CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS

DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

" UN TRADUCTOR DEL CASTELLANO A LOGICA CLAUSULAR "

Tesis que presenta la Sra. Silvia del Carmen Guardati Buemo para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERIA ELECTRICA. Trabajo dirigido por el Prof. Guillermo Morales Luna, con la codirección del Prof. Sergio Chapa Vergara.

Becario de UNESCO

México D.F., Septiembre de 1987

AGRADECIMIENTOS

A UNESCO, por la ayuda económica que me brindó, sin la cual no hubiera podido realizar los estudios de Maestría.

Al Prof. Guillermo Morales Luna, un reconocimiento especial por la dirección que hizo de esta tesis.

A los Profesores Zdenek Zdrahal y Sergio Chapa Vergara por el interés que mostraron en la corrección de este trabajo.

A la Sección de Computación del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional.

A mis padres,

Nuemí Buemo de Guardati

y

Antonio Guardati

Y a mis hermanos:

Liliana, Antonio y Pablo

RESUMEN

El trabajo de tesis que aquí presentamos, es un traductor del lenguaje castellano a Prolog. El sistema acepta enunciados formulados en lenguaje natural y su salida son enunciados Prolog.
El traductor consiste de un editor de gramáticas, un intérprete que realiza el análisis de las oraciones con respecto a la gramática construída por el editor, un traductor de los enunciados (una vez analizados por el intérprete) a fórmulas de la lógica de primer orden, y finalmente un traductor a Prolog, que toma las fórmulas de la lógica clausular, detecta las cláusulas de Horn y las expresa de manera que el intérprete de Prolog las acepte.
El sistema fue implementado en lenguaje Lisp, versión Mulisp 83.
El diseño del traductor desarrollado en este trabajo fue presentado en la conferencia EXPERT SYSTEMS 87, organizada por el Grupo de Interés Especial en los Sistemas Expertos de la Sociedad Británica de Computación [7].

INDICE

1- INTRODUCCION

El traductor desarrollado convierte oraciones formuladas en castellano a fórmulas de la lógica clausular.
Este sistema está formado por un editor de gramáticas, un analizador sintáctico y un traductor a Prolog.
El editor de gramáticas permite a cada usuario especificar su propia gramática. La misma determinará qué oraciones serán reconocidas por el analizador sintáctico.
El analizador sintáctico reconoce enunciados en castellano y obtiene su correspondiente representación clausular.
El traductor a Prolog (aún no implementado) toma las cláusulas generadas por el analizador sintáctico, selecciona aquellas que son de Horn y las escribe como un programa en Prolog.
Una proposición atómica se representa como una lista de objetos, una cláusula como una pareja $(\alpha\ c)$, donde α y c están formadas por una lista de proposiciones atómicas, y un enunciado se representa como una lista de cláusulas.
El traductor cuenta con procedimientos que le permiten hacer un tratamiento adecuado de conectivos, variables y cuantificadores.

A continuación presentaremos cómo hemos estructurado este trabajo. En el capítulo 2 daremos algunos conceptos sobre lógica de primer orden y sobre gramáticas formales. En el capítulo 3, describiremos al sistema. Presentaremos algunos ejemplos para mostrar el funcionamiento del traductor y proporcionaremos un manual de uso. En el capítulo 4 expondremos cómo se representan las cláusulas del

cálculo de predicados y cómo se procesan conectivos y cuantificadores. En el capítulo 5 presentaremos al editor de gramáticas, y en el capítulo 6 al analizador sintáctico.Finalmente en el capítulo 7, daremos algunas ventajas y desventajas del sistema.

2- CALCULO DE PREDICADOS Y GRAMATICAS FORMALES

En este capítulo presentaremos conceptos básicos sobre el cálculo de predicados y sobre las gramáticas formales.
La exposición de estos temas es de gran utilidad, ya que los mismos constituyen la *base conceptual* del sistema.

2.1- Motivacion

La tesis que presentamos es un traductor de lenguaje natural a Prolog. Es decir, enunciados dados en castellano se convierten a un programa Prolog.
Prolog es un lenguaje de programación de alto nivel, basado en el cálculo de predicados.
Sintácticamente, Prolog es un lenguaje clausular de primer orden. Se le llama *clausular* porque toda fórmula o instrucción en Prolog debe ser una cláusula del tipo llamado " cláusulas de Horn ".
Semánticamente, Prolog computa un modelo canónico inducido por un conjunto de cláusulas de Horn.
Una característica destacable del lenguaje Prolog, es su " método de resolución " que se basa en dos procesos:

- silogismo disyuntivo
- unificación

El primero de estos procesos (silogismo disyuntivo), es un *principio de inferencia*. El segundo (unificación), es un método por medio del cual dos fórmulas se hacen *iguales*. Cuando ésto sucede decimos que *se unifican*.
Para ilustrar estos conceptos, mostraremos ejemplos:

1) Ejemplo de silogismo disyuntivo:

Sean las premisas:

$$P_1 \vee P_2 \qquad (1)$$

$$\neg P_1 \qquad (2)$$

De (1) y (2), el silogismo disyuntivo nos permite inferir: P_2

2) Ejemplo de unificación:

Sean las fórmulas:

$$F(X,a) \qquad (1)$$

$$F(p(b),Y) \qquad (2)$$

donde: X e Y son variables.

De (1) y (2), el proceso de unificación nos permite obtener la fórmula: $F(p(b),a)$.

A partir de dos fórmulas se obtuvo una nueva fórmula, por la substitución de X por $p(b)$ y de Y por a.

Por todo lo expresado sobre Prolog hacemos uso de él, usando su máquina de inferencia para efectuar inferencias y poder contestar preguntas.

Un futuro desarrollo de este trabajo, consistirá en implementar un mecanismo de inferencia propio.

2.2- Calculo de Predicados

A continuación presentaremos algunos conceptos básicos del cálculo de predicados. Estos conceptos son parte de la base teórica del sistema implementado.

Comenzaremos con una breve exposición sobre el cálculo proposicional.

El *cálculo proposicional* es un método de cálculo, que trata

proposiciones o enunciados.

Gramaticalmente, una proposición es una oración o conjunto de palabras con sentido completo.

En la lógica, sólo se consideran como proposiciones a aquellas oraciones del tipo declarativas, que son las que afirman o niegan algo.

Toda proposición tiene un valor de verdad asociado, que puede ser *falso* o *cierto*.

En cada proposición distinguimos tres elementos, que son:

- el sujeto: persona o cosa de quien se afirma o niega algo;
- el predicado: lo que se afirma o niega del sujeto;
- la cópula: el verbo que afirma o niega.

De acuerdo a su complejidad, las proposiciones pueden ser *simples* o *compuestas*.

Son simples cuando expresan una relación entre un solo sujeto y un solo predicado.

Son compuestas cuando están formadas por más de una proposición simple.

A continuación presentamos algunos ejemplos de proposiciones.

a) " los pájaros vuelan "

Es una proposición *simple*, con valor de verdad *cierto*.

b) " la luna tiene luz propia "

Es una proposición *simple*, con valor de verdad *falso*.

c) " juan es profesor y chofer "

Es una proposición *compuesta*, que equivale a decir:

" juan es profesor "

Y " juan es chofer "

Las proposiciones pueden combinarse, por medio de conectivos

lógicos, originando una nueva proposición.

En este caso, el valor de verdad de la proposición obtenida dependerá de los valores de verdad de las proposiciones que la componen.

Describiremos a continuación los conectivos lógicos manejados por el cálculo proposicional.

1) La negación ($\neg$). Dada una proposición p, podemos obtener su negación haciendo: $\neg p$. Lo cual se lee: " p no es cierta ".

El valor de verdad de la proposición negada será verdadero cuando el valor de verdad de la proposición original sea falso. Y será falso en caso contrario.

2) La conjunción ($\wedge$). Dada la proposición p y la proposición q, podemos obtener su conjunción haciendo: $p \wedge q$. Lo cual se lee: " p y q ". El valor de verdad de la conjunción será verdadero sólo si las dos proposiciones tienen un valor de verdad verdadero. Y será falso en caso contrario.

3) La disyunción ($\vee$). Aquí, sólo trataremos la disyunción inclusiva. Dada la proposición p y la proposición q, podemos obtener su disyunción haciendo: $p \vee q$. Lo cual se lee: " p o q ". El valor de verdad de la disyunción será verdadero si al menos una de las proposiciones tiene valor de verdad verdadero. Y será falso en caso contrario.

4) La implicación ($\Rightarrow$). Dada la proposición p y la proposición q expresamos $p \Rightarrow q$, que se lee: " si p entonces q "; p recibe el nombre de *antecedente* y q el de *consecuente*. El valor de verdad de la implicación será falso en el caso que el antecedente tenga un valor de verdad verdadero y el consecuente un valor de verdad falso, en los otros casos el valor de verdad será verdadero.

5) La equivalencia o bicondicional ($\equiv$). Dada la proposición p y la

proposición q expresamos $p \leftrightarrow q$, que se lee: " p si y sólo si q ". El valor de verdad de la equivalencia será verdadero cuando p y q tengan el mismo valor de verdad. Y será falso en caso contrario.

Ejemplos de proposiciones y conectivos:

Sea la proposición p que expresa el enunciado:

" las flores tienen perfume "

Sea la proposición q que expresa el enunciado:

" los pájaros viven en los bosques "

1) Si hacemos $\neg p$, el enunciado anterior se leerá como:

" NO ES CIERTO QUE las flores tienen perfume "

2) Si hacemos $p \wedge q$, la conjunción de las proposiciones se leerá como:

" las flores tienen perfume Y los pájaros viven en los bosques "

3) Si hacemos $p \rightarrow q$, la implicación de las proposiciones se leerá como:

" SI las flores tienen perfume ENTONCES los pájaros viven en el bosque "

En este caso la condicional no tiene un sentido real, ya que no es condición que las flores tengan perfume para que los pájaros vivan en los bosques.

A continuación presentaremos la definición de fórmula bien formada.

Formulas bien formadas (FBF)

Decimos que una fórmula del cálculo proposicional es una " fórmula

bien formada " si satisface alguna de las siguientes reglas:

1) Una *variable proposicional* aislada es una FBF.

2) Si A es una FBF, entonces $\neg A$ es una FBF.

3) Si A y B son FBF, entonces las siguientes fórmulas son FBF:

$$(A \wedge B),\ (A \vee B),\ (A \equiv B),\ (A \Rightarrow B)$$

4) Una cadena de símbolos es una FBF, si y sólo si el serlo se origina de la aplicación de las reglas 1, 2 y 3, un número finito de veces.

