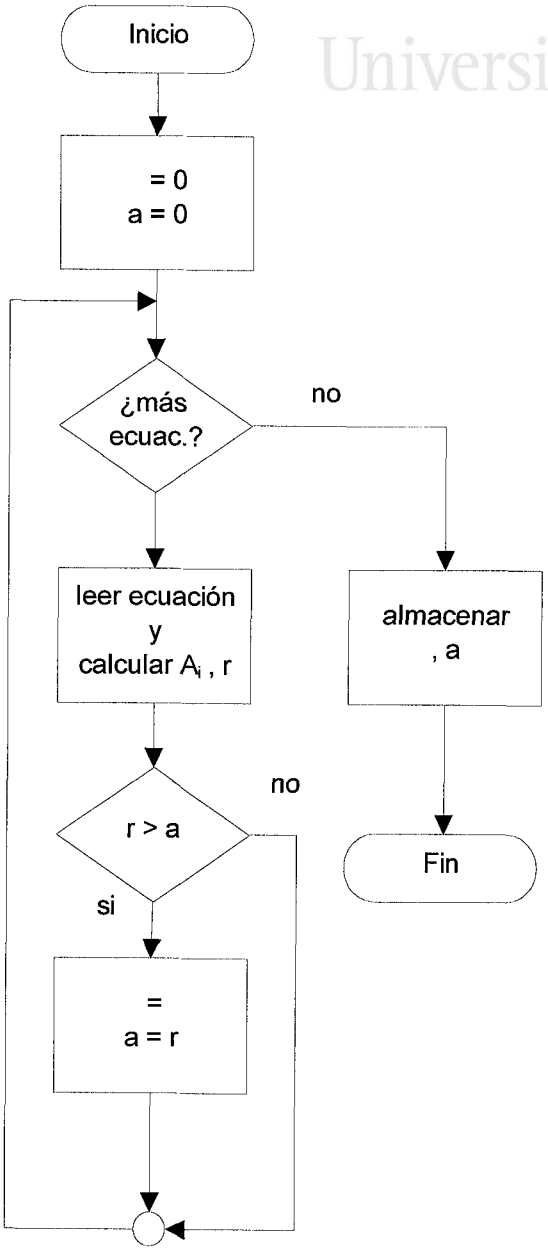


figura 3.13

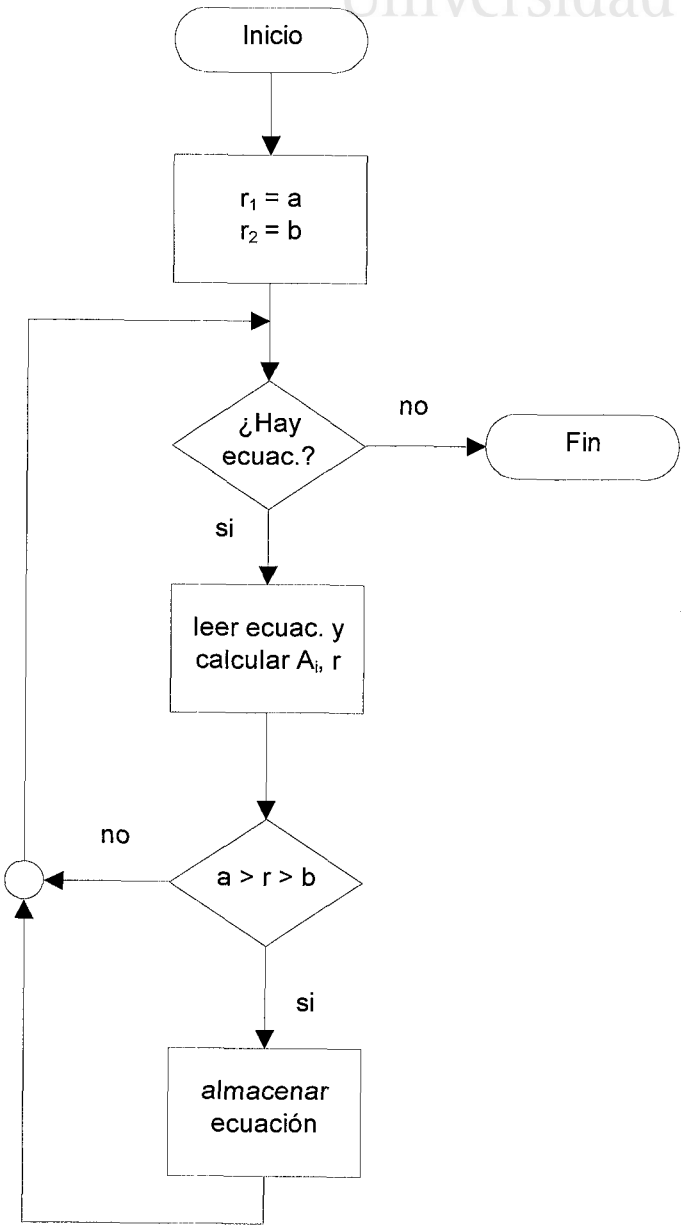


figura 3.14

3.7. GRAMÁTICA GENERATIVA DE LAS ECUACIONES DE ESTADO.

Supongamos que tenemos un sistema de ecuaciones diferenciales $\frac{\partial y_i}{\partial t}$, $\forall$i = 1,2,...., m que representa el estudio de un cierto Modelo Base. Si cada ecuación diferencial depende de n_i variables podemos poner que

$$\frac{\partial y_i}{\partial t} = A_{i_1} + A_{i_2} + ... + A_{i_{ni}} = \sum_{k=1}^{n_i} A_{i_k} = A_i$$

y si estas son modelizadas a partir de ecuaciones de flujo

$$A_{i_k} = \psi_1^{i_k} \oplus \psi_2^{i_k} \oplus .... \oplus \psi_{hk}^{i_k}$$

se obtendrá, para cada ecuación diferencial una expresión matemática como combinación de ecuaciones de flujo

$$\frac{\partial y_i}{\partial t} = \sum_{k=1}^{n_i} \left[\sum_{j=1}^{hk} \psi_j^{ik} \right] = A_i$$

Estas ecuaciones diferenciales reciben el nombre de **ecuación de estado** y para cada una de ellas podemos definir una gramática generativa de la siguiente manera:

Una gramática generativa de ecuaciones de estado, G_S, es una cuádrupla

$$G_S = \langle V_S, V_A, S, P \rangle$$

donde :

V_S es el lenguaje se símbolos terminales formado por las ecuaciones de flujo obtenidas en las gramáticas recognoscitivas de las ecuaciones de flujo más los símbolos + y – de las operaciones aritméticas que unen estas ecuaciones de flujo.

V_A es el alfabeto de símbolos auxiliares, formado por los símbolos

$$V_A = \{S, R_1, Ri, R_{iA}, S, i = 2,3,...,hk\}$$

P es el conjunto de reglas de producción siguiente:

$$S \rightarrow {}^{i_k}_{1} R_1 \qquad R_2 \rightarrow {}^{i_k}_{2} R_{2A} \ \ldots\ldots \qquad R_{hk-1} \rightarrow {}^{i_k}_{hk-1} R_{(hk-1)A} \qquad R_{hk} \rightarrow {}^{i_k}_{hk}$$

$$R_1 \rightarrow +R_2 \qquad R_{2A} \rightarrow +R_3 \ \ldots\ldots\ldots \qquad R_{(hk-1)A} \rightarrow +R_{hk}$$

$$R_1 \rightarrow -R_2 \qquad R_{2A} \rightarrow -R_3 \ \ldots\ldots\ldots \qquad R_{(hk-1)A} \rightarrow -R_{hk}$$

y S es el símbolo inicial.

El diagrama correspondiente puede verse en la figura 3.15

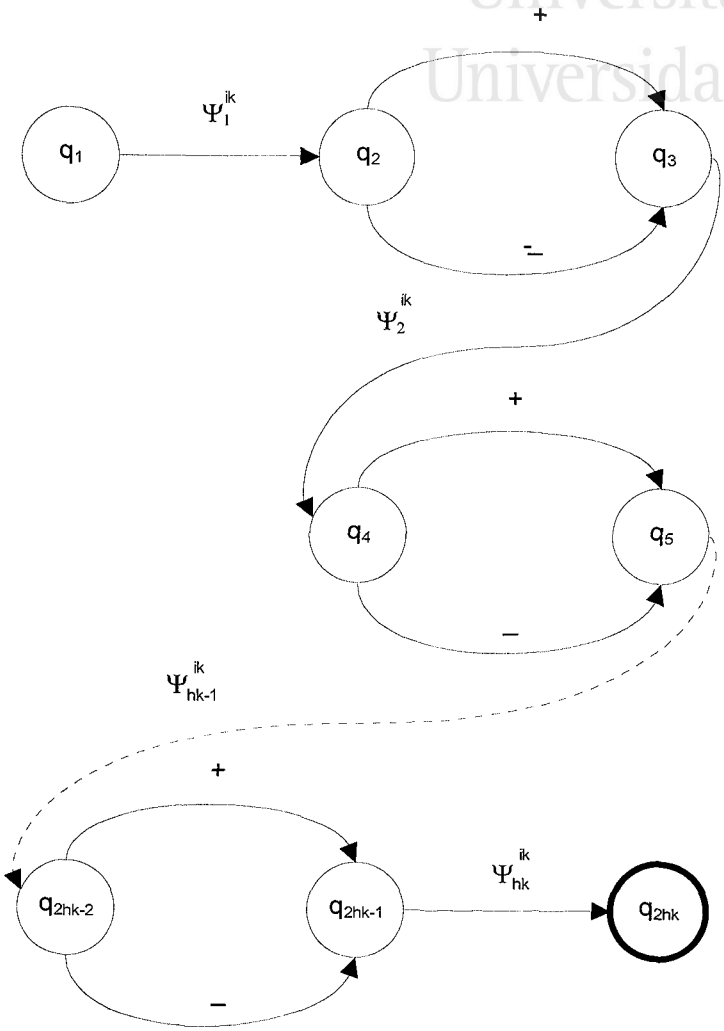


figura 3.15

3.8. GRAMÁTICAS RECOGNOSCITIVAS DE LAS ECUACIONES DE ESTADO.

En este sentido podemos definir, para el sistema de ecuaciones diferenciales, llamado sistema de **ecuaciones de estado**, diversas gramáticas recognoscitivas.

Tres son las gramáticas que vamos a tener en cuenta:

1) Un primer proceso de recognoscibilidad, denotado por GR_{S1} es aquél por el que se concibe el Modelo Base a partir del estudio de un sistema de ecuaciones diferenciales y determinado por la concepción del modelo en sí mismo.

2) Un segundo proceso de recognoscibilidad GR_{S2}, independiente del primero y que determina si la sentencia A_j de la ecuación $\frac{\partial y_j}{\partial t} = A_j$ es o no válida.

Por ello podemos hacer varias consideraciones:

1ª. Fijamos las palabras y las sentencias y si cualquier palabra es correcta, la sentencia también lo es. Este tipo de proceso de recognoscibilidad se determina en el proceso de validación. Esto lo confirma y lo refuerza Jorgensen (1988) en (32) cuando dice: (a) la validación se requiere siempre. (b) Debe intentarse obtener datos para la validación distintos a los utilizados en la calibración. Es importante disponer de un amplio margen de datos de los

Volver al índice/Tornar a l'índex

atributos medibles definidos en los objetivos del modelo (GR_{S1}). (c) Los criterios de validación (GR_{S2}) se formulan en base a los objetivos del modelo y de la cantidad de datos.

2ª. Supongamos que tiene lugar una ligera variación de los datos de un atributo medible. En tal caso, el criterio consiste en cambiar Ψ_{ij} por Ψ'_{ij} donde Ψ'_{ij} es sinónima de Ψ_{ij}. Por tanto pasaríamos de A_j a A'_j , es decir de la sentencia A_j a su sinónima A'_j.

3) Finalmente mostraremos un tercer criterio de recognoscibilidad, denotado por GR_{S3}. Este criterio se basa en las palabras que ha generado la gramática recognoscitiva GR_{F2}. Para cada ecuación de estado, definidas en forma de sentencias, definimos un coeficiente de determinación d_k a partir de los coeficientes de determinación r^k_{ij} de las ecuaciones de flujo que forman la sentencia, en la forma

$$d_k = \frac{\sum_{j=1}^{l_k} r_{ij}^k}{l_k}$$

donde l_k es la longitud de la sentencia o número de ecuaciones de flujo que la constituyen. Al coeficiente d_k le llamaremos **coeficiente complejo de determinación de la ecuación de estado.**

Este coeficiente es el que nos llevará a la elección de la ecuación.

CAPÍTULO 4

CODIFICACIÓN DE LOS LENGUAJES EMPLEADOS EN SISTEMAS COMPLEJOS

CAPÍTULO 4

CODIFICACIÓN DE LOS LENGUAJES EMPLEADOS EN LOS SISTEMAS COMPLEJOS

4.1. INTRODUCCIÓN.

La codificación es un problema esencial en las matemáticas. La codificación nos permite reducir el estudio de unos objetos a otros que resultan más convenientes por su fácil manipulación u otros motivos. Rápidamente nos vienen a la mente códigos utilizados habitualmente como el sistema de numeración decimal que nos permite representar los diferentes números o el sistema de coordenadas cartesianas que nos permite codificar figuras geométricas en expresiones analíticas.

En el capítulo 3 se han obtenido diferentes gramáticas que generan lenguajes que nos permiten construir las ecuaciones matemáticas que modelizan las variables de un sistema. Dependiendo de la propia estructura del sistema y las gramáticas utilizadas por el modelizador, las ecuaciones que resultan pueden tener una expresión analítica compleja.

Al objeto de poder almacenar y manipular un conjunto de estas ecuaciones, es conveniente establecer códigos que simplifiquen dichas expresiones.

Es imprescindible que estos códigos verifiquen ciertas propiedades, entre las que destacaremos las siguientes:

- Decodificación única
- Proceso sencillo de decodificación, e
- Independencia frente a las transformadas o conjunto de funciones elegidas.

En el presente capítulo se aborda, en primer lugar, el estudio de códigos que codifican las funciones transformadas de orden uno y orden dos. **Estos códigos son códigos bloque no singulares**, es decir, son códigos en los que a todos los símbolos del alfabeto fuente (alfabeto de la gramática generativa de funciones transformadas empleado) les corresponde una secuencia fija de símbolos del alfabeto código y estas cadenas son todas ellas distintas. Además, veremos que la codificación de transformadas de orden superior a dos se basa en la utilización de los códigos obtenidos en las transformadas de orden uno y dos.

En segundo lugar se estudia la codificación de las funciones transformadas de orden $n > 2$, observándose que esta extensión de los códigos también es no singular, lo que garantiza que los códigos definidos son unívocamente decodificables.

A continuación se define un código para las ecuaciones matemáticas, ecuaciones de flujo, generadas a partir de las funciones transformadas. Este código utiliza tres vectores para el almacenamiento de la ecuación de flujo. Un vector es el encargado de almacenar los coeficientes reales que multiplican a cada transformada, un segundo vector almacena los códigos de las operaciones que enlazan las diversas transformadas y un tercer vector que almacena los códigos de cada transformada.

4.2. CONCEPTOS GENERALES.

4.2.1. CÓDIGO.

Definición : Sea S = $\{s_1, s_2, ..., s_q\}$ el conjunto de símbolos de un determinado alfabeto. Se define un código como la correspondencia de todas las secuencias posibles de símbolos de S a secuencias de símbolos de otro alfabeto X = $\{x_1, x_2, ..., x_r\}$. S recibe el nombre de **alfabeto fuente** y X **alfabeto código**.

Como la definición de código empleada es muy general, vamos a limitar nuestra atención a códigos que posean ciertas propiedades suplementarias. Para ello se definen los siguientes conceptos.

4.2.2. PROPIEDADES DE LOS CÓDIGOS

Definición 1: Código bloque. La primera propiedad exigida es que el código constituya un **código bloque,** es decir, que asigne cada uno de los símbolos del alfabeto fuente, S, a una secuencia fija de símbolos del alfabeto código X. A cada una de estas secuencias fijas le llamamos **palabra código** y representaremos por X_i a la palabra código que corresponde al símbolo $s_i \in S$.

Definición 2: Código no singular. Es evidente que al utilizar un código bloque han de imponerse también ciertas restricciones; una restricción natural es que todas las palabras código sean distintas, es decir, lo que se denomina un **código no singular**.

Definición 3: Código unívocamente decodificable. Sea $s_{i1}s_{i2}....s_{in}$ una secuencia de n símbolos del alfabeto fuente a la que corresponde una secuencia de símbolos $X_{i1}X_{i2}....X_{in}$ de las palabras código. La **extensión de orden n** de un código bloque que hace corresponder los símbolos s_i con las palabras código X_i es

el código bloque que hace corresponder las secuencias de símbolos $s_{i1}s_{i2}....s_{in}$ con las secuencias de las palabras código $X_{i1}X_{i2}....X_{in}$.

Si la extensión de orden n de un código bloque, para todo n, es no singular, podemos asegurar que el código es **unívocamente decodificable**.

Definición 4: Código instantáneo. Otra restricción que podemos imponer, aunque no es absolutamente necesaria, a los códigos es que sean **instantáneos**, es decir, que puedan decodificarse las palabras de una secuencia sin precisar el conocimiento de los símbolos que las sucedan. Si llamamos **prefijo** de una palabra código $X_i = x_{i1}x_{i2}....x_{im}$ a la secuencia de símbolos $x_{i1}x_{i2}....x_{ij}$ con $j \leq m$ es obvio que una condición necesaria y suficiente para que un código sea instantáneo es que ninguna palabra del código coincida con el prefijo de otra.

Propiedad 1: Si representamos por l_1, l_2, ..., l_q las longitudes de las palabras del código, es decir, el número de símbolos que forman las palabras código, una condición suficiente para que exista un código instantáneo con palabras de longitud l_1, l_2, ..., l_q la obtuvo Kraft en 1949 (36) y la expresó mediante la inecuación

$$\sum_{i=1}^{q} r^{-l_i} \leq 1 \qquad (4.1.)$$

siendo r el número se símbolos del alfabeto código.

El hecho de que la relación anterior también sea válida para los códigos unívocamente decodificables fue probada por McMillan (40) en 1956 y posteriormente por Karush (33) en 1961 que simplificó la demostración como puede verse en el texto "Teoria de la Información y Codificación" de Norman Abramson (45).

La figura 4.1. muestra la ramificación seguida en el árbol para llegar a la subclase correspondiente a los códigos instantáneos.

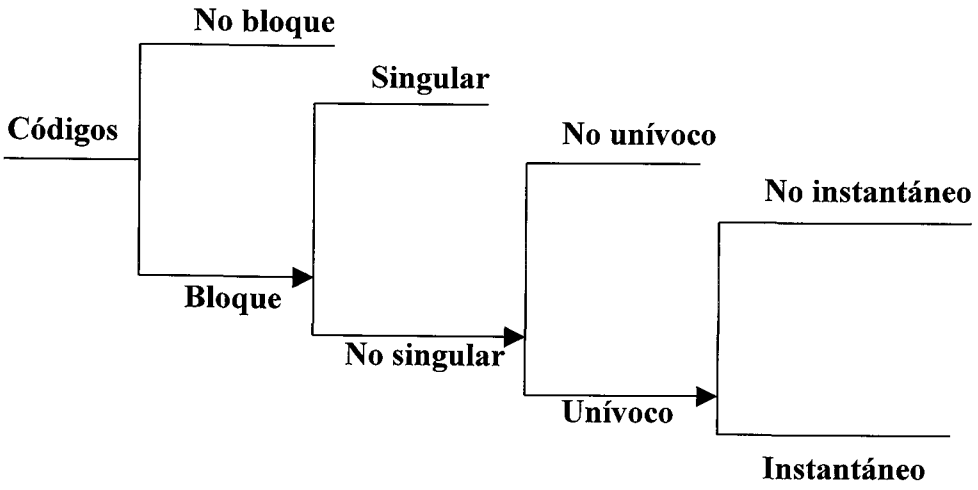


figura 4.1. Subclases de códigos.

Definición 5: Códigos compactos. Como podemos suponer, para un alfabeto fuente y un alfabeto código dados, es posible, sin embargo, elaborar más de un código instantáneo o univocamente decodificable.

De ello se deduce que es importante adoptar un criterio que permita elegir uno de entre ellos. Desde el punto de vista de la mera economía de expresión y la consecuente economía en el equipo de comunicación, sin tener en cuenta otras consideraciones, es preferible un código formado por muchas palabras cortas a uno con palabras de gran longitud.

Si definimos la longitud media de un código por la ecuación

$$L = \sum_{i=1}^{q} P_i \cdot l_i \qquad (4.2.)$$

donde P_i es la probabilidad de que la fuente emita el símbolo i , será interesante encontrar códigos unívocos de longitud media mínima. A estos códigos se les llama **códigos compactos**.

Recordando que la entropía de una fuente de memoria nula, es decir, de una fuente de información en la que los símbolos emitidos, s_1, s_2, ..., s_q son estadísticamente independientes, con probabilidades P_1, P_2, ..., P_q , es

$$H(S) = -\sum_{i=1}^{q} P_i \log P_i \qquad (4.3.)$$

y que si $Q_i \geq 0$, i=1,....,q , para cualquier i , y $\sum_{i=1}^{q} Q_i = 1$ se demuestra en (45) que

$$H(S) \leq -\sum_{i=1}^{q} P_i \log Q_i \qquad (4.4.)$$

y tomando

$$Q_i = \frac{r^{-l_i}}{\sum_{i=1}^{q} r^{-l_i}} \qquad (4.5.)$$

se llega a la conclusión de que

$$H(S) = -\sum_{i=1}^{q} P_i \log P_i \leq -\sum_{i=1}^{q} P_i \log(r^{-l_i}) + \sum_{i=1}^{q} P_i \left(\log \sum_{i=1}^{q} r^{-l_i} \right) \leq L \log r \qquad (4.6.)$$

y expresando la entropía en unidades r-arias, $H_r(S) \leq L$, desigualdad que se convierte en igualdad para $\sum_{i=1}^{q} r^{-l_i}$.

o de otra manera

$$\log_r \frac{1}{P_i} = l_i \text{ para todo i}$$

Resumiendo estas consideraciones, puede decirse que L alcanzará su valor mínimo si pueden elegirse las longitudes de las palabras, l_i, iguales a $\log_r(1/P_i)$.

En otras palabras, la condición de igualdad es que las probabilidades de los símbolos sean de la forma $\left(\frac{1}{r}\right)^{\alpha_i}$ donde α_i es un número entero. Eligiendo $l_i = \alpha_i$ se habrán encontrado las longitudes de las palabras que constituyen un código compacto.

Para el caso de que las probabilidades de los símbolos sean arbitrarias puede utilizarse el algoritmo de Huffman para la obtención de códigos compactos. Este algoritmo se basa en ordenar los símbolos s_1, s_2, ..., s_q de una fuente S cuyas probabilidades son P_1, P_2, ..., P_q, respectivamente, de forma que $P_1 \geq P_2 \geq ..... \geq P_q$. Imaginando que los dos últimos símbolos se confunden en uno sólo se obtiene una nueva fuente con q-1 símbolos. Esta fuente se denomina fuente reducida de S. Los símbolos de esta nueva fuente pueden reordenarse, agrupando de nuevo los dos de menor probabilidad para formar una nueva fuente reducida. Continuando el proceso repetidas veces, de llega a una fuente con dos únicos símbolos. Huffman demostró que el código instantáneo compacto de una de las fuentes de la secuencia se deduce fácilmente conocido el de la fuente inmediata siguiente. Por lo tanto empezando por codificar la última fuente con un código instantáneo compacto (por ejemplo, con el código 0 y 1), se irá ascendiendo hasta encontrar el código instantáneo compacto correspondiente a la fuente original.

Sea S_j una de las fuentes de la secuencia. Uno de sus símbolos, digamos s_a, estará formado por dos símbolos, s_{a0} y s_{a1}, de la fuente precedente S_{j-1}. Todos los demás símbolos de S_j se identifican con los símbolos de S_{j-1}. Según esto, el código instantáneo correspondiente a S_{j-1} se deduce del correspondiente a S_j de acuerdo con la regla siguiente:

"Se asigna a cada símbolo de S_{j-1} (excepto s_{a0} y s_{a1}) la palabra asignada al símbolo de S_j. Las palabras correspondientes a s_{a0} y s_{a1} se forman añadiendo un 0 y un 1, respectivamente, a la palabra asignada a s_a."

4.3. CODIFICACIÓN DE LAS FUNCIONES TRANSFORMADAS

4.3.1. CODIFICACIÓN DE LAS FUNCIONES TRANSFORMADAS DE PRIMER ORDEN.

En el apartado 3.4. definimos el vocabulario principal de la gramática generativa de las ecuaciones transformadas de primer orden como el conjunto

$$V_T = \{f_1, f_2, ..., f_m, (,), x\}$$

Si consideramos este vocabulario como el **alfabeto fuente** podemos definir un **código** como la correspondencia de todas las secuencias posibles de símbolos de $V_T \in L(G_{T1})$ a secuencias de símbolos de algún alfabeto $X = \{x_1, x_2, ..., x_r\}$ al que llamaremos **alfabeto código**.

a) Código binario de longitud constante (CLCT1).

Basándonos en la teoría expuesta en el apartado 5.2.2., podemos realizar la codificación de la gramática generativa de las ecuaciones transformadas de primer orden $V_T = \{f_1, f_2, ..., f_m, (,), x\}$ mediante un código de longitud constante formado por el alfabeto código binario $X = \{0, 1\}$ de la siguiente manera:

Como el número de símbolos del alfabeto fuente es m+3 y el número de símbolos del alfabeto código es 2 aplicando la inecuación de Kraft se debe cumplir que

$$\sum_{i=1}^{m+3} 2^{-l_i} \leq 1 \qquad (4.7.)$$

por lo que

$$\sum_{i=1}^{m+3} 2^{-l_i} = 2^{-l} + 2^{-l} + \overset{m+3}{\ldots\ldots\ldots} + 2^{-l} = (m+3)2^{-l} \leq 1$$

es decir,

$$2^{-l} \leq \frac{1}{m+3} \quad , \quad -l \ln 2 \leq \ln \frac{1}{m+3} \quad , \quad -l \ln 2 \leq -\ln(m+3)$$

por lo que necesitamos que las palabras código tengan una longitud l tal que

$$l \geq \frac{\ln(m+3)}{\ln 2} \qquad (4.8.)$$

<u>Proposición 1:</u> Sea el conjunto $V_T = \{f_1, f_2, \ldots\ldots, f_m, (,), x\}$, al que llamaremos alfabeto fuente y el conjunto $X = \{0, 1\}$ al que llamaremos alfabeto código. Sea además L(X) el conjunto formado por todas las cadenas de longitud n tal que $2^{n-1} \leq m+3 \leq 2^n$. Se puede definir un código unívocamente decodificable como la correspondencia de todas las secuencias posibles de $V_T \in L(G_{T1})$ a secuencias de símbolos de $X \in L(X)$.

Demostración: Consideremos la correspondencia definida mediante la tabla siguiente:

alfabeto fuente	palabras código
f_1	$b_1 = 00....01$ ($\leftarrow n \rightarrow$)
f_2	$b_2 = 00....10$ ($\leftarrow n \rightarrow$)
.....	.....
f_m	$b_m = 00....11$ ($\leftarrow n \rightarrow$)
x	$b_{m+1} = 01....00$ ($\leftarrow n \rightarrow$)
(	$b_{m+2} = 01....10$ ($\leftarrow n \rightarrow$)
)	$b_{m+3} = 01....11$ ($\leftarrow n \rightarrow$)

figura 4.2

donde b_i es la representación en binario del número i utilizando n dígitos.

El código formado constituye un código bloque no singular, pues de la figura 5.2. se desprende que $\forall\ s_i, s_j \in V_T$ si $i \neq j$ se tiene que $b_i \neq b_j$.

Además, como hemos elegido las cadenas $b_i \in L(X)$ de longitud constante, se tiene garantizada que la extensión de cualquier orden del código también forma un código no singular.

De todo ello se deduce que el código es unívocamente decodificable, c.q.d.

También es fácil observar que el código es instantáneo ya que ninguna palabra del código es prefijo de otra.

Ejemplo:

Si el número de funciones es 5 se tendrá que

$$l \geq \frac{\ln(5+3)}{\ln 2} = \frac{\ln 8}{\ln 2} = \frac{\ln 2^3}{\ln 2} = 3$$

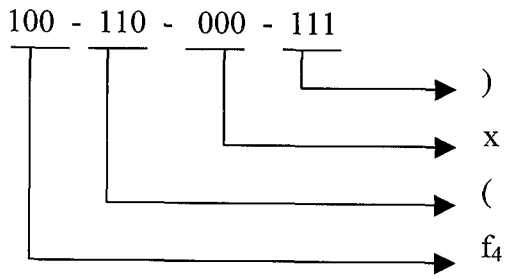
y un código podría ser:

alfabeto fuente	palabras código
f_1	001
f_2	010
f_3	011
f_4	100
f_5	101
x	000
(	110
)	111

figura 4.3

Así el código para $f_4(x)$ será 100110000111

y la decodificación se realiza partiendo la cadena en subcadenas de longitud 3, es decir,



Matemáticamente se puede obtener la decodificación dividiendo por m la cadena sucesivas veces. Los restos de la división y el último cociente nos dan el resultado de la decodificación buscada.