Presentamos a continuación, un ejemplo tomado de [3]:

Sea la cadena Ao:

$((p) \Rightarrow ((q) \Rightarrow (r))\ ) \Rightarrow (\ ((p) \Rightarrow (q)) \Rightarrow ((p) \Rightarrow (r))\ )$

Veremos que Ao es una FBF.

Se tiene:

$A_o = (A_1) \Rightarrow (B_1)$

donde: $A_1 = (p) \Rightarrow (\ (q) \Rightarrow (r)\ )$

$B_1 = (\ (p) \Rightarrow (q)\ ) \Rightarrow (\ (p) \Rightarrow (r)\ )$

Ahora: $A_1 = (A_2) \Rightarrow (B_2)$

donde: $A_2 = p$ que es una FBF

$B_2 = (q) \Rightarrow (r)$

También: $B_2 = (A_3) \Rightarrow (B_3)$

donde: $A_3 = q$ que es una FBF

$B_3 = r$ que es una FBF

Y: $B_1 = (A_4) \Rightarrow (B_4)$

donde: $A_4 = (p) \Rightarrow (q)$

$B_4 = (p) \Rightarrow (r)$

Con el mismo criterio utilizado para B_2, se puede demostrar que A_4 y B_4 son FBF.

Así aplicando sucesivamente las reglas, se demuestra que Ao es una

fórmula bien formada.

Formas normales

Cuando se trabaja con fórmulas complejas (donde aparecen muchas variables y conectivos) resulta complicado y difícil establecer relaciones lógicas entre ellas.

Por ello, es de gran utilidad obtener de cada fórmula una forma estándar de la misma.

Las formas estándares también son importantes en los casos en que dada una FBF *A* es conveniente seleccionar una FBF *B* equivalente a *A*,que sea mínima en cierto sentido(que sea lo más simple posible).

Presentaremos a continuación dos formas estándares, canónicas o normales: la *forma normal disyuntiva* y la *forma normal conjuntiva*.

Una fórmula está en *forma normal disyuntiva* (FND), si es de la forma:

$$B_1 \vee B_2 \vee \ldots \vee B_n$$

donde: 1) cada B_i es una conjunción elemental

2) no hay dos B_i que sean iguales

Una fórmula está en *forma normal conjuntiva* (FNC), si es de la forma:

$$B_1 \wedge B_2 \wedge \ldots \wedge B_n$$

donde: 1) cada B_i es una disyunción elemental

2) no hay dos B_i que sean iguales

Toda fórmula puede ser expresada en su forma normal.

Para obtener la transformación de cualquier fórmula a una forma normal, existen dos métodos.

Primero: mediante transformaciones basadas en equivalencias

lógicas. Por ejemplo, sea la cadena:

$\neg(p \wedge q) \Rightarrow \neg p \vee r$ (1)

$\neg\neg(p \wedge q) \vee \neg p \vee r$ (2) por: $(p \Rightarrow q) \equiv \neg p \vee q$

$(p \wedge q) \vee \neg p \vee r$ (3) por: $\neg\neg p \equiv p$

La fórmula (3) es una FND de (1).

Segundo: mediante el examen de la tabla de verdad. Este método es útil cuando se conoce sólo la tabla de verdad y no la fórmula.

A continuación presentaremos un sistema lógico que nos ofrece mayor flexibilidad en el tratamiento de relaciones entre individuos. Se trata del *cálculo de predicados de primer orden.*

Calculo de predicados de primer orden

Cuando presentamos el cálculo proposicional dijimos que trataba " proposiciones ", es decir oraciones declarativas que tienen un valor de verdad fijo (*verdadero* o *falso*).

El cálculo de predicados maneja *predicados*. Los predicados son sentencias que contienen una o más variables y por lo tanto su valor de verdad depende de los valores que substituyan a esas variables.

El cálculo de predicados es una extensión del cálculo proposicional, por lo tanto contiene todo lo del cálculo proposicional más otras facilidades.

Por ejemplo, el cálculo de predicados nos permite tratar con individuos, con relaciones entre individuos y con propiedades de conjuntos de individuos.

Con el cálculo de predicados, también podemos manejar *formas proposicionales*. Las formas proposicionales son estructuras que

aparecen como enunciados declarativos, pero no contienen un valor de verdad definido ya que manejan variables.

Los elementos utilizados en el cálculo de predicados son:

1) Constantes y variables: Representan a individuos o entidades.

2) Predicados: Expresan una propiedad de alguna variable o una relación entre una o más variables. Por ejemplo: " rojo(X) ", donde: X es una variable

rojo es un predicado

El enunciado " rojo(X) ", se lee como " X es rojo ".

3)Funciones: Representan transformaciones. Por ejemplo, con las funciones unarias " padre(X) " y " madre(X) " y el predicado binario " casado (X,Y) ", la expresión: casado(padre(X),madre(X)) indica que el padre de X y la madre de X están casados.

4) Cuantificadores: En la lógica de primer orden sólo se cuantifican las variables. Existen dos tipos de cuantificadores: el existencial (∃) y el universal (∀).

∃X se lee: " existe un X "

∀X se lee: " para todo X "

Ejemplo:

(∀X) (∃Y) (madre(Y,X))

Esta expresión se lee: " Para todo X existe una Y tal que Y es madre de X ", lo que es equivalente a decir: " todo individuo tiene una madre ".

Un término es: una constante, una variable o una función cuyos argumentos son términos.

Son *atómicas* las fórmulas del predicado de primer orden, cuando tienen la forma: $P(t_1,\ldots,t_n)$, donde P es un predicado y $t_1 \ldots t_n$ son términos.

Pero, una fórmula puede ser más compleja cuando se construye con combinaciones de fórmulas atómicas mediante conectivos lógicos y cuantificadores.

Ahora presentaremos una forma especial en la que puede estar expresada cualquier fórmula del cálculo de predicados. Esta es la forma llamada *forma normal PRENEX*.

Forma Normal PRENEX

Una fórmula del cálculo de predicados de primer orden está en la forma normal PRENEX, si es de la forma:

$$(Q_1 X_1)\ (Q_2 X_2)\ \ldots\ (Q_n X_n)\ M$$

donde:

1) Cada $(Q_i X_i)$ es $(\forall X_i)$ o $(\exists X_i)$
2) M es una FBF, que no contiene ningún cuantificador.
3) $X_i \neq X_j$ si $i \neq j$
4) Las variables $X_1, \ldots, X_n$ ocurren en M.
5) El alcance de $(Q_n X_n)$ es M.

Toda fórmula bien formada del cálculo de predicados tiene una fórmula *equivalente* en forma normal PRENEX.

Para lograr la transformación de cualquier FBF a una de la forma normal PRENEX, habrá que pasar *todos* los cuantificadores de la fórmula a la izquierda de ella. Para mayor información acerca de este método se remite al lector a las referencias [3] y [5].

Forma clausular

En la forma clausular los elementos básicos son las *proposiciones atómicas*, que representan relaciones entre individuos.

Existen estructuras más complejas, que expresan que una *conjunción de condiciones atómicas* implican una *conclusión atómica*. Estas

estructuras son llamadas *cláusulas*.

En general, una cláusula es una expresión de la forma:

$$B_1,\ldots,B_m \Leftarrow A_1,\ldots,A_n$$

donde:

1) $B_1,\ldots,B_m, A_1,\ldots,A_n$ son fórmulas atómicas.
2) n y m son ≥ 0
3) Las fórmulas atómicas $A_1,\ldots,A_n$ son las condiciones de la cláusula.
4) Las fórmulas atómicas $B_1,\ldots,B_m$ son las conclusiones alternativas de la cláusula.

Entonces, a una cláusula la definimos como " la conjunción de las condiciones implica la disyunción de las conclusiones ".

Para algunas aplicaciones de la lógica, es suficiente restringir la forma de las cláusulas a aquellas que contienen a lo sumo una conclusión. Este tipo de cláusulas son conocidas como *cláusulas de Horn*.

Clausulas de Horn

Una cláusula de Horn es una expresión de la forma:

$$B \Leftarrow A_1,\ldots,A_n$$

donde:

1) A_i y B son predicados con un cierto número de argumentos
2) Los k argumentos de cada predicado son términos.

Ejemplo de una cláusula de Horn:

abuelo(X,Y) ⇐ padre(X,Z) and padre(Z,Y)

donde:

abuelo(X,Y) es la *conclusión*

padre(X,Z) y padre(Z,Y) son las *condiciones*

Las cláusulas de Horn se aplican en el área de Bases de Datos, en programación lógica (por ejemplo, el lenguaje Prolog) y para expresar modelos de problemas de Inteligencia Artificial.

Para concluir esta sección observamos que la FNC de una proposición es, en esencia, una expresión equivalente consistente de una conjunción de cláusulas.

2.3- Gramaticas formales

En esta sección presentaremos las gramáticas y los lenguajes formales, además qué relaciones existen entre éstos y el cálculo de predicados.

Con el desarrollo de las computadoras, aumentaron los estudios sobre lenguajes formales. Estos estudios tenían y tienen dos orientaciones. La primera, se refiere a formalizar las propiedades de los lenguajes naturales adecuados para construir algoritmos que permitan la traducción automática. La segunda, se refiere a los lenguajes de programación. Todos los lenguajes de programación de alto nivel que se han desarrollado, son formales por naturaleza. Por lo tanto resulta necesario estudiar las características de los lenguajes formales, si se quieren alcanzar lenguajes cada vez más flexibles y poderosos, y traductores capaces de entenderlos y producir un lenguaje apropiado para la máquina.

La *Forma Normal de Backus* (FNB) constituye la más importante aportación con respecto a la descripción de lenguajes formales. La FNB permite describir concisa y precisamente un lenguaje formal.

La FNB hace uso de ciertos símbolos para la descripción de lenguajes:

-Los paréntesis angulares, "< >", se utilizan para representar los

símbolos no terminales.

-El símbolo "::=", se usa como símbolo definitorio. Es decir, indica que la expresión que se encuentra a la izquierda puede tomar alguno de los valores expresados a su derecha.

-La barra "|", se usa para separar definiciones alternativas.

A continuación presentaremos un ejemplo del uso de la FNB, para describir una FBF (consideremos solo cuatro variables).

‹VARIABLE› ::= p | q | r | s

‹FBF› ::= ‹variable› | ¬ (‹FBF›) | (‹FBF›) → (‹FBF›)

La primera línea de la definición sintáctica de las FBF define las variables.