En nuestro ejemplo, como m = 5, el número de símbolos es m+3 = 8 (en binario 1000) por lo que

100110000111 mod 1000 = 111 →)
100110111000 div 1000 = 100110000
100110000 mod 1000 = 000 → x
100110000 div 1000 = 100110
100110 mod 1000 = 110 → (
100110 div 1000 = 100 → f_4

b) Código binario compacto (CBCT1).

Como podemos suponer, para un alfabeto fuente y un alfabeto código dados, es posible, sin embargo, elaborar más de un código instantáneo o univocamente decodificable.

Teniendo en cuenta que la emisión de los símbolos del alfabeto fuente f_1, f_2, ..., f_m es equiprobable, y prescindiendo de la subcadena (x), ya que sabemos

que todas las cadenas terminan igual, podemos construir un código compacto binario aplicando el algoritmo de Huffman (29).

Como ejemplo de aplicación supongamos el caso de 5 funciones

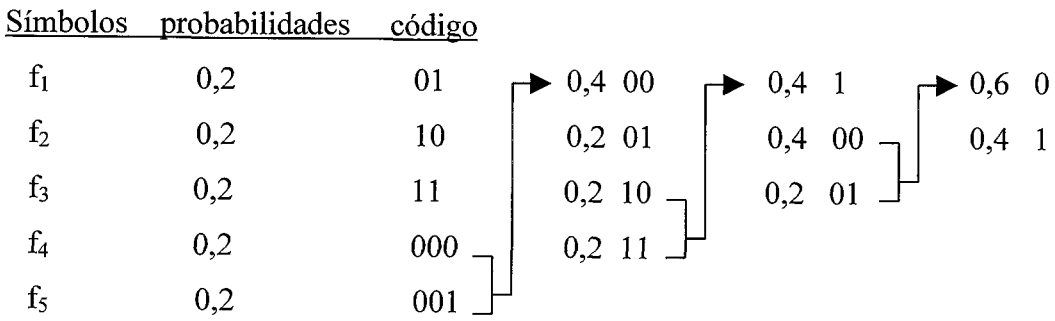


figura 4.4

de donde resulta el siguiente código

alfabeto fuente	palabras código
f_1	01
f_2	10
f_3	11
f_4	000
f_5	001

figura 4.5

Las longitudes medias para este código han resultado ser:

$$L = 2\cdot0,2 + 2\cdot0,2 + 2\cdot0,2 + 3\cdot0,2 + 3\cdot0,2 = 2,4 \text{ binits/símbolo}$$

Para completar este código solamente deberemos anteponer o posponer el código que deseemos emplear para la variable a la que se aplican las funciones.

c) Código según Gödel (CGT1).

A continuación, y para finalizar con la codificación de las funciones transformadas de primer orden podemos ofrecer, a modo de ejemplo, otro tipo de código basado en la técnica que propuso Kurt Gödel, consistente en ordenar los símbolos de la fuente y hacerles corresponder a cada uno de ellos un número natural, es decir, se establece la correspondencia de $S \rightarrow N$ dada por $s_i \rightarrow i$. La codificación de una cadena $x_0x_1...x_n$ se realiza asignándole el número natural $p_0^{y_0}.p_1^{y_1}....p_n^{y_n}$, siendo p_i el i-ésimo número primo e y_i el código correspondiente al símbolo x_i.

Para ello, asociaremos a cada símbolo del alfabeto fuente un número natural:

símbolo :	f_1	f_2	f_3	.....	f_m	(	x	)
número asociado, y :	1	2	3		m	m+1	m+2	m+3

Como la descomposición factorial de un número en sus factores primos es única, la decodificación única está garantizada.

Como ejemplo podemos poner:

símbolo :	(	x	)	f_1	f_2	f_3	f_4	f_5
número asociado :	1	2	3	4	5	6	7	8

La codificación de $f_3(x)$ será

$$2^6 \cdot 3^1 \cdot 5^2 \cdot 7^3 = 1646400$$

4.3.2. CODIFICACIÓN DE LAS FUNCIONES TRANSFORMADAS DE SEGUNDO ORDEN.

En el apartado 4.5. del tema anterior definíamos las funciones transformadas de segundo orden como las expresiones de la forma

$$\phi_i \otimes_n \phi_j$$

siendo $\otimes_n$ una operación entre funciones y $\phi_i = f_i(x)$ en la que x es la variable que aparece en el alfabeto principal de la gramática generativa de las ecuaciones transformadas de segundo orden.

Vamos a exponer tres códigos distintos:

a) **Código binario de longitud constante (CLCT2).**

De la misma manera que codificamos las ecuaciones transformadas de primer orden mediante el código CLCT2, podemos realizar la codificación de la gramática generativa de las ecuaciones transformadas de segundo orden $V_T = \{f_1, f_2, \ldots\ldots, f_m, \otimes_1, \otimes_2, \ldots\ldots, \otimes_k, (\ ,\ ), x\ \}$ mediante un código de longitud constante formado por el alfabeto código binario $X = \{\ 0, 1\}$ de la siguiente manera:

Como el número de símbolos del alfabeto fuente es m+k+3 y el número de símbolos del alfabeto código es 2 aplicando la inecuación de Kraft se debe cumplir que

$$\sum_{i=1}^{m+k+3} 2^{-l_i} \leq 1 \qquad (4.9.)$$

por lo que

$$\sum_{i=1}^{m+k+3} 2^{-l_i} = 2^{-l} + 2^{-l} + \ldots\ldots^{m+k+3} \ldots + 2^{-l} = (m+k+3)2^{-l} \leq 1$$

es decir,

$$2^{-l} \leq \frac{1}{m+k+3} \quad , \quad -l\ln 2 \leq \ln\frac{1}{m+k+3} \quad , \quad -l\ln 2 \leq -\ln(m+k+3)$$

por lo que necesitamos que las palabras código tengan una longitud l tal que

$$l \geq \frac{\ln(m+k+3)}{\ln 2} \qquad (4.10.)$$

<u>Proposición 2:</u> Sea el conjunto $V_T = \{f_1, f_2, \ldots\ldots, f_m, \otimes_1, \otimes_2, \ldots., \otimes_k (,), x\}$, al que llamaremos alfabeto fuente y el conjunto X = {0, 1} al que llamaremos alfabeto código. Sea además L(X) el conjunto formado por todas las cadenas de longitud n tal que $2^{n-1} \leq m+k+3 \leq 2^n$. Se puede definir un código unívocamente decodificable como la correspondencia de todas las secuencias posibles de $V_T \in L(G_{T2})$ a secuencias de símbolos de $X \in L(X)$.

Demostración: Consideremos la correspondencia definida mediante la tabla

alfabeto fuente	palabras código
f_1	$b_1 = \overset{\leftarrow n \rightarrow}{00....01}$
.....	.....
f_m	$b_m = \overset{\leftarrow n \rightarrow}{00....11}$
$\otimes_1$	$b_{m+1} = \overset{\leftarrow n \rightarrow}{01....00}$
.....	.....
$\otimes_k$	$b_{m+k} = \overset{\leftarrow n \rightarrow}{01....10}$
x	$b_{m+k+1} = \overset{\leftarrow n \rightarrow}{11....00}$
(	$b_{m+k+2} = \overset{\leftarrow n \rightarrow}{..1..0..}$
)	$b_{m+k+3} = \overset{\leftarrow n \rightarrow}{..1..1..}$

figura 4.6

donde b_i es la representación en binario del número i utilizando n dígitos.

El código formado constituye un código bloque no singular, pues de la figura 5.6. se desprende que $\forall\ s_i, s_j \in V_T$ si $i \neq j$ se tiene que $b_i \neq b_j$.

Además, como hemos elegido las cadenas $b_i \in L(X)$ de longitud constante, se tiene garantizada que la extensión de cualquier orden del código también forma un código no singular.

De todo ello se deduce que el código es unívocamente decodificable, c.q.d.

También es fácil observar que el código es instantáneo ya que ninguna palabra del código es prefijo de otra.

Ejemplo:

Si el número de funciones es 5 y el número de operaciones 2 se tendrá que

$$l \geq \frac{\ln(5+2+3)}{\ln 2} = \frac{\ln 10}{\ln 2} = 3{,}3219$$

es decir, $l = 4$

y un código podría ser:

alfabeto fuente	palabras código
f_1	0001
f_2	0010
f_3	0011
f_4	0100
f_5	0101
$\otimes_1$	0110
$\otimes_2$	0111
x	1000
(	1001
)	1010

figura 4.7

Así, el código para $f_3(x) \otimes_1 f_2(x)$ será

0011100110001010011000101001100010 10

b) Código binario compacto (CBCT2).

En la codificación que acabamos de estudiar las cadenas que resultan de la codificación de las transformadas de segundo orden son bastante largas. Es por ello, por lo que a continuación estudiamos un código más compacto para codificar dichas funciones transformadas. Y para ello podemos prescindir de la subcadena (x) ya que en todas las transformadas aparece la misma variable.

Si disponemos de m transformadas de primer orden, sabemos que el número mínimo de dígitos necesarios para poder expresar mediante un número en binario cada función es el menor número natural n_1 tal que $2^{n_1} \geq m$ y el número mínimo de dígitos necesarios para poder expresar las k operaciones es el menor número natural n_2 tal que $2^{n_2} \geq k$.

De ahí que el número mínimo de dígitos binarios para poder expresar todas las funciones $f_i \otimes_k f_j$ será n, cumpliéndose

$$2^n \geq m \cdot k \cdot m = m^2 \cdot k$$

Enunciaremos la siguiente proposición:

Proposición 3: Sea el conjunto $V_T = \{f_1, f_2, ..., f_m, \otimes_1, \otimes_2, ..., \otimes_k\}$, al que llamaremos alfabeto fuente y el conjunto X = {0, 1} al que llamaremos alfabeto código. Sea además L(X) el conjunto formado por todas las cadenas de longitud n tal que $2^{n-1} \leq m^2 \cdot k \leq 2^n$. Se puede definir un código unívocamente decodificable como la correspondencia de todas las secuencias posibles de $V_T \in L(G_{T2})$ a secuencias de símbolos de $X \in L(X)$.

Demostración: Consideremos las correspondencias definidas mediante las tablas 4.8 y 4.9.

alfabeto fuente	palabras código
f_1	$a_0 = \overset{\leftarrow n_1 \rightarrow}{00....00}$
f_2	$a_1 = \overset{\leftarrow n_1 \rightarrow}{00....01}$
.....	.....
f_m	$a_{m-1} = \overset{\leftarrow n_1 \rightarrow}{..1..0..}$

figura 4.8

donde a_i es la representación en binario del número i-1 utilizando n_1 dígitos.

alfabeto fuente	palabras código
$\otimes_1$	$b_0 = \overset{\leftarrow n_2 \rightarrow}{00....00}$
$\otimes_2$	$b_1 = \overset{\leftarrow n_2 \rightarrow}{00....01}$
.....	.....
$\otimes_k$	$b_{k-1} = \overset{\leftarrow n_2 \rightarrow}{..1..0..}$

figura 4.9

donde b_i es la representación en binario del número i-1 utilizando n_2 dígitos.

El código que corresponde a la transformada $f_i \otimes_n f_j$ es la representación en binario del número $A = (a_{i-1} \cdot k + b_{n-1}) \cdot m + a_{j-1}$ con n dígitos.

La codificación se realizará del siguiente modo:

$A \bmod m = a_{j-1} \longrightarrow f_j$

$A \operatorname{div} m = B = a_{i-1} \cdot k + b_{n-1}$

$B \bmod k = b_{n-1} \longrightarrow \otimes_n$

$B \operatorname{div} k = a_{i-1} \longrightarrow f_i$

Ejemplo:

Si el número de funciones es m = 5 y el número de operaciones es k = 3 emplearemos el siguiente código

$f_1 = 000$	$\otimes_1 = 00$
$f_2 = 001$	$\otimes_2 = 01$
$f_3 = 010$	$\otimes_3 = 10$
$f_4 = 011$	
$f_5 = 100$	

Entonces, como el número máximo de ecuaciones, de la forma $f_i \otimes_n f_j$, que se pueden formar es $5 \cdot 3 \cdot 5 = 75$, se necesitan números binarios de n = 6 digitos para codificarlas, ya que $2^7 > 75 > 2^6$, y su código sería el que sigue:

$f_1 \otimes_1 f_1 = 0000000$	$f_1 \otimes_2 f_1 = 0000101$	$f_1 \otimes_3 f_1 = 0001010$
$f_1 \otimes_1 f_2 = 0000001$	$f_1 \otimes_2 f_2 = 0000110$	$f_1 \otimes_3 f_2 = 0001011$
$f_1 \otimes_1 f_3 = 0000010$	$f_1 \otimes_2 f_3 = 0000111$	$f_1 \otimes_3 f_3 = 0001100$
$f_1 \otimes_1 f_4 = 0000011$	$f_1 \otimes_2 f_4 = 0001000$	$f_1 \otimes_3 f_4 = 0001101$
$f_1 \otimes_1 f_5 = 0000100$	$f_1 \otimes_2 f_5 = 0001001$	$f_1 \otimes_3 f_5 = 0001110$

$f_2 \otimes_1 f_1 = 0001111$	$f_2 \otimes_2 f_1 = 0010100$	$f_2 \otimes_3 f_1 = 0011001$
$f_2 \otimes_1 f_2 = 0010000$	$f_2 \otimes_2 f_2 = 0010101$	$f_2 \otimes_3 f_2 = 0011010$
$f_2 \otimes_1 f_3 = 0010001$	$f_2 \otimes_2 f_3 = 0010110$	$f_2 \otimes_3 f_3 = 0011011$
$f_2 \otimes_1 f_4 = 0010010$	$f_2 \otimes_2 f_4 = 0010111$	$f_2 \otimes_3 f_4 = 0011100$
$f_2 \otimes_1 f_5 = 0010011$	$f_2 \otimes_2 f_5 = 0011000$	$f_2 \otimes_3 f_5 = 0011101$
$f_3 \otimes_1 f_1 = 0011110$	$f_3 \otimes_2 f_1 = 0100011$	$f_3 \otimes_3 f_1 = 0101000$
$f_3 \otimes_1 f_2 = 0011111$	$f_3 \otimes_2 f_2 = 0100100$	$f_3 \otimes_3 f_2 = 0101001$
$f_3 \otimes_1 f_3 = 0100000$	$f_3 \otimes_2 f_3 = 0100101$	$f_3 \otimes_3 f_3 = 0101010$
$f_3 \otimes_1 f_4 = 0100001$	$f_3 \otimes_2 f_4 = 0100110$	$f_3 \otimes_3 f_4 = 0101011$
$f_3 \otimes_1 f_5 = 0100010$	$f_3 \otimes_2 f_5 = 0100111$	$f_3 \otimes_3 f_5 = 0101100$
$f_4 \otimes_1 f_1 = 0101101$	$f_4 \otimes_2 f_1 = 0110010$	$f_4 \otimes_3 f_1 = 0110111$
$f_4 \otimes_1 f_2 = 0101110$	$f_4 \otimes_2 f_2 = 0110011$	$f_4 \otimes_3 f_2 = 0111000$
$f_4 \otimes_1 f_3 = 0101111$	$f_4 \otimes_2 f_3 = 0110100$	$f_4 \otimes_3 f_3 = 0111001$
$f_4 \otimes_1 f_4 = 0110000$	$f_4 \otimes_2 f_4 = 0110101$	$f_4 \otimes_3 f_4 = 0111010$
$f_4 \otimes_1 f_5 = 0110001$	$f_4 \otimes_2 f_5 = 0110110$	$f_4 \otimes_3 f_5 = 0111011$
$f_5 \otimes_1 f_1 = 0111100$	$f_5 \otimes_2 f_1 = 1000001$	$f_5 \otimes_3 f_1 = 1000110$
$f_5 \otimes_1 f_2 = 0111101$	$f_5 \otimes_2 f_2 = 1000010$	$f_5 \otimes_3 f_2 = 1000111$
$f_5 \otimes_1 f_3 = 0111110$	$f_5 \otimes_2 f_3 = 1000011$	$f_5 \otimes_3 f_3 = 1001000$
$f_5 \otimes_1 f_4 = 0111111$	$f_5 \otimes_2 f_4 = 1000100$	$f_5 \otimes_3 f_4 = 1001001$
$f_5 \otimes_1 f_5 = 1000000$	$f_5 \otimes_2 f_5 = 1000101$	$f_5 \otimes_3 f_5 = 1001010$

Para explicar su decodificación tomaremos como ejemplo la ecuación 0101000.

0101000 mod 101 = 000 que corresponde a f_1
Además 0101000 div 101 = 1000 , por lo que
1000 mod 11 = 10 que corresponde a $\otimes_3$
y como 1000 div 11 = 010 que corresponde a f_3

se tendrá finalmente que $0101000 \rightarrow f_3 \otimes_3 f_1$.

c) <u>Código basado en Gödel</u>

Siguiendo el mismo procedimiento que el utilizado en el apartado 5.2.1. podemos poner para 5 funciones y 3 operaciones el siguiente código

símbolo :	(	x	)	f_1	f_2	f_3	f_4	f_5	$\otimes_1$	$\otimes_2$	$\otimes_3$
número asociado :	1	2	3	4	5	6	7	8	9	10	11

con lo que la codificación de la ecuación $f_2(x) \otimes_2 f_3(x)$ será

$$2^5 \cdot 3^1 \cdot 5^2 \cdot 7^3 \cdot 11^{10} \cdot 13^6 \cdot 17^1 \cdot 19^2 \cdot 23^3 =$$
$$= 769541345821883405315794769520 0$$

4.3.3. CODIFICACIÓN DE LAS FUNCIONES TRANSFORMADAS DE TERCER ORDEN.

En el apartado 3.3.3. del tema anterior definíamos las funciones transformadas de tercer orden como las expresiones de la forma

$$\phi_i \otimes_m (\phi_j \otimes_n \phi_k) \text{ ó } (\phi_i \otimes_m \phi_j) \otimes_n \phi_k$$

es decir, $T_1 \otimes T_2$ ó $T_2 \otimes T_1$ donde T_1 representa una función transformada de primer orden y T_2 una función transformada de segundo orden.

Por tanto, es lógico pensar que la codificación de las ecuaciones transformadas de orden 3 puede realizarse a partir de los resultados obtenidos en la codificación de las transformadas de orden uno y dos.

a) Código binario (CBT3).

Como hemos visto anteriormente, seguiremos llamando $f_1, f_2, ..., f_m$ a las m funciones de variable real utilizadas como funciones transformadas de primer orden y $\otimes_1, \otimes_2, ..., \otimes_k$ a las k operaciones definidas entre las funciones anteriores.

Para la codificación de las m funciones transformadas de primer orden utilizaremos el número binario i-1 (expresado con n dígitos, siendo $2^n \geq m$) para representar la función f_i y para las transformadas de segundo orden el código CBCT2.

Además como las operaciones que intervienen en las ecuaciones transformadas de tercer orden no son, en general, conmutativas, necesitamos para su decodificación algo que nos indique si la ecuación es de la forma $T_1 \otimes T_2$ ó $T_2 \otimes T_1$. Esto lo podemos solucionar añadiendo un bit (0 ó 1) al comienzo de la cadena codificada.

Veamos un ejemplo práctico con las 5 funciones y 3 operaciones utilizadas en el apartado anterior.

Sea la ecuación $f_2(x) \otimes_1 (f_2(x) \otimes_2 f_4(x))$. Prescindiendo de los paréntesis y la variable, pues no van a ser necesarias, como a:

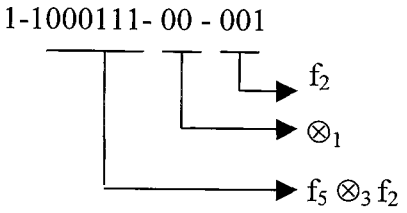
f_2 le corresponde el código 001
$\otimes_1$ le corresponde 00
$f_2 \otimes_2 f_4$ le corresponde 0010111

y empleando el dígito 0 como primer dígito de la cadena codificada para indicar que la ecuación es de la clase $T_1 \otimes T_2$ resulta que

$$f_2(x) \otimes_1 (f_2(x) \otimes_2 f_4(x)) \rightarrow 0001000010111$$

Sea, ahora la cadena 1100011100001 su decodificación es como sigue:

La cadena empieza con un 1, luego es de la clase $T_2 \otimes T_1$ por lo que



y por tanto

$$1100011100001 \rightarrow (f_5 \otimes_3 f_2) \otimes_1 f_2$$

b) Código basado en Gödel

Para este código no es necesario añadir ningún concepto teórico nuevo a los vistos anteriormente y bastará con un ejemplo.

Observando la codificación empleada en el apartado anterior, la codificación de $f_2(x) \otimes_1 (f_2(x) \otimes_2 f_4(x))$ será

$$2^5 \cdot 3^1 \cdot 5^2 \cdot 7^3 \cdot 9^9 \cdot 11^1 \cdot 13^5 \cdot 17^1 \cdot 19^2 \cdot 23^3 \cdot 29^{10} \cdot 31^7 \cdot 37^1 \cdot 41^2 \cdot 43^3 \cdot 47^3$$

4.3.4. CODIFICACIÓN DE LAS FUNCIONES TRANSFORMADAS DE ORDEN N.

a) Código basado en CLCT1 y CBCT2.

Todas las funciones transformadas de orden $n \geq 3$ pueden codificarse a partir de la codificación de las transformadas de orden uno y dos con la adición, en los códigos, de los paréntesis necesarios debidos a los tipos de las operaciones utilizadas en la construcción de las funciones transformadas de orden n y que deberá estudiarse para cada caso en particular.

Así, por ejemplo si tuvieramos que codificar transformadas de orden 4 procederíamos de la siguiente manera:

Representemos por $T_i = \{T_i^1, T_i^2, \ldots\ldots\}$ el conjunto de todas las transformadas de orden i, i = 1,2,3,4 y por $\otimes$ cualquier operación utilizada. Cada transformada de orden 4 puede descomponerse en transformadas de orden uno y orden 2, como se indica en la figura 4.10, obteniendo así, diferentes tipos de transformadas.

En el conjunto de transformadas de orden 4, T_4 , podemos establecer la siguiente relación:

Sean T_4^i, T_4^j y T_4^k son funciones transformadas cualesquiera de T_4. Diremos que **"dos funciones T_4^i, T_4^j están relacionadas si en su descomposición en transformadas de orden uno y dos son del mismo tipo".**

Representando por M la relación "pertenecer al mismo tipo" puede verse fácilmente que M es una relación de equivalencia, ya que verifica las propiedades

1) Reflexiva: T_4^i M T_4^i , $\forall i$
2) Simétrica: Si T_4^i M T_4^j entonces T_4^j M T_4^i , $\forall i,j$.
3) Transitiva: Si T_4^i M T_4^j y T_4^j M T_4^k entonces T_4^i M T_4^k , $\forall i,j, k$.

quedando el conjunto T_4 dividido en clases de equivalencia. En la figura 5.10 pueden observarse 8 clases de equivalencia, aunque en los casos prácticos alguna

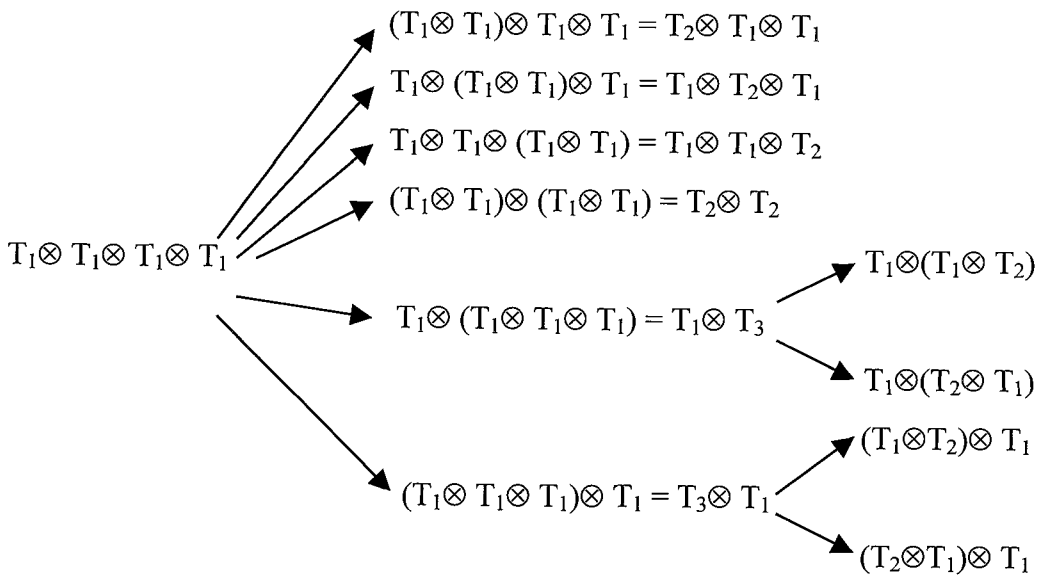


figura 4.10

de ellas podría no existir dependiendo de las propiedades de las operaciones utilizadas.

Ejemplo práctico de codificación:

Sea el caso de 5 funciones f_1, f_2, f_3, f_4, f_5 y 3 operaciones $\otimes_1$, $\otimes_2$, $\otimes_3$.

Empleando los códigos de los apartados anteriores y añadiendo al principio de la cadena codificada un código que indique la clase de transformada se tendrá:

Si los códigos del tipo de transformada son :

$T_2 \otimes T_1 \otimes T_1 = 000$

$T_1 \otimes T_2 \otimes T_1 = 001$

$T_1 \otimes T_1 \otimes T_2 = 010$

$T_2 \otimes T_2 = 011$

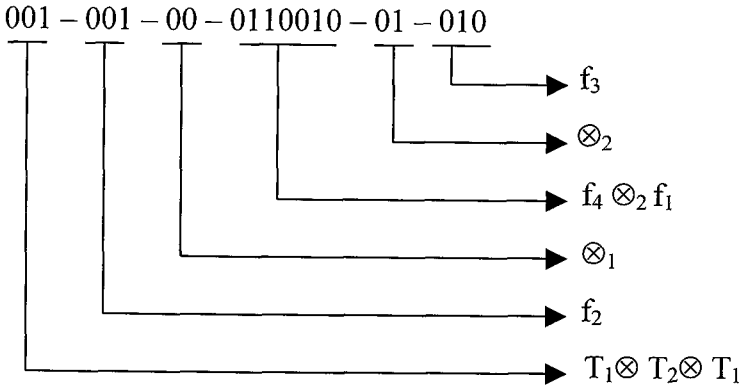
$T_1 \otimes (T_1 \otimes T_2) = 100$

$T_1 \otimes (T_2 \otimes T_1) = 101$

$(T_1 \otimes T_2) \otimes T_1 = 110$

$(T_2 \otimes T_1) \otimes T_1 = 111$

la codificación de $f_2(x) \otimes_1 (f_4(x) \otimes_2 f_1(x)) \otimes_2 f_3(x)$ será



b) Código CBTn.

El proceso de codificación utilizado en el apartado anterior puede no ser demasiado útil para extenderlo a transformadas de orden elevado, ya que en su descomposición en transformadas de orden uno y dos el conjunto cociente que se obtiene, T_n/M, posee demasiadas clases de equivalencia.

Por este motivo, y debido q que en una función transformada de cualquier orden sólo aparecen transformadas de primer orden, operadores y paréntesis (prescindimos de la variable) nos inclinamos por utilizar el método de Huffman para la creación de códigos binarios en la que consideraremos que todos los símbolos del alfabeto fuente son equiprobables

Como ejemplo vamos a realizar la codificación del caso de cinco transformadas de primer orden, f_1, f_2, f_3, f_4 y f_5 dos operadores $\otimes_1$ y $\otimes_2$.

Sím.	prob.	cód.														
f_1	1/9	**001**	2/9	000	2/9	11	2/9	10	2/9	01	3/9	00	4/9	1	5/9	0
f_2	1/9	**010**	1/9	001	2/9	000	2/9	11	2/9	10	2/9	01	3/9	00	4/9	1
f_3	1/9	**011**	1/9	010	1/9	001	2/9	000	2/9	11	2/9	10	2/9	01		
f_4	1/9	**100**	1/9	011	1/9	010	1/9	001	2/9	000	2/9	11				
f_5	1/9	**101**	1/9	100	1/9	011	1/9	010	1/9	001						
$\otimes_1$	1/9	**110**	1/9	101	1/9	100	1/9	011								
$\otimes_2$	1/9	**111**	1/9	110	1/9	101										
(	1/9	**0000**	1/9	111												
)	1/9	**0001**														

Así si f_1= sen(x) , f_2 = exp(x), f_3 = ln(x), $f_4 = x^2$, f_5 = 1/x, $\otimes_1$ = +, $\otimes_2$ = o

la codificación de la función sen(x)+1/x+ln(x^2) sería, prescindiendo de la variable será

$$f_1+f_5+(f_3 o f_4) \rightarrow 001110101110000001111111000001$$

y la decodificación de 001111101110011 es

$$001\text{-}111\text{-}101\text{-}110\text{-}011 \rightarrow f_1 o f_5+f_3 \rightarrow \text{sen}(1/x)+\ln(x)$$

4.4. CODIFICACIÓN DE LAS ECUACIONES DE FLUJO.

En la sección 3.5. definíamos una ecuación de flujo como toda expresión de la forma

$$\varphi = A_0 \oplus_{k_1} A_1 t_{i_1}^{j_1} \oplus_{k_2} A_2 t_{i_2}^{j_2} \oplus_{k_3} \ldots\ldots \oplus_{k_n} A_n t_{i_n}^{j_n}$$

donde

φ es una variable de flujo,

$A_0, A_1, \ldots\ldots, A_n$ son coeficientes reales,

$\oplus_{k_1}, \oplus_{k_2}, \ldots\ldots, \oplus_{k_n}$ son operadores aritméticos y

$t_{i_1}^{j_1}, t_{i_2}^{j_2}, \ldots\ldots, t_{i_n}^{j_n}$ son funciones transformadas

y donde los superíndices indican la variable independiente de cada transformada.