La última línea expresa que una FBF es cualquier variable; o el símbolo de negación y un paréntesis abierto, seguidos por cualquier FBF y un paréntesis cerrado; o un paréntesis abierto seguido por cualquier FBF seguida por un paréntesis cerrado, un símbolo de implicación, un paréntesis abierto, luego cualquier FBF, y por último de un paréntesis cerrado.

Ahora presentaremos una definición formal de *gramática*.

Definicion formal de gramatica

Definimos una gramática G, como el cuádruple:

$$G = (N, T, P, \Sigma)$$

donde:

N : es el conjunto de símbolos no terminales.

T : es el conjunto de símbolos terminales.

P : es el conjunto de producciones o reglas gramaticales.

Σ : es el símbolo inicial ($\Sigma \in N$).

Las reglas gramaticales o producciones determinan cómo se pueden

generar o producir nuevas secuencias. Se definen como: $\alpha \rightarrow \beta$, que se lee " α produce a β " o " β es reducido a α ".

La jerarquía de gramáticas propuesta por Chomski, resulta de gran utilidad en los trabajos relacionados con el tratamiento de lenguajes, tanto artificiales (como los lenguajes de programación) como naturales.

Según esta jerarquía, cada lenguaje queda caracterizado por el tipo de producciones de la gramática que lo genera.

Así se definen cuatro tipos de gramáticas:

NOMBRE	TIPO	FORMA DE PRODUCCIONES
irrestrictas	0	$\phi\ A\ \psi \rightarrow \phi\ \omega\ \psi$
sensitivas	1	$\phi\ A\ \psi \rightarrow \phi\ \omega\ \psi$ $\omega \neq \lambda \quad \Sigma \rightarrow \lambda$
libres	2	$A \rightarrow \omega \quad \omega \neq \lambda$ $\Sigma \rightarrow \lambda$
regulares	3	$A \rightarrow \alpha B \quad \Sigma \rightarrow \lambda$ $A \rightarrow \alpha$ (a derecha) $A \rightarrow B\alpha \quad \Sigma \rightarrow \lambda$ $A \rightarrow \alpha$ (a izquierda)

donde:

- $A \in N \cup \{\Sigma\}$
- $\phi, \omega, \psi \in (N \cup T)^*$
- $B \in N$
- $\alpha \in T$
- λ es la cadena vacía

Gramatica tipo 0 (irresticta):

Este tipo de gramáticas permite que la cadena pueda ir contrayéndose. Una producción, en las grmáticas irrestrictas, tiene la forma: " $\phi\ A\ \psi \rightarrow \phi\ \omega\ \psi$ ", donde ω puede ser la cadena vacía (λ). Si esto sucede, la longitud del miembro de la izquierda

va a ser menor que la del miembro de la derecha.

Ejemplo:

Sea G_0: $\Sigma \rightarrow A$

$A \rightarrow \alpha A$

$A \rightarrow bC$

$bC \rightarrow b$

La regla $bC \rightarrow b$ *contrae* la cadena generada.

Gramatica tipo 1 (sensitiva al contexto):

Este tipo de gramáticas se caracteriza por contar con producciones donde su aplicación depende del contexto en el cual estén los símbolos terminales. Una producción, en las gramáticas sensitivas al contexto, tiene la forma: " $\phi A \psi \rightarrow \phi \omega \psi$ ", donde $\omega \neq \lambda$. Pero esta regla gramatical está expresando que solo podrá llevarse a cabo la substitución de A por ω, en el contexto de ϕ y ψ.

Ejemplo:

Sea G_1: $\Sigma \rightarrow A$

$A \rightarrow \alpha AB$

$A \rightarrow \alpha b$

$bB \rightarrow bb$

La regla $bB \rightarrow bb$ está determinando que la substitución de B se efectúe solo en el contexto de b.

Gramatica tipo 2 (libre de contexto):

En este tipo de gramáticas, la substitución de símbolos no terminales es independiente del contexto en el cual se encuentran. Es decir, la substitución depende de los símbolos y no del contexto.

Los lenguajes artificiales, especialmente los lenguajes de

programación, usan este tipo de gramáticas.

Ejemplo:

Sea G_2: $\Sigma \rightarrow A$

$A \rightarrow aAb$

$A \rightarrow ab$

La substitución del símbolo no terminal A no depende del contexto.

Gramatica tipo 3 (regulares):

Las gramáticas regulares pueden serlo por derecha o por izquierda. Lo son por derecha, cuando permiten que el lenguaje que generan se expanda por la derecha. Y lo son por la izquierda, cuando la expansión se da por la izquierda.

Ejemplos:

Sea G_1:		Sea G_2:	
Σ	$\rightarrow 1A$	Σ	$\rightarrow A1$
Σ	$\rightarrow 1$	Σ	$\rightarrow 1$
A	$\rightarrow 0B$	A	$\rightarrow B0$
B	$\rightarrow 1A$	B	$\rightarrow A1$
B	$\rightarrow 1$	B	$\rightarrow 1$
(regular por la derecha)		(regular por la izquierda)	

Una gramática genera un lenguaje con ciertas características. Esas características dependen de las producciones que tenga la gramática. A continuación ofrecemos al lector una definición formal de lenguaje.

Definicion formal de lenguaje

Un lenguaje L, generado por una gramática formal G, es el conjunto de cadenas de simbolos terminales derivables del símbolo inicial Σ.

$$L(G) = \{ W \in T^* / \Sigma \overset{*}{\rightarrow} W \}$$

donde:

T^* designa el conjunto de secuencias de símbolos de T

$\Sigma \overset{*}{\rightarrow} W$ es una forma abreviada de escribir todas las derivaciones: $\Sigma \rightarrow W_1 \rightarrow W_2 \rightarrow \ldots \rightarrow W$

La complejidad de un lenguaje, depende de la complejidad de la gramática que lo produzca. Como ya mencionamos cuando presentamos la definición formal de una gramática, las características de un lenguaje resultan directamente del tipo de producciones de la gramática en la cual tienen origen.

A continuación trataremos de establecer una relación entre el cálculo de predicados y las gramáticas formales.
Dijimos que una gramática permite generar un lenguaje, y que tiene ciertos elementos. Estos elementos guardan una correspondencia con los elementos de la lógica de primer orden.
1) Toda gramática tiene un conjunto finito no vacío de símbolos, llamado el *alfabeto*.
El alfabeto de la gramática se corresponde con el conjunto de variables y conectivos del cálculo proposicional.
2) Toda gramática tiene un conjunto de reglas o producciones aplicables al alfabeto, que determinan cómo combinar sucesiones de símbolos en oraciones.
Las reglas de la gramática se corresponden con las fórmulas bien formadas del cálculo de predicados.
Por todo ésto, podemos decir también que existe una relación directa entre el cálculo de predicados y oraciones formuladas en lenguaje natural. Sin embargo, la transformación o traducción del castellano a la forma clausular no siempre resulta sencilla de

realizar.

Muchas veces, es necesario efectuar ciertas modificaciones en la oración original (sin alterar su sentido) para lograr que pueda ser transformada a una forma clausular. De esta forma los elementos de la oración se expresan por medio de *cuantificadores* y *conectivos*.

Oraciones sencillas, como enunciados de predicados y relaciones, son fáciles de traducir.

Por ejemplo, la oración: " maría es buena " se traduce como $B(X)$ donde B es el nombre del predicado (en el ejemplo es_buena) y X es una variable del mismo (en el ejemplo la variable tiene un valor igual a **maría**).

Pero no todas las oraciones son tan sencillas. Existen otras más complejas que para representarlas en forma clausular requieren de expresiones cuantificadas y de conectivos.

Por ejemplo, la oración: " algunos hombres son violentos ", requiere de dos predicados básicos y de cuantificadores:

el predicado $H(X)$ expresa que " X es un hombre "

el predicado $V(X)$ expresa que " X es violento "

La fórmula final, en el cálculo de predicados es:

$$\exists X \,/\, H(X) \wedge V(X)$$

la cual se lee: " existe un X tal que X es un hombre y X es violento ".

3- DESCRIPCION DEL SISTEMA

Presentaremos nuestro sistema mediante su manual de uso y lo ilustraremos con un ejemplo que maneja estructuras de árbol.
El lector encontrará útil regresar a este capítulo cuando en los posteriores se presenten los algoritmos de representación utilizados. En una primera lectura recomendamos concentrarse en el procedimiento de uso del sistema.

3.1- Elementos del sistema

El sistema está formado por un Editor de Gramáticas, un Analizador Sintáctico y un Traductor a Prolog.
El sistema puede comenzar a operar a partir del editor de gramáticas o del analizador sintáctico. A partir del primero, cuando el usuario quiere definir su propia gramática para luego usarla en el proceso de análisis y traducción de oraciones en castellano. A partir del segundo, cuando el usuario hace uso de alguna gramática ya creada, y sólo utiliza el analizador.
El editor de gramáticas (se describirá en el capítulo 5) recibe ciertas especificaciones del usuario que le permiten generar una gramática. La salida del editor (la gramática) será usada por el analizador sintáctico para efectuar el reconocimiento de una oración en castellano.
El analizador sintáctico recibe como entrada la gramática y la oración que debe analizarse. También hace uso de un diccionario. La salida del analizador es la traducción de la oración a una forma clausular. Estas cláusulas serán la entrada del traductor a

Prolog.

El traductor a Prolog recibe las cláusulas, selecciona aquellas que son de Horn y las escribe como un programa en Prolog (este subsistema aún no ha sido implementado).

Al entrar al sistema, en la pantalla aparece un menú en el cual se presentan las diferentes opciones que se ofrecen al usuario.

En la siguiente sección se presenta un Manual de Uso, en el cual se explica cómo utilizar al sistema.

3.3- Manual de Uso

En esta sección explicaremos cómo hacer uso del sistema traductor que aquí presentamos.

Para mayor claridad, escribiremos los mensajes del sistema con letra estándar, mientras que a las órdenes que deberá dar el usuario las escribiremos en negritas.

Para poner en funcionamiento al traductor hay que proceder de la siguiente manera:

1- Instalar el sistema operativo MS-DOS

2- Instalar el intérprete Lisp. Hacer:

A> MULISP ‹RETURN›

3- Una vez instalado Lisp, quitar el disco de Lisp y colocar el disco en el cual se haya almacenado nuestro sistema. Luego hacer:

$ (LOAD 'TRESLOG.SYS) ‹RETURN›

En la pantalla aparecerá el siguiente menú:

O P T I O N S

F EDIT FUNCTION
V EDIT VARIABLE
P EDIT PROPERTY
E EVAL LISP
T TRACE FUNCTION
U UNTRACE FUNCTON
R READ FILE
W WRITE FILE
X EXIT TO DOS

ENTER CHOICE:

Seleccionar la opción E (EVAL LISP).