Suponiendo que ya tenemos codificados todos los operadores aritméticos y todas las funciones transformadas mediante alguno de los procesos explicados en los apartados anteriores, la codificación de una ecuación de flujo se reduce al almacenamiento de los coeficientes, operadores y funciones transformadas en tres arrays paralelos unidimensionales

$$A = \begin{pmatrix} A_0 \\ A_1 \\ \dots \\ \dots \\ A_n \end{pmatrix} \qquad \oplus = \begin{pmatrix} \oplus_{k_1} \\ \oplus_{k_2} \\ \dots \\ \dots \\ \oplus_{k_n} \end{pmatrix} \qquad t = \begin{pmatrix} t_{i_1}^{j_1} \\ t_{i_2}^{j_2} \\ \dots \\ \dots \\ t_{i_n}^{j_n} \end{pmatrix}$$

Ejemplo.

Sea φ una variable de flujo que depende de 4 variables x_1, x_2, x_3 y x_4.

Las funciones transformadas de orden uno utilizadas son

$f_1 = \text{sen}(x)$, $f_2 = \exp(x)$, $f_3 = \ln(x)$, $f_4 = x^2$ y $f_5 = 1/x$

y los operadores son $\otimes_1 = +$ (suma) y $\otimes_2 = *$ (producto).

La codificación de la función

$$\varphi = 3 + 2 \cdot 1/x_1 \cdot \text{sen}\, x_1 + 0{,}5x_2^2 - \ln x_3 + 4{,}3\ e^{x_4} .1/x_4$$

podría realizarse de la siguiente manera:

Codificación de los operadores:

$$+ = 0 \ , \quad * = 1$$

Codificación de las variables:

$$x_1 = 00, \quad x_2 = 01, \quad x_3 = 10 \quad \text{y} \quad x_4 = 11$$

Codificación de las transformadas: Para las transformadas de orden dos y utilizando el código CBCT2 se tiene que

$$1/x \cdot \text{senx} = f_5 \otimes_2 f_1 \rightarrow 0101101$$

$$e^x \cdot 1/x = f_2 \otimes_2 f_5 \rightarrow 0011101$$

y para las transformadas de orden uno, utilizando el código CBCT1 se tiene que

$x^2 = f_4 \rightarrow 000$

$\ln x = f_3 \rightarrow 11$

resultando que φ se codifica mediante los arrays siguientes:

$$A = \begin{pmatrix} 3 \\ 2 \\ 0{,}5 \\ -1 \\ 4{,}3 \end{pmatrix} \qquad \oplus = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad t = \begin{pmatrix} 0101101 \\ 000 \\ 11 \\ 0011101 \end{pmatrix}$$

CAPÍTULO 5

APLICACIÓN:

SUBMODELO

REPRODUCTIVO

DEL

MODELO MARIOLA

CAPÍTULO 5

APLICACIÓN: SUBMODELO REPRODUCTIVO DEL MODELO MARIOLA

5.1. INTRODUCCIÓN.

El estudio que vamos a realizar está basado en un caso concreto de modelización matemática publicado en la revista Ecological Modelling (ver 56, Villacampa et al., 1999) y realizado por un grupo de investigadores entre los que se encuentran el autor del presente proyecto y el director del mismo.

En este apartado exponemos de forma resumida los objetivos que se persiguen con el estudio y modelización de un subsistema (subsistema reproductivo) que forma parte del modelo MARIOLA que modeliza un ecosistema terrestre concreto.

La erosión, y el proceso de desertización, una de sus obvias consecuencias, es uno de los principales problemas medioambientales de los países mediterráneos. Grandes áreas de antiguos bosques se han convertido en zonas de arbustos, consideradas como la fase final de un proceso de degradación.

Sin embargo, las tierras arbustivas mediterráneas juegan un papel importante en la protección del suelo frente a los procesos de erosión hídrica,

especialmente en áreas orográficas caracterizadas por sus grandes pendientes, muy dañadas por precipitaciones irregulares muy intensas y localizadas.

Si fuésemos capaces de predecir el comportamiento de un ecosistema de arbustos frente a variaciones climáticas, dispondríamos de una poderosa herramienta para tomar medidas preventivas sobre la planificación y protección del suelo.

El modelo MARIOLA ((54)Usó-Domenech et al. 1995; (57)Villacampa et al. 1997) apareció como consecuencia de estas necesidades. Dicho modelo está basado en un ecosistema terrestre montañoso localizado en la Sierra de Mariola (Alicante), y es una simulación del comportamiento y evolución de un típico ecosistema de arbustos de las zonas mediterráneas.

El modelo MARIOLA consiste en los siguientes submodelos:

1. Submodelo de crecimiento: crecimiento, defoliación y destrucción de la biomasa.

2. Submodelo de la descomposición de la biomasa perdida.

3.Submodelo reproductivo :formación de brotes, floración y fructificación.

Este modelo fue desarrollado para el arbusto mediterráneo Cistus Albidus L. ((54)Usó-Domenech et al., 1997). Las interacciones entre arbustos mediterráneos en el contexto del modelo MARIOLA también ha sido estudiado en (52)(Mateu et al., 1997).

El arbusto mediterráneo Cistus Albidus L. se encuentra abundantemente a lo largo de toda la Comunidad Valenciana. El tipo biológico corresponde a la Nanofanerifita con un tamaño comprendido entre los 40 y los 120 cm. Su período de floración corresponde en su mayor parte al período comprendido entre Abril y Junio aunque puede aparecer una primera floración en Marzo y la última a finales de Julio. Los fenómenos reproductivos dependen de las características climáticas donde se localizan las especies vegetales, pero también dependen de determinados factores como el nitrógeno y el magnesio aunque es muy dificil calcular valores numéricos para estas variables.

Para determinar el número de brotes, flores y frutos ha sido necesario un proceso de conteo visual y que en el caso del Cistus Albidus L. el proceso es relativamente fácil ya que los brotes, flores y frutos, así como las flores que se marchitan y caen se detectan fácilmente.

La zona experimental ha sido la Sierra de Mariola, en Agres, provincia de Alicante, UTM = 30SYH19, con una altitud de 850 m.con un seguimiento de las características climáticas basado en 16 semanas de observación : temperatura media anual de 14,1 ºC con unos valores extremos de 6,5 ºC en Enero y 23,1ºC en Agosto, una pluviometría anual de 600 mm. y una media de evapotranspiración de 757 mm. El substrato geológico es de piedras calizas del período cretáceo sobre las cuáles están situadas plataformas calcáreas con una reforestación de Pinus halepensis.

5.2. HIPÓTESIS INICIALES.

Para realizar el estudio del submodelo reproductivo comenzaremos fijando los siguientes conceptos:

1. El conjunto de p-símbolos.

El conjunto de p-símbolos o atributos medibles que van a ser tenidos en cuenta (formado por elementos climáticos y otros elementos referidos a la planta) se exponen, junto con los símbolos que se utilizan para su representación en la tabla 5.1.

Atributos Medibles	Descripción
M	**Humedad del aire(%)**
NHS	**Número de horas de insolación**
T	**Temperatura (°C)**
PLU	**Precipitación (l/m^2)**
WS	**Velocidad del viento (km/h)**
TB	**Biomasa(verde +leñosa, g)**
NGEM	**Número de brotes o yemas**
NFLOR	**Número de flores**
NFRUT	**Número de frutos**

Tabla 5.1

2. Conjunto de funciones transformadas de primer orden.

Las gramáticas generativas de las funciones transformadas que vamos a desarrollar están basadas en el siguiente conjunto de frunciones reales de variable real, y que llamaremos conjunto de funciones transformadas de primer orden

$$\{T^1(x)\} = \{x^2, \exp(0{,}1x), \exp(-0{,}1x), \operatorname{arctg}(x), \cos(x), \frac{1}{x}, \sqrt{x}, \ln(x)\} \quad (5.1.)$$

3. Conjunto de variables de flujo.

El conjunto de variables de flujo, así como los símbolos que emplearemos para representarlas se exponen en la tabla 5.2.

Variables de flujo	Descripción
CRGEM	crecimiento de brotes(número)
CRFLOR	crecimiento de flores(número)
CRFRUT	crecimiento de frutos(número)
DGEM	destrucción de brotes(número)
DFLOR	destrucción de flores(número)
DFRUT	destrucción de frutos(número)
GERM	germinación(número)

Tabla 5.2

Estas variables de flujo son funciones que dependen de ciertos atributos medibles y que el criterio del modelizador ha definido de la siguiente forma:

$$\begin{aligned}
&CRGEM(t) = f(M, PLU/100, TB)\\
&CRFLOR(t) = f(M, T, PLU/100, NHS, NGEM)\\
&CRFRUT(t) = f(M, T, PLU/100, NHS, NFLOR)\\
&DGEM(t) = f(M, T, PLU/100, NGEM, WS)\\
&DFLOR(t) = f(M, T, PLU/100, WS, NFLOR)\\
&DFRUT(t) = f(M, T, PLU/100, WS, NFRUT)\\
&GERM(t) = f(M, T, PLU/100, NFRUT, NHS)
\end{aligned} \qquad (5.2)$$

4. Variables y ecuaciones de estado.

El sistema de ecuaciones diferenciales se ha elaborado siguiendo el primer proceso de recognoscibilidad GR_{S1}, resultando ser

$$\begin{cases}
\dfrac{dNGEM(t)}{dt} = CRGEM(t) - DGEM(t) - CRFLOR(t)\\
\dfrac{dNFLOR(t)}{dt} = CRFLOR(t) - DFLOR(t) - CRFRUT(t)\\
\dfrac{dNFRUT(t)}{dt} = CRFRUT(t) - DFRUT(t) - GERM(t)
\end{cases} \qquad (5.3.)$$

5. Vocabularios de primer y segundo orden.

Los vocabularios de primer orden, VPO, de los p-símbolos son:

$$V^1_M,\ V^1_T,\ V^1_{PLU},\ V^1_{WS},\ V^1_{NFRUT},\ V^1_{NHS},\ V^1_{NGEM},\ V^1_{TB}\ \text{y}\ V^1_{NFLOR} \qquad (5.4.)$$

y únicamente están formados por las transformadas de orden cero y orden uno.

También se han utilizado vocabularios de segundo orden, VSO, formados por el producto de dos p-símbolos

$$V^2_{M,T},\ V^2_{M,PLU},\ V^2_{M,WS},\ \ldots\ldots\ldots\ldots,\ V^2_{TB,NFLOR} \qquad (5.5.)$$

por lo que el t-léxico que se genera es

$$\text{t-L} = \{\ V^1_{M},\ V^1_{T},\ V^1_{PLU},\ \ldots\ldots\ldots,V^1_{NFLOR},\ V^2_{M,T},\ V^2_{M,PLU},\ V^2_{M,WS},\ \ldots\ldots\ldots,\ V^2_{TB,NFLOR}\ \} \qquad (5.6.)$$

5.3. GRAMÁTICA GENERATIVA DE LAS FUNCIONES TRANSFORMADAS.

Con los atributos medibles tenidos en cuenta y las funciones reales empleadas, definimos las **gramáticas generativas de las funciones transformadas** de la siguiente manera:

1. **Vocabulario principal:** Es el conjunto formado por los siguientes símbolos, donde x_i representa un p-símbolo de la tabla 1.

$$V_{Ti} = \{x_i, sqr, \exp, \text{arctg}, \cos, sqrt, \ln, (,), 1, 0.1, -, \cdot, /\} \qquad (5.7.)$$

2. Vocabulario auxiliar: Es el conjunto formado por los siguientes símbolos, donde S es el símbolo inicial:

$$V_A = \{S, R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8, R_9, R_{10}, R_{11} \qquad (5.8.)$$

3. Conjunto de producciones o reglas de formación: Es el conjunto de reglas que se exponen a continuación:

$$P = \left\{ \begin{array}{l} S \to x_i, S \to f_j R_1, S \to \exp R_2, S \to 1R_3, S \to x_i R_4, R_1 \to (R_5, \\ R_5 \to x_i R_6, R_6 \to), R_2 \to (R_7, R_7 \to -R_8, R_7 \to 0.1R_9, R_8 \to 0.1R_9, \\ R_9 \to \cdot R_5, R_3 \to / R_{10}, R_{10} \to x_i, R_4 \to \cdot R_{11}, R_{11} \to x_k \end{array} \right\} \qquad (5.9.)$$

donde f_j representa una función de $\{T^1(x)$ - {exp(0.1x) , 1/x} y $k > i$.

4. Reconocedor finito de las gramáticas generativas de las funciones transformadas.

Debido a que todas las leyes de formación son de la forma

$$A \to aB \quad \text{ó} \quad A \to a$$

con A, B $\in V_A$ y a $\in V_T$ las gramáticas que hemos definido son regulares.

La expresión regular de las cadenas generadas por estas gramáticas viene dada por la expresión

$$\alpha = x_i + f_j(x_i) + \exp([-0.1 + 0.1]x_i) + 1/x_i + x_i \cdot x_k \qquad (5.10)$$

siendo su diagrama de Moore el correspondiente a la figura 5.1

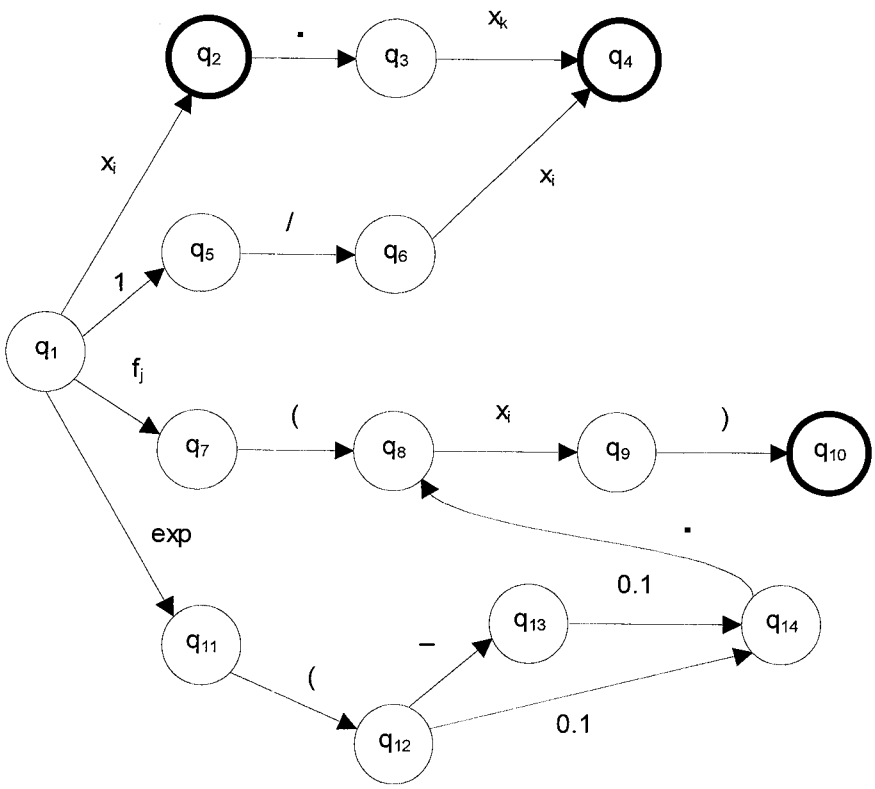


figura 5.1

5.4. GRAMATICAS GENERATIVA Y RECOGNOSCITIVA DE LAS ECUACIONES DE FLUJO.

Con los lenguajes que se obtienen con las gramáticas generativas de las funciones transformadas diseñadas en el apartado anterior vamos a construir las gramáticas de las ecuaciones de flujo.