4- A continuación dar la orden:

$ (TRESLOG) ‹RETURN›

Al invocar la función TRESLOG, aparecerá en la pantalla el siguiente menú:

M E N U

1- REVISAR LOS TIPOS DE RELACIONES YA DEFINIDOS

2- CREAR UN NUEVO TIPO DE RELACIONES

3- ABANDONAR TRESLOG

INGRESE OPCION:

al desplegarse este menú, el usuario deberá ingresar la opción deseada, y oprimir la tecla ‹RETURN›.

En el caso que la opción ingresada sea distinta a las contempladas se llamará a la función CARTELES con un cinco como argumento (esta función se explicará en el capítulo 6, sección 6.1) y luego se volverá a desplegar este menú.

1- REVISAR LOS TIPOS DE RELACIONES YA DEFINIDOS: esta opción permite al usuario recorrer todos los tipos de relaciones ya definidos.

2- CREAR UN NUEVO TIPO DE RELACIONES: esta opción permite al

usuario crear su propio tipo de relaciones.

3- ABANDONAR TRESLOG: esta opción permite al usuario volver al modo evaluación de Lisp.

Si el usuario desea regresar al DOS, deberá proceder de la siguiente manera:

1- Estando en modo evaluación hacer:

```
$  RETURN NIL                ‹RETURN›
```

2- En la pantalla aparecerá el menú de Lisp ya presentado, seleccionar la opción X (EXIT TO DOS). Luego en la pantalla aparecerá :

```
A›
```

En seguida se explica cada una de las opciones:

α) Primera opción: cuando esta opción es seleccionada, el sistema presentará al usuario la información propia de cada tipo de relaciones que se recorra:

NOMBRE DEL TIPO DE RELACIONES : nombre

VECTOR Sgn : (n A_0 ... A_n)

INDICE i : i

LISTA DE NOMBRES DE RELACIONES INVOLUCRADAS EN nombre: (nom_1...nom_m)

UTILIZARA ESTE TIPO DE RELACIONES SI/NO ?

A continuación explicaremos cada uno de estos datos proporcionados por el sistema.

▸ nombre : es el nombre del tipo de relaciones que se está mostrando. Cada tipo se identifica por un nombre.

- (n A_0 ...A_n) : este vector da la siguiente información

  n : el número de argumentos de la relación

  A_0: definición de la relación

  A_1 .. A_n definición de los n argumentos

- i : este índice indica la posición en la cual aparece la definición del predicado

- (nom_1 ...nom_m) : cada nom_i da el nombre de una relación involucrada en "nombre".

El sistema pregunta al usuario si desea trabajar con el tipo de relaciones presentado.

Si el usuario ingresa SI, el sistema pasa el control al analizador sintáctico.

Si el usuario ingresa NO, se presentará en la pantalla la información referida al siguiente tipo de relaciones. En el caso que ya hayan sido recorridos todos los tipos o bien, que aún no se haya creado ningún tipo de relaciones, aparecerá en la pantalla el siguiente mensaje:

YA FUERON MOSTRADOS TODOS LOS TIPOS DE RELACIONES DEFINIDOS

1-DESEA VOLVER A REVISARLOS ?

2-DESEA CREAR UN NUEVO TIPO DE RELACIONES ?

3-DESEA ABANDONAR TRESLOG ?

INGRESE OPCION :

* Si el usuario ingresa :1, se volverán a recorrer todos los tipos de relaciones definidos.

* Si el usuario ingresa :2, se actuará como si en el menú

principal hubiera seleccionado la segunda opción (la misma se describirá en el punto *b*).

* Si el usuario ingresa :3, se actuará como se describió anteriormente para el caso en que el usuario elija la tercera opción en el menú principal.

b) Segunda opción: cuando se selecciona esta opción, el sistema pedirá al usuario la información necesaria para poder definir un nuevo tipo de relaciones.

Para explicar mejor al lector cómo proceder, nos apoyaremos en un ejemplo sencillo.

Queremos definir un tipo de relaciones que permita tratar un lenguaje para el manejo de estructuras de árbol (este ejemplo se presentará completo en la sección 3.3).

Llamaremos al tipo de relaciones: "BINARIAS1". En BINARIAS1 se tratará a la relación *altura*. Esta relación tendrá dos argumentos. El primero será el nombre de un nodo, y el segundo será un número que representa la altura de dicho nodo dentro del árbol.

Las oraciones que podrán reconocerse corresponden a la siguiente gramática:

‹BINARIAS1› ::= ‹ARG1› ‹RELACION› ‹ARG2›

‹ARG1› ::= ‹NOMBRE_NODO› / ‹FRASE1›

‹ARG2› ::= ‹NUMERO›

‹RELACION› ::= ‹VERBO4› ‹SUSTAN5›

‹FRASE1› ::= ‹FRASE2› ‹NOMBRE_NODO›

‹FRASE2› ::= ‹ARTIC› ‹SUSTAN1›

‹NOMBRE_NODO› ::= *a* / *e* / *i* / *o* / *u* / *aba* / ...

‹NUMERO› ::= uno / dos / tres / cuatro / ...

‹VERBO4› ::= tiene / tienen

‹ARTIC› ::= el / los / la / las / un / una ...

‹SUSTAN1› ::= nodo / nodos

‹SUSTAN5› ::= altura

El traductor, utilizando esta gramática, podrá procesar oraciones como:

" el nodo *a* tiene altura dos "

" *e* tiene altura tres "

" *e* y *u* tienen altura tres "

" *a*, *i* y *o* tienen altura dos o tres "

A continuación describiremos cómo ingresar la información necesaria para definir este tipo.

Al seleccionar la segunda opción del menú principal, aparecerán en la pantalla los siguientes mensajes (junto a cada mensaje se pone la información dada por el usuario) :

INGRESE EL NOMBRE DEL TIPO DE RELACIONES A DEFINIR: BINARIAS1

INGRESE LISTA DE RELACIONES INVOLUCRADAS EN : BINARIAS1

(altura)

DE UNA LISTA DE PALABRAS QUE INDIQUEN LA RELACION: altura

(altura)

CUANTOS ARGUMENTOS TENDRA LA RELACION ? : 2

INDIQUE UN INDICE i $\leq$ 2 TAL QUE: (arg_1 ... arg_i rel arg_{i+1} ...arg_n)

INGRESE INDICE: 1

DESEA DAR EL NOMBRE DE LA NUEVA GRAMATICA ? NO

Luego se limpiará la pantalla y aparecerá el siguiente mensaje:

INDIQUE COMO DEFINIRA: arg_i

‹arg_i› ::= (<NOMBRE_NODO>)

Nuevamente se limpia la pantalla y aparece:

ES TERMINAL: ‹NOMBRE_NODO› ? SI

CUAL DE ESTAS CLASES GRAMATICALES DA: arg_i ? -INDIQUELO ENTRE: ‹›-

(‹NOMBRE_NODO›)

<NOMBRE_NODO>

EXISTE OTRA OPCION PARA: arg_i ? SI

Vuelve a limpiarse la pantalla, y se pedirá nuevamente cómo se definirá a arg_i.

INDIQUE COMO DEFINIRA: arg_i

‹arg1› ::= (<FRASE1>)

ES TERMINAL: ‹FRASE1› ? NO

COMO DEFINIRA AL ELEMENTO: ‹FRASE1› ?

‹FRASE1› ::= (<FRASE2> <NOMBRE_NODO>)

ES TERMINAL: ‹FRASE2› ? NO

ES TERMINAL: ‹NOMBRE_NODO› ? SI

COMO DEFINIRA AL ELEMENTO: ‹FRASE2› ?

‹FRASE2› ::= (‹ARTIC› ‹SUSTAN1›)

ES TERMINAL: ‹ARTIC› ? SI

ES TERMINAL: ‹SUSTAN1› ? SI

CUAL DE ESTAS CLASES GRAMATICALES DA: arg_1 ? -INDIQUELO ENTRE:‹›-

(‹ARTIC› ‹SUSTAN1› ‹NOMBRE_NODO›)

‹NOMBRE_NODO›

EXISTE OTRA OPCION PARA: arg_1 ? **NO**

Así quedó definido el primer argumento de la relación. Este mismo proceso se repetirá para la relación y para el segundo argumento.

INDIQUE COMO DEFINIRA: rel

‹rel› ::= (‹VERBO4› ‹SUSTAN5›)

ES TERMINAL: ‹VERBO4› ? SI

ES TERMINAL: ‹SUSTAN5›? SI

CUAL DE ESTAS CLASES GRAMATICALES DA: rel ? -INDIQUELO ENTRE:‹›-

(‹VERBO4› ‹SUSTAN5›)

‹SUSTAN5›

EXISTE OTRA OPCION PARA: rel ? **NO**

Para definir al segundo argumento procedemos de la siguiente manera:

INDIQUE COMO DEFINIRA: arg_2

‹arg_2› ::= (‹NUMERO›)

ES TERMINAL: ‹NUMERO› ? SI

CUAL DE ESTAS CLASES GRAMATICALES DA: arg_2 ? -INDIQUELO ENTRE:‹›-

(‹NUMERO›)

‹NUMERO›

EXISTE OTRA OPCION PARA: arg_2 ? NO

Así quedó definida la gramática que usará el sistema para oraciones que den instancias de la relación *altura*.
Una vez concluída la definición del tipo BINARIAS1, se limpiará la pantalla y aparecerán los siguientes mensajes:

1- DESEA REALIZAR LA TRADUCCION DE ORACIONES ?

2- DESEA VOLVER AL MENU PRINCIPAL ?

INGRESE OPCION :

▸ Si el usuario ingresa: 1, se limpia la pantalla y aparece el mensaje:
INGRESE ORACION A TRADUCIR
y el sistema se queda esperando hasta que el usuario ingrese una oración y oprima la tecla ‹RETURN›
▸ Si el usuario ingresa: 2, aparecerá en la pantalla el menú principal del sistema.
A continuación presentaremos algunos ejemplos, donde mostraremos la representación utilizada para las cláusulas.

INGRESE ORACION A TRADUCIR

(el nodo a tiene altura dos) ‹RETURN›

Se limpiará la pantalla y aparecerá:

LISTA DE CLAUSULAS FORMADAS

(NIL ((altura a dos)))

VECTOR DE VARIABLES CUANTIFICADAS

NIL

PARA CONTINUAR OPRIMA LA TECLA S : S

La interpretación de las claúsulas formadas es naturalmente la declarada en castellano. Su definición formal aparece en el capítulo 4.
El vector de variables cuantificadas se construye según se verá en el capítulo 4 y capítulo 6, sección 6.3.

Los siguientes ejemplos los presentaremos dando la oración (precedida por un número) seguida del resultado obtenido por el traductor.
El * reemplaza a la coma que es una palabra reservada de Lisp.

1 (el nodo a y el nodo e tienen altura tres)

LISTA DE CLAUSULAS FORMADAS

(NIL ((altura a tres)))*

(NIL ((altura e tres)))

VECTOR DE VARIABLES CUANTIFICADAS

NIL

2 (*aba* * *aca* o *ici* tienen altura dos)

LISTA DE CLAUSULAS FORMADAS

(NIL ((altura *aba* dos) (altura *aca* dos) (altura *ici* dos)))

VECTOR DE VARIABLES CUANTIFICADAS

NIL

3 (TODO NODO tiene altura M)

LISTA DE CLAUSULAS FORMADAS

(NIL ((altura NODO M)))

VECTOR DE VARIABLES CUANTIFICADAS

((NODO (NODO)) (M (LIBRE)))

4 (PARA TODO NODO EXISTE UN M TAL QUE NODO tiene altura M)

LISTA DE CLAUSULAS FORMADAS

(NIL ((altura NODO M)))

VECTOR DE VARIABLES CUANTIFICADAS

((NODO (NODO)) (M (NODO)))

5 (α tiene altura dos O α tiene altura M)

LISTA DE CLAUSULAS FORMADAS

(NIL ((altura α dos) (altura α M)))

VECTOR DE VARIABLES CUANTIFICADAS

((M (LIBRE)))

σ (α no tiene altura dos)

LISTA DE CLAUSULAS FORMADAS

  (((altura α dos)) NIL)

VECTOR DE VARIABLES CUANTIFICADAS

NIL

ϗ (NO ES CIERTO QUE α tiene altura dos)

LISTA DE CLAUSULAS FORMADAS

  (((altura α dos)) NIL)

VECTOR DE VARIABLES CUANTIFICADAS

NIL

ϐ (α no tiene altura dos O e tiene altura tres)

LISTA DE CLAUSULAS FORMADAS

  (((altura α dos)) ((altura e tres)))

VECTOR DE VARIABLES CUANTIFICADAS

NIL

ϙ (SI α y e tienen altura dos ENTONCES i tiene altura tres)

LISTA DE CLAUSULAS FORMADAS

  (((altura α dos) (altura e dos)) ((altura i tres)))

VECTOR DE VARIABLES CUANTIFICADAS

NIL

10 (NO ES CIERTO QUE NODO1 o NODO2 no tienen altura tres)

LISTA DE CLAUSULAS FORMADAS

(NIL ((altura NODO1 tres) (altura NODO2 tres)))

VECTOR DE VARIABLES CUANTIFICADAS

((NODO1 (LIBRE)) (NODO2 (LIBRE)))

11 **(el α tiene altura dos)**

Esta oración no se corresponde con la gramática definida (comienza con un artículo y sigue con un nombre de nodo y no con un sustantivo como lo especifica la gramática).

El sistema borra la pantalla y aparece el siguiente mensaje:

SU ORACION NO CONCUERDA CON LA GRAMATICA

PARA CONTINUAR OPRIMA LA TECLA S: S

Para terminar con la descripción del manual de uso, presentaremos cómo incorporar nuevas palabras al diccionario.

Supongamos que queremos traducir la oración:

el nodo α tiene altura tres

y la palabra **altura** es desconocida para nuestro sistema, es decir no pertenece al dicionario. Entonces, en el momento de hacer el análisis y detectar que altura es desconocida, se limpiará la pantalla y aparecerá el siguiente mensaje:

altura

SU PALABRA NO PERTENECE AL DICCIONARIO: LA INGRESA -SI/NO- ? SI

CLASE DE LA PALABRA: ——————————→ SUSTANS ‹RETURN›

MODO O GENERO DE LA PALABRA: ——————————→ F ‹RETURN›

TIEMPO O NUMERO DE LA PALABRA: ——————————→ S ‹RETURN›

PERSONA DE LA PALABRA: ——————————→ 3 ‹RETURN›

en el caso que se hubiera ingresado NO, el sistema despliega en la pantalla el siguiente mensaje:

SE INTERRUMPE EL ANALISIS YA QUE UD. NO INGRESO LA PALABRA

PARA CONTINUAR OPRIMA LA TECLA S: S

Hasta aquí hemos presentado el manual de uso del sistema.
En la siguiente sección presentaremos una aplicación del traductor.

3.3- Una aplicacion del sistema

En esta sección presentaremos un ejemplo de una aplicación del traductor, motivo de este trabajo de tesis.
El ejemplo consiste de diferentes tipos de relaciones que permiten crear un lenguaje para manejar estructuras tipo árbol.
Se definieron cuatro tipos de relaciones: las UNARIAS, las BINARIAS, las BINARIAS1 y las TERCIARIAS.

Se trabajan con dos tipos de objetos: nombres de nodos (NOMBRE_NODO) y números (NUMERO).
Dentro del tipo UNARIAS se encuentran las relaciones: " raiz, nodo

y hoja ". Cada una de ellas indica qué parte del árbol constituye un nodo. Por ejemplo, por medio de la relación " raiz " se puede establecer que un cierto nodo es la raiz del árbol. El tipo de su argumento es: NOMBRE_NODO.

Dentro del tipo BINARIAS se encuentran las relaciones: " engendra, es_hijo_de, hermano, es_descendiente_de, es_ascendiente_de, y estar_a_la_misma_altura_de". Cada una de ellas indica una relación entre dos nodos. El tipo de sus argumentos es: NOMBRE_NODO y NOMBRE_NODO.

Dentro del tipo BINARIAS1 se encuentra la relación: "altura". Esta relación indica qué altura tiene cada nodo. El tipo de sus argumentos es: NOMBRE_NODO y NUMERO.

Dentro del tipo TERCIARIAS se encuentra la relación: "diferencia_de_altura". Esta relación indica qué diferencia de altura existe entre dos nodos. El tipo de sus argumentos es: NOMBRE_NODO, NOMBRE_NODO y NUMERO.

Para definir a cada uno de estos tipos procedimos como se describió en la sección 3.2, cuando describimos el tipo BINARIAS1. A continuación presentamos cómo quedaron definidos los vectores Sgn y las gramáticas correspondientes a cada tipo. Las gramáticas incluyen los elementos incorporados automáticamente por el sistema para el tratamiento de conectivos, variables y cuantificadores. También damos una lista con las palabras claves para cada relación.

NOMBRES DE TIPOS DE RELACIONES CREADAS

UNARIAS BINARIAS BINARIAS1 TERCIARIAS

NOMBRES DE GRAMATICAS CREADAS

GRAM1 GRAM2 GRAM3 GRAM4

A cada tipo de relaciones creado le corresponde una gramática, así GRAM1 es la gramática definida para UNARIAS, GRAM2 para BINARIAS, GRAM3 para BINARIAS1 y GRAM4 para TERCIARIAS.

TIPO " UNARIAS "

- **VECTOR Sgn DEL TIPO " UNARIAS " :**

(1 ‹RAIZ› ‹NOMBRE_NODO›)

- **INDICE i DEL TIPO " UNARIAS " : 1**

- **NOMBRES DE RELACIONES INVOLUCRADAS EN " UNARIAS " :**

(raiz nodo hoja)

- **GRAMATICA PARA EL TIPO " UNARIAS " (GRAM1) :**

‹UNARIAS› ::= ‹ARG1› ‹RELACION›
‹ARG1› ::= ‹NOMBRE_NODO› / ‹FRASE1›
‹RELACION› ::= ‹VERBO1› ‹FRASE2› [‹FRASE4›] /
‹VERBO1› ‹FRASE3› [‹FRASE4›] /
‹VERBO1› ‹FRASE5› [‹FRASE4›]
‹FRASE1› ::= ‹FRASE2› ‹NOMBRE_NODO›
‹FRASE2› ::= ‹ARTIC› ‹SUSTAN1›
‹FRASE3› ::= ‹ARTIC› ‹RAIZ›

‹FRASE4› ::= ‹ARTCON› ‹SUSTAN2›

‹FRASE5› ::= ‹ARTIC› ‹SUSTAN3›

‹NOMBRE_NODO› ::= *a* / *e* / *i* / *o* / *u* / *aba* / ...

‹VERBO1› ::= es / son

‹SUSTAN1› ::= nodo / nodos

‹SUSTAN2› ::= árbol

‹SUSTAN3› ::= hoja / hojas

‹ARTIC› ::= el / los / la / las / unos / ...

‹ARTCON› ::= del

‹RAIZ› ::= raíz

El traductor, con esta gramática, podrá reconocer oraciones como

" *a* es nodo del arbol "

" *e* es raiz del arbol "

" *e* y *o* son hojas "

" *u* no es hoja del arbol "

TIPO " BINARIAS "

▸ VECTOR Sgn DEL TIPO " BINARIAS " :

(2 ‹VERBO2› ‹NOMBRE_NODO› ‹NOMBRE_NODO›)

▸ INDICE i DL TIPO " BINARIAS " : 1

▸ NOMBRES DE RELACIONES INVOLUCRADAS EN " BINARIAS " :

(engendra hijo_de ascendiente_de descendiente_de hermano estar_a_la_misma_altura_de)

▸ GRAMATICA PARA EL TIPO "BINARIAS " (GRAM2) ":

‹BINARIAS› ::= ‹ARG1› ‹RELACION› ‹ARG2›

‹ARG1› ::= ‹NOMBRE_NODO› / ‹FRASE1›

‹ARG2› ::= ‹NOMBRE_NODO› / ‹FRASE1›

‹RELACION› ::= ‹VERBO1› ‹SUSTAN4› ‹PREP› /
‹VERBO2› ‹PREP› /
‹VERBO3› ‹FRASE6›

‹FRASE1› ::= ‹FRASE2› ‹NOMBRE_NODO›

‹FRASE2› ::= ‹ARTIC› ‹SUSTAN1›

‹FRASE6› ::= ‹PREP› ‹ARTIC› ‹PAL› ‹SUSTAN5› ‹PREP›

‹NOMBRE_NODO› ::= *a* / *e* / *i* / *o* / *u* / *aba* / ...