En esta aplicación necesitamos obtener expresiones matemáticas para modelizar siete variables de flujo: CRGEM, DGEM, CRFLOR, DFLOR, CRFRUT, DFRUT y GERM, como hemos indicado anteriormente en la tabla 6.2. Estas expresiones matemáticas son las que conocemos con el nombre de ecuaciones de flujo.

Para su construcción se ha decidido que cada ecuación de flujo esté formada por la suma de cinco funciones transformadas.

Nuestro objetivo es, pues, crear gramáticas generativas en las que los lenguajes generados por ellas contengan "palabras" (ecuaciones de flujo) sean de la forma

$$\Psi = A_0 + \sum_{i=1}^{5} A_i \cdot t_i \qquad (5.11)$$

donde por t_i representamos cualquier transformada obtenida en las gramáticas generativas del apartado anterior y los A_i, i = 0..5 son coeficientes numéricos.

Las gramáticas generativas de las ecuaciones de flujo tendrán la siguiente estructura:

1. Vocabulario principal:

Está formado por los coeficientes reales, las funciones transformadas y los operadores aritméticos que intervienen en las ecuaciones.

$$V_T = \{A_0, A_1, A_2, A_3, A_4, A_5, t_1, t_2, t_3, t_4, t_5, +, \cdot\} \qquad (5.12)$$

2. Vocabulario auxiliar:

Está formado por el símbolo inicial S y todos los símbolos necesarios para formar las reglas de producción de las cadenas

$$V_A = \{S, R_{4i}, R_{4i+1}, R_{4i+2}, R_{4i+3}, i = 0,1, 2, 3, 4\} \qquad (5.13)$$

3. Conjunto de producciones o reglas de formación:

Vienen dadas por el siguiente conjunto

$$P = \left\{ \begin{array}{l} S \to A_0 R_0, R_{4i} \to +R_{4i+1}, R_{4i+1} \to A_{i+1} R_{4i+2}, \\ R_{4i+2} \to \cdot R_{4i+3}, R_{4i+3} \to t_{i+1} R_{4i+4}, R_{16} \to +R_{17}, \\ R_{17} \to A_5 R_{18}, R_{18} \to \cdot R_{19}, R_{19} \to t_5 \end{array} \right\} \qquad (5.14)$$

con i = 0, 1, 2, 3.

4. Reconocedor finito de las gramáticas generativas de las ecuaciones de flujo

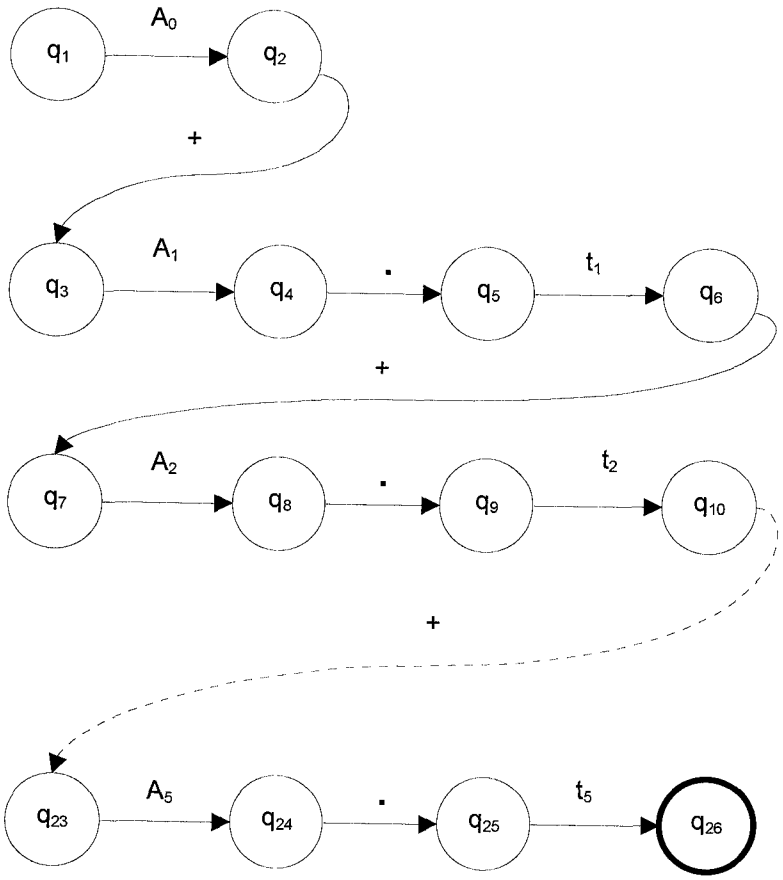


figura 5.2

5. Gramática recognoscitiva de las ecuaciones de flujo.

A partir de los datos experimentales obtenidos a lo largo de 18 semanas de observación y aplicando el primer criterio de recognoscibilidad GRF_1 con la

ayuda del programa REGRESSUS (6) se han obtenido las siguientes ecuaciones de flujo:

$$CRGEM = 11937827736.6333\left(\frac{PLU}{100}\right)^2 + 8887432236.4781\left(\frac{1}{M}\right) + \\ + 25968866013.6976\cos\left(\frac{PLU}{100}\right) + 8888766569.492\,\text{arctg}\,M - \\ - 39931297044 \qquad (5.15)$$

$$DGEM = 12800952588.6678\left(\frac{PLU}{100}\right)^2 + 9529999838.4237\left(\frac{1}{M}\right) + \\ + 27846458313.5398\cos\left(\frac{PLU}{100}\right) - 40.4298\cos WS + \\ + 9531439387.3246\,\text{arctg}\left(\frac{PLU}{100}\right) + 42818396553.4922 \qquad (5.16)$$

$$CRFLOR = 24538061248.2749\left(\frac{PLU}{100}\right)^2 + 18268010624.3174\frac{1}{M} + \\ + 0.0001\exp(0.1\cdot NGEM) + 53378691559.5850\cdot\cos\left(\frac{PLU}{100}\right) + \\ + 18270752691.6942\cdot\text{arctg}\,M - 82078300505.1208 \qquad (5.17)$$

$$DFLOR = -2885250093.0345 \cdot \left(\frac{PLU}{100}\right)^2 - 2148000863.9357 \cdot \left(\frac{1}{M}\right) - \\ - 6276407624.4184 \cdot \cos\left(\frac{PLU}{100}\right) - 2148323388.5355 \cdot \text{arctg}(M) + \\ + 2.0595 \cdot \sqrt{NFLOR} + 9650983489.3318 \qquad (5.18)$$

$$CRFRUT = -1.8245 \cdot \left(\frac{PLU}{100}\right)^2 - 13582666169.5563 \cdot \left(\frac{1}{M}\right) - \\ - 33.9305 \cdot \exp(-0.1 \cdot NFLOR) - 39688258767.6687 \cdot \cos\left(\frac{PLU}{100}\right) - \\ - 13584716165.7817 \cdot \text{arctg}(M) + 61027064334.4555 \qquad (5.19)$$

$$DFRUT = 0.005 \cdot T \cdot NFRUT + 0.0175 \cdot \left(\frac{PLU}{100}\right) \cdot NFRUT - \\ - 0.0161 \cdot WS \cdot NFRUT - 0.0002 \cdot NFRUT^2 + \\ + 1.4215 \cdot \exp(-0.1 \cdot NFRUT) - 1.4132 \qquad (5.20)$$

$$GERM = -0.0133 \cdot T \cdot NHS + 21436656856828.5 \cdot NFRUT \cdot NHS - \\ - 0.144616 \cdot NHS^2 - 300113195995598.25 \cdot NFRUT - \\ - 2.0866 \cdot \text{arctg}(NFRUT) + 45.4497 \qquad (5.21)$$

cuyos coeficientes de correlación son los que se exponen en la tabla 5.3 que se expone a continuación:

Ecuaciones	Coeficientes
CRGEM	0.8141
DGEM	0.7909
CRFLOR	0.9916
DFLOR	0.9238
CRFRUT	0.9932
DFRUT	0.9814
GERM	0.9993

tabla 5.3

Finalmente, el proceso de recognoscibilidad RG_{S2} es determinado mediante el proceso de validación , cuyos gráficos se exponen en las figuras 5.3, 5.4 y 5.5. En estas figuras, el trazo negro corresponde a los datos reales y el rojo al obtenido por integración numérica de las ecuaciones de estado.

Terminaremos esta sección obteniendo los coeficientes complejos de determinación de cada ecuación de estado (Criterio de recognoscibilidad GR_{s3})