‹VERBO1› ::= es / son

‹VERBO2› ::= genera / engendra / desciende

‹VERBO3› ::= esta / estan

‹SUSTAN1› ::= nodo / nodos

‹SUSTAN4› ::= hijo / padre / hermano / descendiente / ascendiente

‹SUSTAN5› ::= altura / distancia

‹ARTIC› ::= el / los / la / las / unos

‹PREP› ::= de / a

‹PAL› ::= misma

El traductor, con esta gramática, podrá reconocer oraciones como:

" el nodo *e* desciende de *i* "

" *i* es padre de *e* "

" *u* es hermano de *i* y *o* "

" *a* esta a la misma altura de *aba* "

" *aca* no esta a la misma altura de *aba* "

" *u* y *o* son hijos del nodo *aca* "

" e o i son descendientes de u "

TIPO " BINARIAS1 "

▸ VECTOR Sgn DEL TIPO " BINARIAS1 " :

(2 ‹SUSTAN5› ‹NOMBRE_NODO› ‹NUMERO›)

▸ INDICE i DEL TIPO " BINARIAS1 " : 1

▸ NOMBRES DE RELACIONES INVOLUCRADAS EN " BINARIAS1 " :

(altura)

▸ GRAMATRICA PARA EL TIPO " BINARIAS1 " (GRAM3):

Fue descrita en este capítulo, en la sección 3.2.

TIPO " TERCIARIAS "

▸ VECTOR Sgn DEL TIPO " TERCIARIAS " :

(3 ‹SUSTAN5› ‹NOMBRE_NODO› ‹NOMBRE_NODO› ‹NUMERO›)

▸ INDICE i DEL TIPO " TERCIARIAS " : 2

▸ NOMBRES DE RELACIONES INVOLUCRADAS EN " TERCIARIAS " :

(diferencia_de_altura)

▸ GRAMATICA PARA EL TIPO " TERCIARIAS " (GRAM4) :

‹TERCIARIAS› ::= ‹ARG1› ‹ARG2› ‹RELACION› ‹ARG3›

‹ARG1› ::= ‹NOMBRE_NODO› ‹CONEC› / ‹FRASE1› ‹CONEC›

‹ARG2› ::= ‹NOMBRE_NODO› / ‹FRASE1›

‹ARG3› ::= ‹NUMERO›

‹RELACION› ::= ‹VERBO3› ‹PREP› ‹ARTIC› ‹FRASE7› /

‹VERBO4› ‹ARTIC› ‹SUSTAN› ‹PREP› ‹FRASE7›

‹FRASE1› ::= ‹FRASE2› ‹NOMBRE_NODO›

‹FRASE2› ::= ‹ARTIC› ‹SUSTAN1›

‹FRASE7› ::= ‹SUSTAN5› ‹PREP›

‹NOMBRE_NODO› ::= *a* / *e* / *i* / *o* / *u* / *aba* / ...

‹CONEC› ::= y

‹NUMERO› ::= uno / dos / tres / cuatro / ...

‹VERBO3› ::= esta / estan

‹VERBO4› ::= tiene / tienen

‹ARTIC› ::= el / los / la / las / unos / ...

‹SUSTAN› ::= diferencia

‹SUSTAN1› ::= nodo / nodos

‹SUSTAN5› ::= altura / distancia

‹PREP› ::= a / de

El traductor, con esta gramática, podrá reconocer oraciones como:

" *a* y *e* tienen una diferencia de altura de tres "

" el nodo *a* y el nodo *o* estan a una distancia de dos "

" *a* y *u* tienen una diferencia de altura de dos o tres "

" *aba* y *u* no estan a una distancia de tres "

Cuando se definieron estas gramáticas (GRAM1, GRAM2, GRAM3 y GRAM4), el sistema automáticamente incorporó las especificaciones

necesarias para permitir el tratamiento de: *conectivos implícitos*, *conectivos explícitos*, *variables* y *cuantificadores*.

Las estructuras utilizadas para definir la gramática, tales como ‹FRASE1›, ‹FRASE2›,..., ‹FRASE7›, fueron especificadas una sola vez. En esta presentación se ha repetido su definición para facilitar la tarea del lector. Además todos los símbolos terminales son comunes a todas las gramáticas.

PALABRAS CLAVES QUE IDENTIFICAN A RELACIONES

(PALABRA CLAVE - TIPO - RELACION)

(	raiz	UNARIAS	raiz)
(	nodo	UNARIAS	nodo)
(	nodos	UNARIAS	nodo)
(	hoja	UNARIAS	hoja)
(	hojas	UNARIAS	hoja)
(	engendra	BINARIAS	engendra)
(	padre	BINARIAS	engendra)
(	hijo	BINARIAS	hijo_de)
(	hijos	BINARIAS	hijo_de)
(	hermano	BINARIAS	hermano)
(	hermanos	BINARIAS	hermano)
(	ascendiente	BINARIAS	ascendiente_de)

```
(    descendiente    BINARIAS          descendiente_de     )
(    desciende       BINARIAS          descendiente_de     )
(    altura          BINARIAS    estar_a_la_misma_altura_de )
(    altura          BINARIAS1         altura              )
(    altura          TERCIARIAS        diferencia_de_altura)
(    distancia       TERCIARIAS        diferencia_de_altura)
```

En el siguiente capítulo presentaremos la representación clausular que hemos utilizado en nuestro sistema.

4- REPRESENTACION DE CLAUSULAS

En este capítulo presentaremos cómo se *representan* las oraciones formuladas en castellano en forma clausular.
Mostraremos la manera en la que se representan las proposiciones atómicas y las cláusulas. También cómo se tratan los conectivos, las variables y los cuantificadores.

Una proposición atómica se representa como una lista de objetos. Los elementos de la lista (los *objetos*) son el nombre y los argumentos de la relación.
Entonces una proposición atómica, que es una *instancia* de una relación, queda representada por una lista con la forma:

(R a_1 a_2 ... a_n)

donde: R es el nombre de la relación

a_i, i=1,...,n son los argumentos

El sistema también puede representar cláusulas. Una cláusula es un par ordenado (α , c), donde α recibe el nombre de *antecedente* y c el de *consecuente* de la cláusula.
Una cláusula se interpreta como: " la conjunción de los elementos de α implica la disyunción de los elementos de c ".
Para representar una cláusula también hacemos uso de las listas. Una cláusula queda representada por una pareja de la forma:

(Prop.atómica* ; Prop.atómica*)

donde el antecedente y el consecuente son listas de proposiciones atómicas.
Las listas del antecedente constituyen la *conjunción de las*

condiciones, y las listas del consecuente constituyen la *disyunción de las conclusiones*.

Por último, el sistema puede representar en forma de cláusulas cualquier oración del castellano, formulada según la gramática definida. La representación de los enunciados se efectúa con una lista de cláusulas:

(cláusulas)

El sistema durante el análisis sintáctico de la oración, reconoce palabras claves. Cuando termina de analizar la oración, si la misma corresponde con la gramática, se construye su representación en forma clausular.

A continuación presentaremos cómo se tratan los conectivos lógicos.

4.1- Manejo de conectivos

En la expresión de oraciones se permiten usar conectivos. Los *conectivos lógicos* permiten crear proposiciones complejas.

Los conectivos que maneja el sistema son: *conjunción*, *disyunción*, *negación* e *implicación*.

Los conectivos tienen prioridades.que se utilizan para determinar cual se procesará primero. En el caso que dos conectivos tengan la misma prioridad, se ejecutará primero el que se encuentre más a la izquierda.

En este trabajo, las prioridades de los conectivos se corresponden (de mayor a menor) con el orden en el cual fueron citados.

Los conectivos pueden presentarse de dos maneras: en su modo *explícito* o en su modo *implícito*.

Los conectivos *explícitos* se encuentran entre proposiciones y los

distinguimos por escribirse con letras mayúsculas. Por ejemplo: Y , E (conjunción); O, U (disyunción); NO ES CIERTO QUE (negación); SI-ENTONCES (implicación).

Los conectivos *implícitos* se encuentran en las proposiciones, afectando los objetos y la relación, y aparecerán en minúsculas. Por ejemplo: y, e (conjunción); o, u (disyunción); no (negación).

Cuando existen conectivos implícitos se originan distintas instancias de una relación. Por ejemplo, la oración:

juan estudia física y matemática

es equivalente a decir:

juan estudia física y

juan estudia matemática

donde cada una de las oraciones representa una instancia de la misma relación.

La implicación sólo puede aparecer en modo *explícito*, mientras que los otros conectivos lógicos (conjunción, disyunción y negación) pueden ocurrir en modo *explícito* y/o *implícito*.

La conjunción y la disyunción implicita afectan tanto a los objetos como a la relación. La negación implicita sólo puede aparecer afectando a la relación.

Se hace uso del algoritmo de Wang. Este algoritmo está ligado a la representación de listas utilizada, ya que toda fórmula podrá ser expresada como una cláusula. El antecedente de la cláusula será una conjunción de proposiciones atómicas (condiciones), mientras que el consecuente será una disyunción de prosiciones atómicas (conclusiones).

A continuación presentaremos el algoritmo:

Primero: Escribir las premisas a la izquierda de la flecha y las conclusiones a la derecha de la flecha.

$$\text{premisa}_1, \ldots, \text{premisa}_n \longrightarrow \text{conclusión}_1, \ldots, \text{conclusión}_m$$

Segundo: Si el principal conectivo de una fórmula bien formada (FBF) es una negación, entonces eliminar el signo de la negación y pasar la FBF al otro lado de la flecha.

Por ejemplo, dada la fórmula:

$$p \vee q , \neg(t \wedge s) , p \vee r \longrightarrow s , \neg p$$

su transformación aplicando la segunda regla nos queda:

$$p \vee q , p \vee r , p \longrightarrow s , (t \wedge s)$$

Tercero: Si el principal conectivo de una FBF a la izquierda de la flecha es una conjunción ($\wedge$), o a la derecha de la flecha es una disyunción ($\vee$), cambiarlos por comas.

Por ejemplo, dada la fórmula:

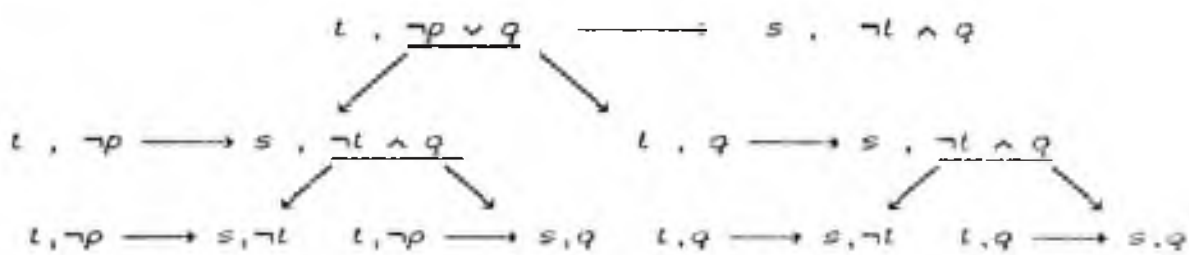
$$p \wedge q , r \longrightarrow s \vee \neg r$$

su transformación aplicando la regla tercera nos queda:

$$p , q , r \longrightarrow s , \neg r$$

Cuarto: Si el principal conectivo de una FBF a la izquierda de la flecha es una disyunción ($\vee$), o a la derecha de la flecha es una conjunción ($\wedge$), dividir la fórmula en dos sub-fórmulas bien formadas.