$$d_{NGEM} = \frac{0.8141 + 0.7909 + 0.9916}{3} = 0.8655 \qquad (5.22)$$

$$d_{NFLOR} = \frac{0.9916 + 0.9238 + 0.9932}{3} = 0.9695 \qquad (5.23)$$

$$d_{NFRUT} = \frac{0.9932 + 0.9814 + 0.9993}{3} = 0.9913 \qquad (5.24)$$

figura 5.3. Validación del número de brotes

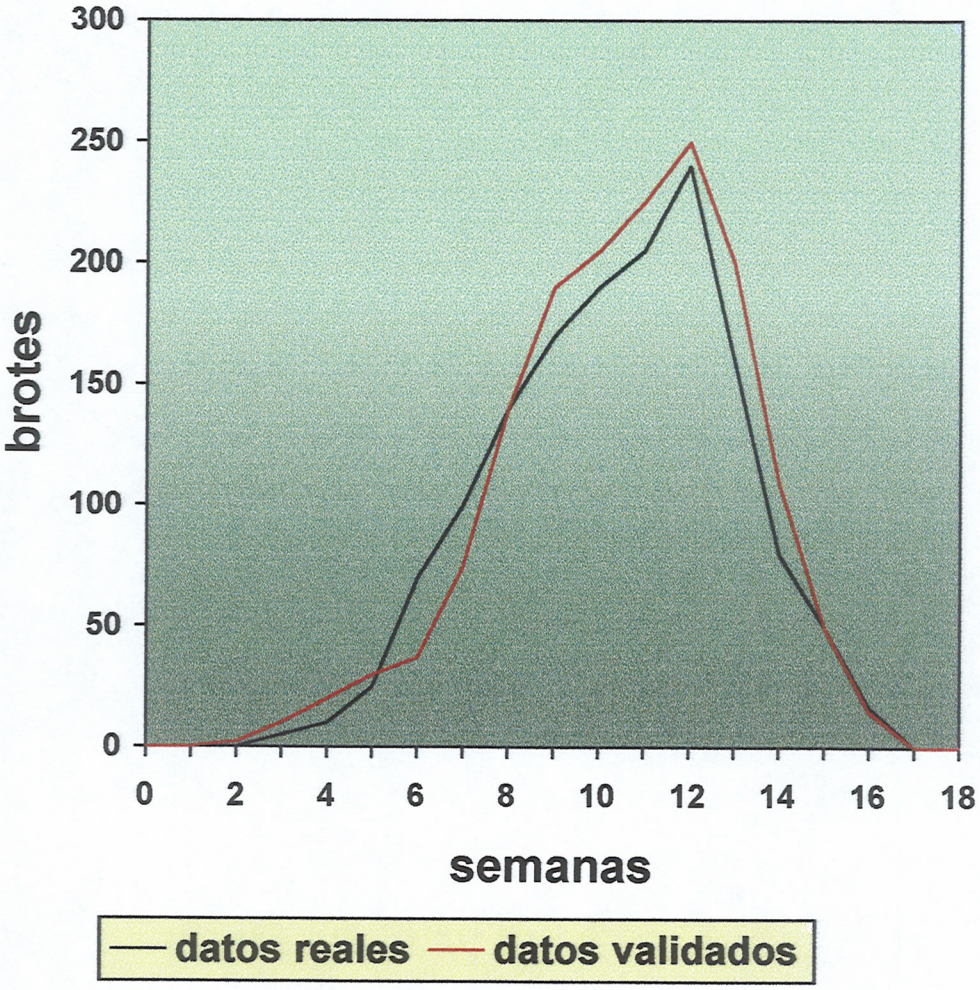


figura 5.4. Validación del número de flores

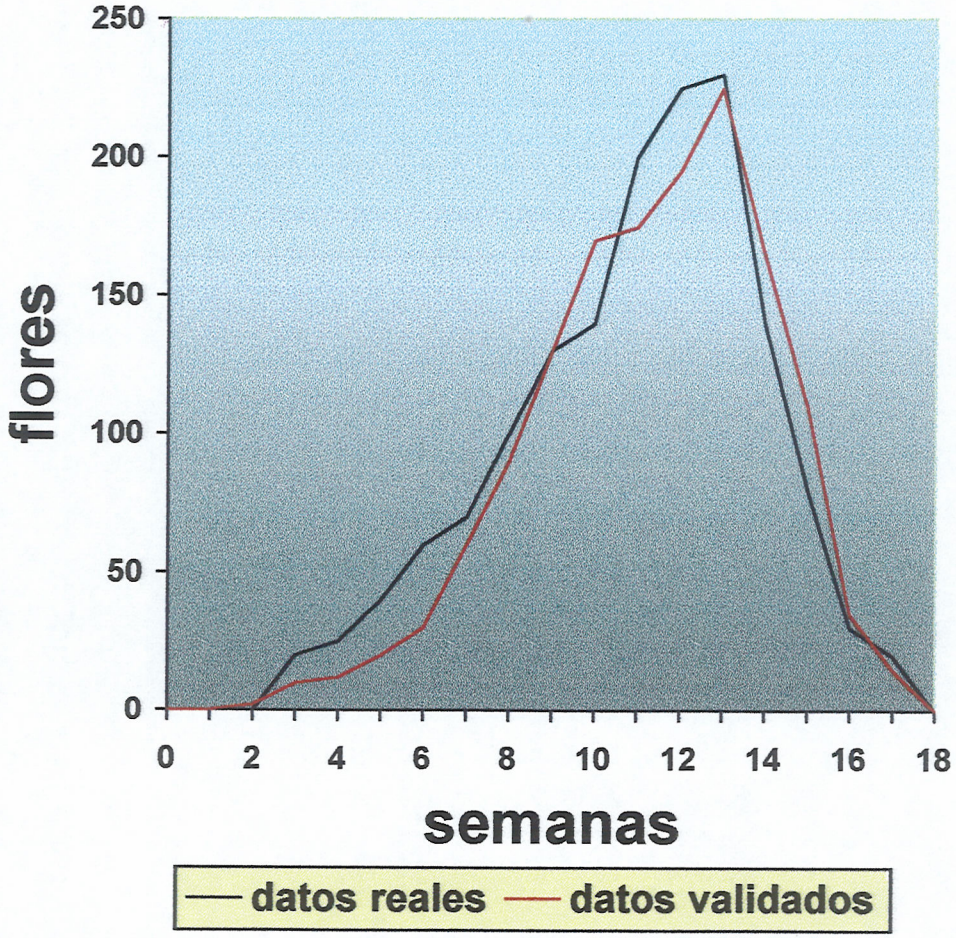
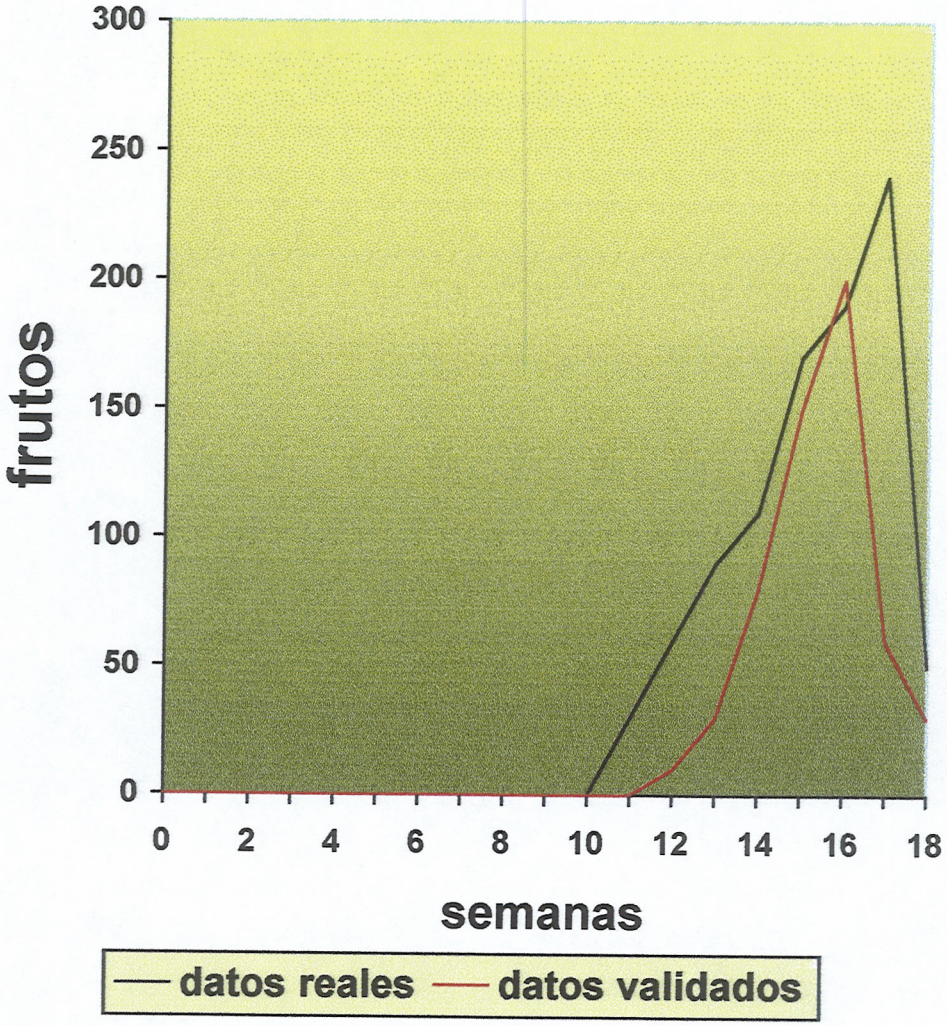


figura 5.5. Validación del número de frutos



5.5. CODIFICACIÓN DE LAS FUNCIONES TRANSFORMADAS Y DE LAS ECUACIONES DE FLUJO.

En el apartado 5.4. vimos que la ecuaciones de flujo tienen una expresión matemática de la forma $\Psi = A_0 + \sum_{i=1}^{5} A_i \cdot t_i$ donde las t_i son variables, transformadas de primer orden de alguna de las variables representadas en la tabla 5.1., es decir, elementos de los vocabularios de primer orden de estas variables, o están formadas por el producto de dos cualesquiera de esas variables, o lo que es lo mismo, elementos de los vocabularios de segundo orden. Nuestro objetivo es, en primer lugar, codificar estas t_i. Al tener nueve variables, ocho transformadas de primer orden, y añadiendo la función identidad $f_0(x) = x$ se podrán codificar, como se indica en las tablas 6.4 y 6.5, de la siguiente manera:

Número	variable	Código
0	M	0000
1	NHS	0001
2	T	0010
3	PLU/100	0011
4	WS	0100
5	TB	0101
6	NGEM	0110
7	NFLOR	0111
8	NFRUT	1000

Tabla 5.4

Número	Función	Código
0	x	0000
1	x^2	0001
2	exp(0.1x)	0010
3	exp(-0.1x)	0011
4	arctg(x)	0100
5	cos(x)	0101
6	1/x	0110
7	$\sqrt{x}$	0111
8	ln(x)	1000

Tabla 5.5

Y como se pueden formar un total de 81 expresiones diferentes, $f_i(x_j)$, se podrán codificar con los números binarios comprendidos entre el 0 y el 80 expresados con siete dígitos de manera que

$$\text{Código de } f_i(x_j) = [9i + j]_{(2)}$$

Un resumen de la codificación de las funciones transformadas, $f_i(x_j)$, se representa, a continuación, en la tabla 5.6.

$f_i(x_j)$	Código
$f_0(x_0) = M$	0000000
$f_0(x_1) = NHS$	0000001
.....	.....
$f_0(x_8) = NFRUT$	0001000
$f_1(x_0) = M^2$	0001001
$f_1(x_1) = NHS^2$	0001010
.....	.....
$f_3(x_2) = exp(-0.1T)$	0011101
.....	.....
.....	.....
$f_8(x_8) = ln(NFRUT)$	1010000

Tabla 5.6

Las expresiones de la forma $x_i \cdot x_j$ se codifican también con siete dígitos binarios de forma que

$$\text{Código de } x_i \cdot x_j = [10i+j]_{(2)}$$

Un resumen de la codificación de las funciones transformadas, $f_i(x_j)$, se representa, a continuación, en la tabla 5.7.

$x_i x_j$	Código
$x_0 \cdot x_1 = M \cdot NHS$	0000001
$x_0 \cdot x_2 = M \cdot T$	0000010
.....	.....
$x_4 \cdot x_8 = WS \cdot NFRUT$	0110000
.....	.....
.....	.....
$x_7 \cdot x_8 = NFLOR \cdot NFRUT$	1001110

Tabla 5.7

Los códigos que hemos establecido para la codificación de las expresiones $f_i(x_j)$ y $x_i \cdot x_j$ no son unívocamente decodificables ya que las cadenas binarias pueden tener una doble interpretación. Así, por ejemplo, con la cadena 0001011 se tiene que

0001100 mod 1001 = 0010 que corresponde a $x_2 = T$
0001100 div 1001 = 0011 que corresponde a $f_3(x) = \exp(-0.1x)$
por lo que la cadena 0001100 puede interpretarse como exp(-0.1T)

pero también

0001100 mod 1010 = 0001 que corresponde a $x_1 = NHS$
0001011 div 1010 = 0010 que corresponde a T
por lo que la cadena 0001011 puede interpretarse como NHS · T

Este problema queda solucionado añadiendo un octavo bit a la izquierda de la cadena que indique la clase de función, pudiendo ser

0 para indicar a las funciones del tipo $f_i(x_j)$ y
1 para indicar al tipo $x_i \cdot x_j$

De esta manera, y volviendo al ejemplo, se tendrá que

la cadena 00001100 corresponde a exp(-0.1·T) y
la cadena 10001100 corresponde a NHS · T

quedando así terminada la codificación de las funciones transformadas.

La codificación de las ecuaciones de flujo se realiza fácilmente a partir de la codificación de la codificación de las transformadas y puede expresarse en forma matricial ya que

$$\Psi = A_0 + \sum_{i=1}^{5} A_i \cdot t_i = \begin{pmatrix} A_0 & A_1 & A_2 & A_3 & A_4 & A_5 \end{pmatrix} \begin{pmatrix} 1 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{pmatrix}$$

En la gramática recognoscitiva del apartado 4 hemos obtenido siete ecuaciones de flujo. Su codificación es la siguiente:

1ª. $\psi_1 = CRGEM = (A_0 \quad A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5)\begin{pmatrix} 1 \\ 00001100 \\ 00110110 \\ 00110000 \\ 00100100 \\ 0 \end{pmatrix}$

con A_0 = -39931297044 A_1 = 11937827736.6333 A_2 = 8887432236.4781
A_3 = 25968866013.6976 A_4 = 8888766569.492 A_5 = 0

2ª. $\psi_2 = DGEM = (A_0 \quad A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5)\begin{pmatrix} 1 \\ 00001100 \\ 00110110 \\ 00110000 \\ 00110001 \\ 00100111 \end{pmatrix}$

con A_0 = 42818396553.4922 A_1 = 12800952588.6678 A_2 = 9529999838.4237
A_3 = 27846458313.5398 A_4 = -40.4298 A_5 = 42818396553.4922

3ª. $\psi_3 = CRFLOR = (A_0 \quad A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5)\begin{pmatrix} 1 \\ 00001100 \\ 00110110 \\ 00110000 \\ 00100100 \\ 01000110 \end{pmatrix}$

con A_0 = -82078300505.1208 A_1 = 24538061248.2749 A_2 = 12268010624.3174
A_3 = 0.0001 A_4 = 53378691559.585 A_5 = 18279752691.6942

4ª. $\psi_4 = DFLOR = (A_0 \quad A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5)\begin{pmatrix} 1 \\ 00001100 \\ 00110110 \\ 00110000 \\ 00100100 \\ 01000110 \end{pmatrix}$

con A_0 = 9650983489.3318 A_1 = -2885250093.0345 A_2 = -2148000863.9357
A_3 = -6276407624.4184 A_4 = -2148323388.5355 A_5 = 2.0595

5ª. $\psi_5 = CRFRUT = (A_0 \quad A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5)\begin{pmatrix} 1 \\ 00001100 \\ 00110110 \\ 00100010 \\ 00110000 \\ 00100100 \end{pmatrix}$

con A_0 = 61027064334.4555 A_1 = –1.8245 A_2 = -13582666169.5563
A_3 = -33.9305 A_4 = -39688258767.6687 A_5 = 13584716165.7817

6ª. $\psi_6 = DFRUT = (A_0 \quad A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5)\begin{pmatrix} 1 \\ 10011100 \\ 10100110 \\ 10110000 \\ 00010001 \\ 00100011 \end{pmatrix}$

con $A_0 = -1.4132$ $A_1 = 0.005$ $A_2 = 0.0175$
$A_3 = -0.0161$ $A_4 = -0.0002$ $A_5 = 1.4215$

7ª. $\psi_7 = GERM = (A_0 \quad A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5)\begin{pmatrix} 1 \\ 10001100 \\ 10010010 \\ 00001010 \\ 00001000 \\ 00101100 \end{pmatrix}$

con $A_0 = 45.4497$ $A_1 = -0.0133$ $A_2 = 21436656856828.5$
$A_3 = -0.144616$ $A_4 = -300113195995598.25$ $A_5 = -2.0866$

con lo que queda terminado el proceso de la codificación.

Para la codificación de estas ecuaciones hemos realizado un programa cuyo listado en PASCAL se ofrece a continuación:

Programa para la codificación de las ecuaciones de flujo

```
program codificar;
uses crt;
const A0 = 0;
      B0 ='1';
type tipoA = array[0..5] of real;
     tipoB = array[1..5] of integer;
     tipoflujo =array[1..5] of string[7];
var A:tipoA;
    B:tipoB;
    ecuacion:tipoflujo;
    nomecuacion:string[7];
    codigo:string[7];
    digito:integer;
    c,d,i,j,transformada,funcion,x1,x2:integer;
(*****************************************)
procedure menu;
begin
   writeln('             FUNCIONES                    VARIABLES ');
   writeln('             ================================== ');
   writeln;
   writeln('          0 --> f(x)=x              0 --> M   ');
   writeln('          1 --> x^2                 1 --> NHS ');
   writeln('          2 --> exp(0.1x)           2 --> T   ');
   writeln('          3 --> exp(-0.1x)          3 --> PLU/100 ');
   writeln('          4 --> arctg(x)            4 --> WS  ');
   writeln('          5 --> cos(x)              5 --> TB  ');
   writeln('          6 --> 1/x                 6 --> NGEM ');
   writeln('          7 --> sqrt(x)             7 --> NFLOR');
   writeln('          8 --> ln(x)               8 --> NFRUT');
   writeln('          9 --> x*y                           ');
   gotoxy(20,17);
   writeln('Pulse INTRO para continuar ......');
   readln;
end;
(*****************************************)
procedure datos;
begin
  gotoxy(20,17);
  writeln('                                                ');
  gotoxy(10,14);
  writeln('________________________________________________  ');
  gotoxy(20,15);
  writeln('Introducci¢n de datos');
  writeln('                   ====================');
  write('Nombre de la ecuaci¢n de flujo : '); readln(nomecuacion);
```

```
   write('coeficiente A0 = ');readln(A[0]);
   for i:=1 to 5 do
    begin
      gotoxy(1,18);
      writeln('                                ');
      writeln('                                ');
      writeln('                                ');
      writeln('                                ');
      gotoxy(1,18);
      write('coeficiente A',i, '= ');readln(A[i]);
      write('funci¢n = ');readln(funcion);
      if funcion=9 then
         begin
           codigo[0]:='1';
           write('primera variable = '); readln(x1);
           write('segunda variable = '); readln(x2);
           transformada:=10*x1+x2;
         end
       else
         begin
          codigo[0]:='0';
            write('variable = ');readln(x1);
            transformada:=9*funcion+x1;
         end;
       B[i]:=transformada;
       for j:=1 to 7 do
       codigo[j]:='0';
       c:=B[i];
       j:=7;
       repeat
       d:=c mod 2;
       if d = 0 then codigo[j]:='0' else codigo[j]:='1';
       c:=c div 2;
       j:=j-1;
       until c<2;
       if transformada = 0 then codigo[j]:='0' else codigo[j]:='1';
          for j:=0 to 7 do
          ecuacion[i][j]:=codigo[j];
   end;
end;
(*****************************************)
begin
clrscr;
menu;
datos;
clrscr;
writeln('La ecuaci¢n de flujo ',nomecuacion,' tiene el siguiente
c¢digo:');
writeln;
```

Volver al índice/Tornar a l'índex


```
writeln('matriz de coeficientes : ');
for i:=0 to 5 do
writeln(A[i]);
writeln;
writeln('matriz de transformadas : ');
writeln;
writeln(B0:5);
for i:=1 to 5 do
  begin
  for j:= 0 to 7 do
    write(ecuacion[i][j]);
    writeln;
  end;
readln
end.
```

CONCLUSIONES

CONCLUSIONES

Hemos elaborado un estudio sobre la modelización de sistemas complejos desde el punto de vista de la lingüística matemática y la teoría de la codificación.