Por ejemplo, dada la fórmula:



Quinto: Si la misma FBF ocurre en ambos lados de la flecha, entonces esa línea se elimina pues es una tautología.

Por ejemplo, dada la fórmula:

$$t \ , \ q \longrightarrow s \ , \ q$$

Por la quinta regla puede eliminarse, ya que es una tautología.

Así, dada una proposición, con la representación que utilizamos y con el algoritmo de Wang, la expresamos mediante una lista mínima de cláusulas.

4.2- Uso de variables

El sistema es capaz de manejar variables para cada tipo de objetos de una relación. El tratamiento se hace durante el análisis de la oración y el proceso de representación. Este es el mismo que para las constantes.

En nuestro sistema las variables se caracterizan por comenzar con letra mayúscula.

4.3- Manejo de cuantificadores

Los cuantificadores que usamos en el sistema, son de dos categorías: *universales* y *existenciales*.

Algunos de los cuantificadores universales utilizados son: TODO , LOS, LAS, CUALQUIER, NINGUN, NINGUNA.

Algunos de los cuantificadores existenciales utilizados son: UN, UNO, UNA, ALGUN, ALGUNA, VARIOS, POCOS.

Los cuantificadores aparecen escritos con letras mayúsculas.

También se manejan las estrucutras: PARA TODO, PARA CUALQUIER, ..., como cuantificadores universales; y las estrucuras: EXISTE UN / TAL QUE , EXISTE ALGUN / TAL QUE, ..., como cuantificadores

existenciales.

Escribimos a cada fórmula en su forma PRENEX.
A cada variable que aparece cuantificada existencialmente, se la expresa como una función SKOLEM de las variables precedentes cuantificadas universalmente. Luego de este proceso, se considera que todas las variables de la fórmula se encuentran cuantificadas universalmente.
Una vez que la fórmula queda libre de cuantificadores, se la expresa en su forma normal conjuntiva y se obtiene su representación clausular, según los procedimientos ya descritos.

Un cuantificador tiene vigencia sobre una variable hasta que termine el proceso de transformación de la proposición, o bien hasta que aparezca la misma variable afectada por otro cuantificador. En este último caso, es necesario definir un nuevo nombre para la variable para poder distinguirla de la o las precedente(s).

Para el tratamiento de los cuantificadores, se construye un vector (durante el análisis de la oración) de variables cuantificadas:Var_Dep, donde cada elemento del vector será una lista.
Esa lista tendrá la forma (X (Y)), donde el primer elemento "X" es el nombre de una variable y el segundo elemento "(Y)" es una lista formada por:

$$Y = \begin{cases} X & \text{si } X \text{ está cuantificada universalmente} \\ \lambda & \text{si } X \text{ es una variable libre} \\ \text{una función Skolem de las variables precedentes cuantificadas universalmente} & \text{si } X \text{ está cuantificada existencialmente} \end{cases}$$

Toda proposición será representada por un *vector de variables cuantificadas* (Var_Dep) y por la *representación clausular* de la misma.

Sean dos proposiciones: Y y Z , con sus respectivas representaciones y vectores Var_Dep.

Según el conectivo que aparezca entre las proposiciones, se operará sobre las representaciones y sobre los vectores de variables cuantificadas.

En el capítulo 6 se expondrá cómo se tratan las representaciones cuando existen conectivos explícitos.

Con respecto a los vectores de variables cuantificadas, si el conectivo es:

1- *conjunción* o *disyunción*, los vectores de las proposiciones afectadas se concatenan, tras haber sido renombradas si hubiese sido necesario.

2- *negación*, el vector se obtiene de la siguiente manera:

$$\text{Var\_dep}_{\neg Y}(\text{var}) = \begin{cases} \text{var} & \text{si Var\_Dep(var)} \neq \text{var} \\ \text{las variables precedentes a var, cuantificadas existencialmente} & \text{en otro caso} \end{cases}$$

lo que se deriva de las Leyes de De Morgan para cuantificadores.

9- *implicación*, se la tratará como $\neg Y \vee Z$.

De esta menera, cualquier fórmula se representa por una lista de cláusulas y por un vector de variables cuantificadas (Var_Dep).

5- EDITOR DE GRAMATICAS

En este capítulo, mencionaremos algunos conceptos sobre la gramática castellana.
También presentaremos la construcción del *editor de gramáticas*.

5.1- Abstraccion de la gramatica castellana

Cuando en la sección 2.2 presentamos las gramáticas formales, vimos los distintos tipos.
Una simplificación, relativamente amplia de la gramática castellana puede considerarse del tipo: *libre de contexto*. Sin embargo, por la amplitud y la flexibilidad que caracterizan a una gramática capaz de generar lenguajes naturales, no podemos afirmar que la gramática castellana se ajusta totalmente a una gramática libre de contexto.

Definimos la gramática castellana como un sistema que permite describir y explicar todas las oraciones posibles del castellano sin necesidad de contenerlas a todas dentro de un inventario [8].
La gramática especifica ciertas estructuras por medio de las cuales un conjunto de palabras adquiere sentido. Por ejemplo, consideremos el siguiente grupo de palabras:

ante hombres ley iguales los todos son la.

Es obvio, que esas palabras, en ese orden, no nos dicen nada. No están expresando una idea, y se debe a que no respetan ninguna estructura dada por la gramática castellana.
Ahora, si reordenamos esas palabras, nos queda la siguiente oración: *todos los hombres son iguales ante la ley*, la cual tiene

sentido para un hispanohablante.

Por lo tanto, podemos afirmar que el castellano, al igual que otros idiomas, no es simplemente un conjunto de palabras. Existen *estructuras* que determinan el modo de conexión entre las palabras.

Podemos decir que una oración es una estructura, compuesta por estructuras. Por ejemplo, dada la oración: *El obrero trabaja mucho*, la podemos definir como una estructura. La que a su vez puede definirse como la suma de otras dos estructuras: *sujeto* y *predicado*.

Las estructuras pueden utilizárse de manera recursiva en la definición de una gramática. La *recursividad* nos permite definir, por medio de una representación finita, un conjunto infinito de oraciones.

A continuación presentamos una gramática escrita en Forma Normal de Backus, que pueda generar la oración: *El obrero trabaja mucho*.

(1)	‹ORACION›	::=	‹FN› ‹FV›
(2)	‹FN›	::=	‹ARTICULO› ‹SUSTANTIVO›
(3)	‹FV›	::=	‹VERBO› ‹ADVERBIO›
(4)	‹ARTICULO›	::=	*El*
(5)	‹SUSTANTIVO›	::=	*obrero*
(6)	‹VERBO›	::=	*trabaja*
(7)	‹ADVERBIO›	::=	*mucho*

Fig.5.1

Generacion de oraciones

El proceso de *generar oraciones* consiste en aplicar sucesivamente

reglas de producción, hasta que no puedan aplicarse nuevas reglas, o bien, hasta que la serie quede constituída solamente por símbolos terminales.

Cada regla de producción especifica cómo una frase puede ser reescrita como la concatenación de otras frases.

Su primer miembro puede aparearse con una sub-serie (o toda) de la serie que ha de transformarse, y su segundo miembro indica el reemplazo a efectuarse para una porción apareada.

Con un ejemplo, sobre la oración mencionada anteriormente, ilustraremos estos conceptos.

PASO	ESTRUCTURA	REGLA APLICADA
a	‹ORACION›	———
b	‹FN› ‹FV›	(1)
c	‹ARTICULO› ‹SUSTANTIVO› ‹FV›	(2)
d	‹ARTICULO› ‹SUSTANTIVO› ‹VERBO› ‹ADVERBIO›	(3)
e	*El* ‹SUSTANTIVO› ‹VERBO› ‹ADVERBIO›	(4)
f	*El obrero* ‹VERBO› ‹ADVERBIO›	(5)
g	*El obrero trabaja* ‹ADVERBIO›	(6)
h	*El obrero trabaja mucho.*	(7)

Nota: Los números de reglas son los que aparecen a la izquierda en la definición de la gramática (Fig.5.1).

El sistema de reglas de reescritura constituye un algoritmo para producir oraciones.

Para representar estructuras, existen modos que son estándares y fáciles de interpretar.

El modo más común es el *árbol*.

Damos a continuación la representación en forma de árbol, de la gramática presentada en la Fig. 5.1.

Las ATN (redes de transición aumentadas), permiten realizar el análisis sintáctico y semántico de oraciones formuladas en lenguaje natural. Presentaremos a continuación, algunos conceptos sobre las ATN.

Redes de transicion aumentadas (ATNs, por sus siglas del inglés Augmented Transition Network)

Un analizador sintáctico puede verse como un autómata que transita de un estado a otro recorriendo una cadena. Cuando se arriba a un estado final el análisis resulta exitoso.