Esto ha sido posible ya que *"todos los modelos tienen en común que encierran experiencia y siempre envuelven signos, señales, sintaxis, semántica y una habilidad para descifrar y obtener resultados", (Patten 42).*

En nuestra metodología utilizamos, para modelizar procesos, un sistema de ecuaciones diferenciales

$$y_i^{'} = g_i(x_{i1}, x_{i2,}.....,x_{in}) \qquad \text{i} = 1,2,...., \text{n}$$

llamadas ecuaciones de estado.

Sus variables, $x_{i1}, x_{i2,}.....,x_{in}$, llamadas variables de flujo, son, a su vez funciones, F_j, llamadas funciones de flujo, que dependen de otras variables

$$x_{ij} = F_j(\varphi_1, \varphi_2,....., \varphi_m)$$

Estas variables, $\varphi_1, \varphi_2,....., \varphi_m$, reciben el nombre de funciones transformadas, que se obtienen a partir de unas funciones reales de variable real, elegidas por el modelizador y aplicadas sobre unos determinados atributos medibles del sistema.

Todas las ecuaciones y funciones que aparecen en el proceso de modelización tiene que ser expresadas matemáticamente disponiendo de datos experimentales de los atributos medibles.

Para la obtención de las funciones transformadas hemos comenzado definiendo los siguientes conceptos:

- t-alfabeto de orden i, i = 1,.....,n de un p-símbolo
- vocabulario de primer orden de un p-símbolo
- vocabulario de orden n de n p-símbolos
- t-léxico, d-léxico y léxico de un Sistema Complejo.

Con estos conceptos, hemos sentado las bases para poder construir Gramáticas Generativas de funciones transformadas. Las palabras de los lenguajes generados por estas gramáticas son las funciones transformadas.

Hemos conseguido que las reglas de las gramáticas cumplan las condiciones necesarias y suficientes para que sean gramáticas regulares, ya que así se tiene garantizado que:

- Los lenguajes pueden expresarse mediante una expresión regular.
- Se dispone de un procedimiento que permite la obtención de un reconocedor finito del lenguaje.
- Se dispone de un modelo de procedimiento para reconocimiento de cadenas.
- Pueden ser utilizados para especificar la construcción de analizadores léxicos.
- Se pueden traducir de forma sencilla a un código en un lenguaje de programación.

Con las funciones transformadas y un conjunto de operadores aritméticos hemos formado el vocabulario de las Gramáticas Generativas de las Ecuaciones

de Flujo. Gramáticas que también son regulares y cuyos lenguajes generan las ecuaciones de flujo.

Hemos construido Gramáticas Recognoscitivas de ecuaciones de flujo que permiten seleccionar aquellas ecuaciones que el modelizador considera importantes, de acuerdo con ciertos criterios preestablecidos que denominamos criterios de recognoscibilidad.

Finalmente hemos construido Gramáticas Generativas y Recognoscitivas de ecuaciones de estado que nos permiten obtener y seleccionar aquellas ecuaciones de estado que mejor modelizan los procesos.

Como las ecuaciones obtenidas en los procesos de modelización suelen tener expresiones analíticas complicadas, hemos ideado códigos que simplifican la escritura de las funciones transformadas y de las ecuaciones de flujo con objeto de paliar en lo posible el problema de su almacenamiento y posterior manipulación.

Los estudios realizados han sido aplicados a un caso concreto de modelización matemática con las siguientes características:

- Se han construido únicamente Gramáticas Generativas de funciones transformadas de primer orden.
- Se han utilizado vocabularios de primer y segundo orden.
- Las Gramáticas generativas de las ecuaciones de flujo crean ecuaciones de flujo con cinco funciones transformadas y el único operador $\oplus$ empleado es la suma.
- Se ha seguido el criterio de recognoscibilidad GRF_1 para la selección de las ecuaciones de flujo.

- Para la elaboración de las ecuaciones de estado se han seguido los criterios de recognoscibilidad GRS_1 y GRS_2.
- Cada ecuación de flujo se ha codificado en dos arrays unidimensionales, uno para los coeficientes y el otro para las funciones transformadas.
- Hemos utilizado códigos binarios para las funciones transformadas.

Para futuras investigaciones serán estudiadas nuevas gramáticas recognoscitivas que nos permitan el análisis de las ecuaciones que mejor estudian los procesos frente a variaciones de los datos.

Serán aplicadas las diversas gramáticas estudiadas en esta memoria a casos concretos de modelos de sistemas complejos, comparando los resultados obtenidos.

Serán utilizados los códigos para la manipulación de ecuaciones obtenidas a partir de pequeñas variaciones de los datos experimentales

Los códigos utilizados abren las puertas a futuras investigaciones en un intento de generalizar la codificación a todo tipo de ecuaciones, ya que pueden ser muy variadas debido a los criterios en los que se base el modelizador.

Asimismo se generalizará computacionalmente la codificación de las ecuaciones para cualquier gramática considerada.

BIBLIOGRAFIA

BIBLIOGRAFÍA

(1) ARBIB M.A. Theories of abstract automata. Prentice Hall,1969.

(2) BALCI O.: Requeriments for models development techniques. Comput. Oper. res. 13.53-67. 1986.

(3) BRUCE H., DEPEW D., SMITH J.: Entropy, Information and Evolution: new perspectives on Physical and Biological Evolution. A bradford book. The MIT Press. Cambridge, Massachussetts. London, England. 1988.

(4) CAMPBELL N.R.: La medición: La forma del pensamiento matemático. Grijalbo.Barcelona. 1974.

(5) CARNAP R.:Einführung in die Symbolische Logik. Wien. 1973.

(6) CASELLES A.,USÓ J.L.:REGRESSUS: A finder of functional relationships in a system. Advances in Support System Research. International Institute for Advanced Studies in System Research and Cybernetics. Windsor. Canadá. 1991

(7) CHECKLAND P.B.: Systems Theory, Systems Practice. N.Y. John Wiley. 1981.

(8) CHEN Z.: Analogy, Systems and Intelligence. Cybernetics and Systems. An international journal, 22, 611-616. 1991.

(9) CHOMSKY N.: Aspects of the theory of syntax. MitPress. Cambridge. 1965.

(10) CHOMSKY N.: Some basic concepts of linguistic.Handbook of Mathematical Psychology. Vol II. John Wiley. N.Y. 1963.

(11) CHOMSKY N.: Syntactic Structures. Mouton.La Haye. 1969.

(12) CHOMSKYN.:The logical structure of linguistic theory. Plenum Press,N.Y.,1975

(13) DAGET J.: Les modeles matematiques en ecologia. Masson. París. 1976.

(14) DAVIES R., O´KEEFE. Simulation modelling with Pascal. Englewood Cliffs. Prentice Hall. 1989.

(15) DAVIS M.,: Computability, complexity and languages: fundamentals of computers science(2° edi.) Academic Press,1994

(16) FERNÁNDEZ G., SÁEZ F.: Fundamentos de Informática. Anaya,1995

(17) FEYS R., FITCH F.B. Dictionary of symbols of mathematical logic. North-Holland Publishing Company. Amsterdam. 1979.

(18) FORRESTER J.: Industrial Dinamics.Massachussets Institute of Technology. Press., Cambridge. 1961.

(19) FORRESTER J.: Pricniples of systems.Cambridge MA. MIT Press 1966.

(20) FORRESTER J.: Urban Dynamics. MIT Press. 1969.

(21) FREY G.: Die Mathematisierung Unserer Welt. W. Kohlammer Verlag. Stuttgart. 1972.

(22) GELOVANY V.A.: A man-machine simulation system for global development processes. System Research II.116-129. London. 1985

(23) GLADKIJ A.V., MEL´CUK I.A.: Introducción a la lingüística matemática.Editorial Planeta.Barcelona. 1972.

(24) GOLOMB S.: A New Derivation of the Entropy Expressions. IRE, Transf. Inform.Theory. Vol II-7, n°3, 166-167. 1961.

(25) GROSS M., LENTIN A.:Notions sur les grammaires formelles. Gautier-Villars.París, 1977.

(26) HARRISON M.A.: Introduction to formal language theory. Addison Wesley, Reading, Mass. 1978.

(27) HOGEWEG P.: Celular automata as a paradigm for ecological Modelling. Applied Mathematics and Computation. 27. 81-100. 1988.

(28) HOPCROFT J., ULLMAN J.: Introduction to automata theory, languages and computation. Addison-Wesley, 1979.

(29) HUFFMAN D.A.: A method for the Construction of Minimum Redundancy Codes. Proc. IRE vol. 40. nº10. 1098-1101. 1952.

(30) INNIS G.S., JAMESON D.A.: Ecological Information for Range Management Decision-Prediction Models. Soc. Range Management 26 th Annu. Meeting. Boise. Idaho. 1973.

(31) IWASA Y., ANDREASEN V., LEVIN S.: Aggregation in model Ecosystems. A perfect aggregation. Ecological Modelling 37. 287-302. 1987.

(32) JORGENSEN S.E.: Fundamentals of Ecological Modelling. Elsevier.Amsterdam.1988.

(33) KARUSH J.: A simple proof of an inequality of McMillan. IRE Trans. Inform. Theory. Vol IT-17 nº 2.pág.118. (1961)

(34) KELLEY D.: Teoría de autómatas y lenguajes formales. Prentice Hall,1998

(35) KONNO N., GUNJI Y.: Mathematical Construction of an Autonomous Artificial Life. Appl. Math. Comput. 46. 33-58. 1991.

(36) KRAFT L.G.: A Device for Quantizing, Grouping and Coding Amplitude Modulated Pulses. M.S. thesis. Electrical Engineering Department. Massachusett Institute of Technology. March 1949.

(37) LIN Y., MA. Y.: Remarks on Analogy between Systems. Int. J.General Systems.13, 135-141. 1987.

(38) LIN Y., MA. Y.: Remarks on the definition of Systems. Syst. Anal. Model. 6,11/12, 923-931. 1989.

(39) MATHEWSON S.C.: The implementation of simulation languages. Computer modelling ans simulation. De. M. Pidd. Chichester. Wiley, 23-56. 1989.

(40) MCMILLAN B.: Two Inequalities Implied by Unique Deciberability. IRE Trans. Inform. Theory. Vol IT-2. 115-116. (1956)

(41) MESAROVIC M., TAKAHARA Y.: General System Theory. Mathematical Foundations. Academic Press. New York. 1978.

(42) MILLER J.G.: Living Systems. New York. Mc Graw-Hill. 1978.

(43) MORECROFT J.D.W.: A critical review of diagramming tools for conceptualizing feedback system models. Dynamics. 8 (1). 20-29. 1982.

(44) MORIN E.: La méthode. Seuil.Paris. 1977.

(45) NORMAN A.: Teoría de la Información y Codificación. Paraninfo. 1986.

(46) PATTEN B.C.: Holoecology. 1997.

(47) SAUSURE F.: Cours de linguistique général. Payot.Lausanna. 1972.

(48) SHANNON C.E., WEAVER W. : The mathematical theory of comunication. Uiv. of Illinois Press. 1949.

(49) SHIELDS M.W.: An introduction to automata theory.Blackwell Scientific Publ.,1987.

(50) SHOEMAKER CH. A.: Mathematical construction of ecological models. Ecosystem Modelling in theory and practice. An introduction with Case Histories. Edited by Charles A.S. Hall and John W. Doy. Jr. John Wiley. 1977.

(51) SINGH J.: Teoría de la Información, del lenguaje y de la cibernética. Alianza. Madrid, 1976.

(52) USÓ-DOMÈNECH J.L., MATEU J., LÓPEZ J.A.: Mathematical and Statistical Formulation of an ecological model with applications. Ecological modelling. 101. 27-40. 1997.

(53) USÓ-DOMÈNECH J.L., VILLACAMPA Y., MATEU J., SALVADOR P.: Towards a linguistic paradigm for mathematical modelling of the complex structural systems: A semantic theory of flow equations (Submitted to Ecological Modelling).1997.

(54) USÓ-DOMÈNECH J.L., VILLACAMPA Y., STÜBING G., KARJALAINEN T, RAMO M.P.: MARIOLA: A model for calculating the response of mediterranean bush ecosystem toclimate variations. Ecological Modelling. 80. 113-129. 1995

Volver al índice/Tornar a l'índex

(55) VILLACAMPA Y., CORTÉS M., VIVES F., USÓ J.L., CASTRO M.A.: A new computational algorihtm to construct mathematical models. Ecosud 99. Lemnos. Grecia. 1999.

(56) VILLACAMPA Y., USO J.L., MATEU J., VIVES F., SASTRE P.: Generative and recognoscitive grammars of ecological models. Ecological Modelling. (in Press). 1999.

(57) VILLACAMPA Y., USO J.L., VIVES F., LLORET M.: A populational model of the reproductive behaviour of mediterranean bushes: a case of Cistus Albidus L. Ecosud 97. Ecosystem Development Computational Mechanics Publications. 395-403. 1997.

(58) VILLACAMPA Y., USÓ J.L.: Mathematical models of complex structural systems. A linguistic vision. International Journals of General Systems (in press) 1999.

(59) VIVES F., VILLACAMPA Y., CORTÉS M., USÓ J.L.: An introduction to coding theory of flow equations in Ecological models. Ecosud 99. Lemnos. Grecia. 1999.

(60) YANG Z.B.: A New Model of General System Theory. Cybernetic and System. An international Journal 20, 67-76. 1989.

(61) ZEIGLER B.P.: Multifacetted modelling and discrete event simulation. Academic Press. London. 1984.

(62) ZHANG S.H., SCHROER B.J., MESSIMER S.L., TSENG F.T.: Software engineering and simulation. Third Int. Sof. for Strat. synth.Conf. Proc. University of Alabama. 33-42. 1990.