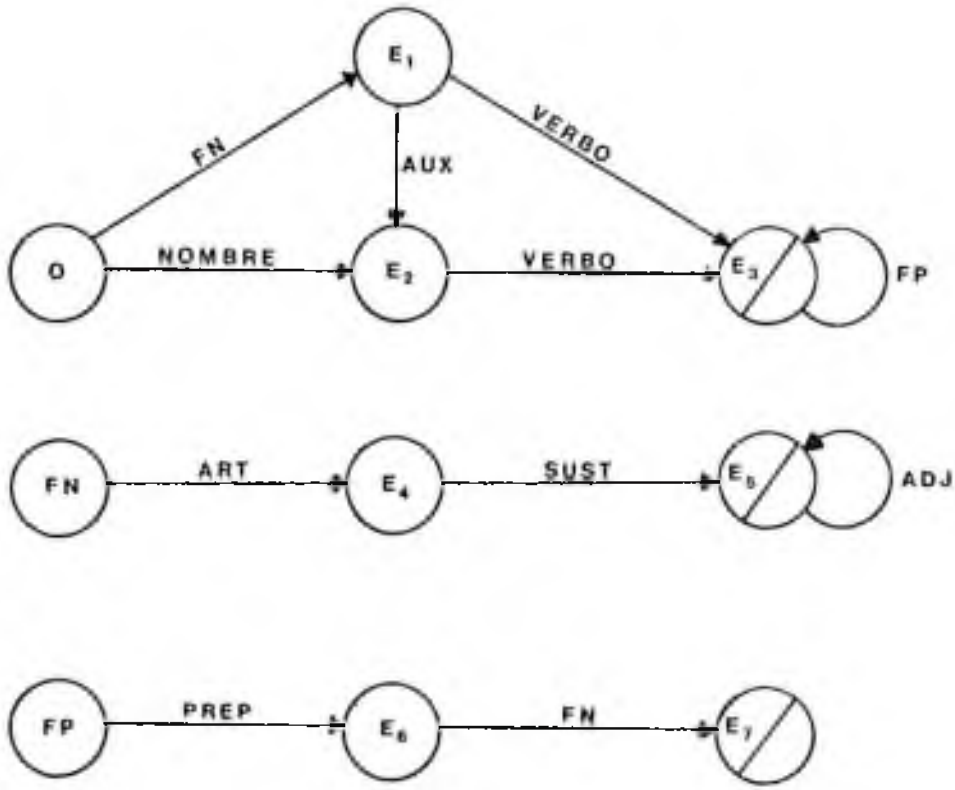
Podemos definir una ATN como una gramática de estados finitos, la que puede ser empleada tanto para el análisis como para la generación de enunciados. Una ATN puede verse como una gráfica dirigida con estados y arcos.

Cada arco está etiquetado con un símbolo que corresponde a un componente establecido en la gramática. Además una ATN permite el manejo de sub-gráficas independientes que pueden llamarse recursivamente.

Se dice que una oración satisface o cumple con la gramática si se

comienza con el estado *inicial* (de la gráfica) y se finaliza con un estado *final* (de la gráfica).

Presentamos la gráfica correspondiente a una ATN, para una gramática sencilla.



Veremos como se efectúa el análisis (según la gráfica) de la siguiente oración: *El hombre asustó a la niña.*

El análisis comienza en el estado inicial 0, la transición se da hacia la sub-red FN, pasando al estado E1.

FN, se satisface totalmente ya que *El* se corresponde con ART, y *hombre* con SUST, llegando al estado final, de la sub-red, E5.

De E1 el análisis sigue con la palabra *asustó*, que es un VERBO, pasando al estado E3. En el estado E3, que es un estado final, si la oración ya se recorrió totalmente se dá por finalizado el análisis. Pero, en nuestro caso aún queda parte de la oración sin ser recorrida: *a la niña*. Entonces, de E3 se pasa a la sub-red FP.

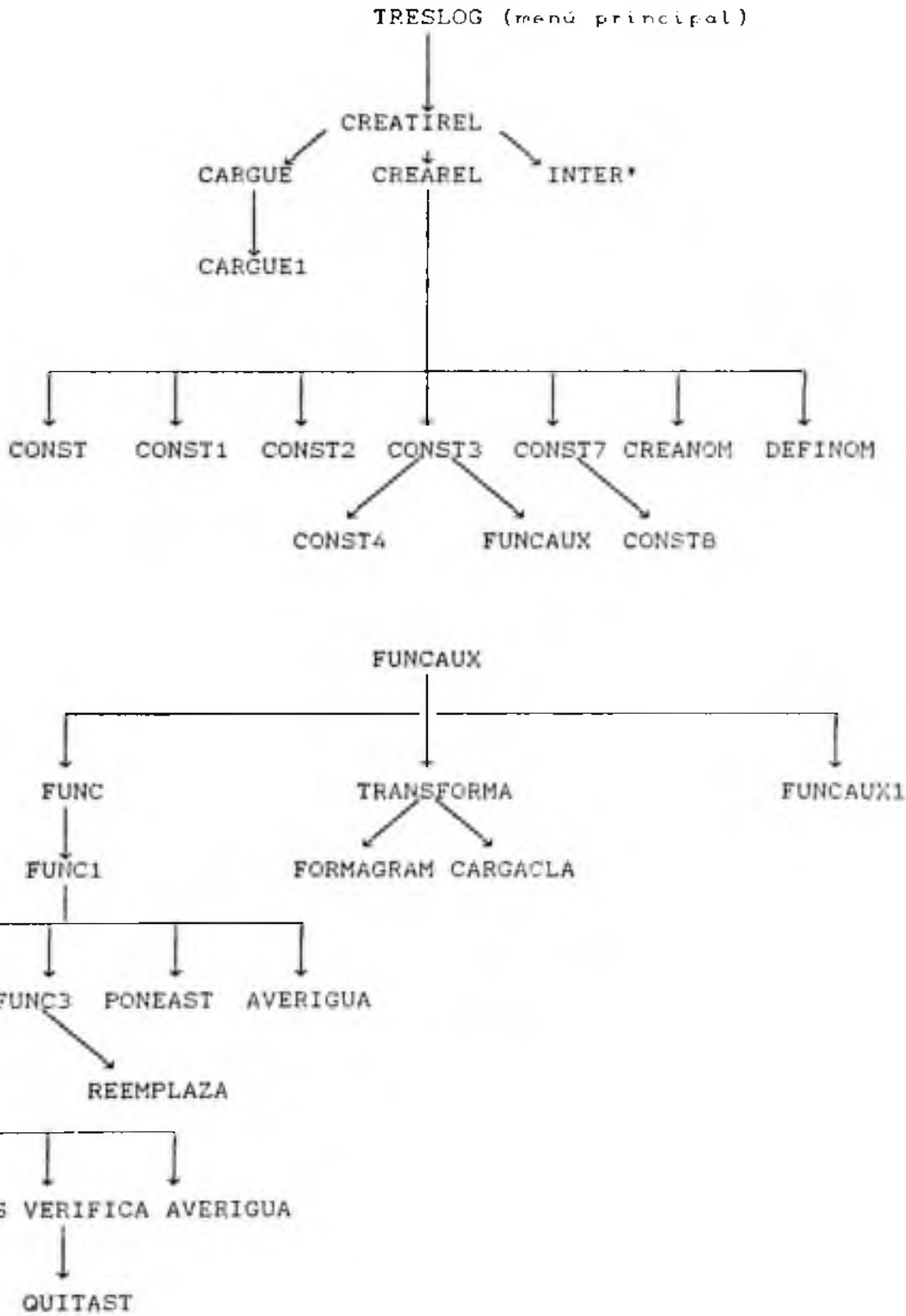
En FP, con *a*, que es una preposición, se pasa al estado Es y de este estado se transfiere el control a la sub-red FN, la cual se satisface con *la niña* (ART seguido de SUST).
Por último, el control queda nuevamente en Es, y como la oración ya fue recorrida totalmente se considera que el análisis finalizó con éxito.
Con este ejemplo tratamos de ilustrar cómo una oración formulada en Lenguaje Natural puede ser reconocida por una ATN.
La ATN puede definirse tan amplia como se desee, para reconocer y analizar oraciones complejas.
Una ATN, es una forma simple y eficiente de realizar el análisis de enunciados en lenguaje natural.
El concepto de ATNs es empleado parcialmente en la implementación del sistema.

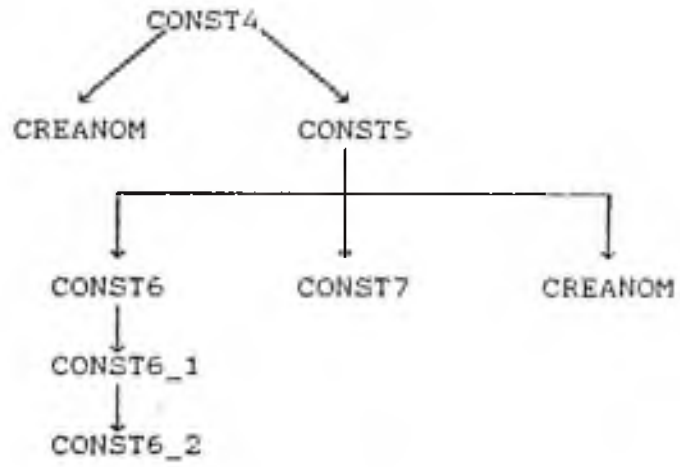
A continuación presentaremos la construcción del *editor de gramáticas*.

5.2- Construccion del Editor de Gramaticas

En este sección presentaremos cómo fue construído el Editor de Gramáticas.
Este subsistema permite al usuario definir su propia gramática, utilizando la Forma Normal de Backus.
La gramática especificada por el usuario, determinará cuales oraciones del castellano serán aceptadas por el analizador.

Antes de describir al editor de gramáticas detalladamente, daremos un esquema general de su composición, presentando la interacción existente entre las funciones que lo constituyen.


TRESLOG (menú principal)
CREATIREL
CARGUE
CREAREL
INTER*
CARGUE1
CONST
CONST1
CONST2
CONST3
CONST7
CREANOM
DEFINOM
CONST4
FUNCAUX
CONST8
FUNCAUX
FUNC
TRANSFORMA
FUNCAUX1
FUNC1
FORMAGRAM
CARGACLA
FUNC2
FUNC3
PONEAST
AVERIGUA
REEMPLAZA
ASTERIS
VERIFICA
AVERIGUA
QUITAST



* Esta función no se describe en este capítulo porque no es parte del Editor de Gramáticas.

Fig. 5.2

La Fig.5.2 podrá servir de guía al lector durante la siguiente presentación.

Como ya mencionamos en el capítulo 3, el sistema despliega un menú en la pantalla. Una de las opciones del menú es definir un nuevo tipo de relaciones.

En el caso que el usuario elija esta opción, el control del sistema se pasa a la función CREATIREL.

Esta función, junto a otras que se van convocando, crea una nueva gramática de acuerdo a los datos ingresados por el usuario.

El Editor de Gramáticas hace uso de las estructuras de datos de Lisp, conocidas con el nombre de *variables globales*. Las variables globales, usadas para almacenar la definición de la gramática, facilitan el llamado entre ellas. Permitiendo así, la recursividad en la definición de la gramática.

Los primeros datos que se requieren del usuario (solicitados por

CREATIREL) son el *nombre del tipo de relación* a definir (que se almacena en la variable TIPO) y una *lista con los nombres de las relaciones involucradas* en TIPO. Luego, por cada una de estas relaciones se pide una lista con palabras que la identifiquen.

Cada una de estas listas de palabras, se almacena junto con el *nombre de la relación* y el *nombre del tipo de relación* correspondiente. Para ello se hace uso de las listas de propiedades, que es otra de las facilidades que ofrece Lisp.

Esta tarea está a cargo de las funciones CARGUE y CARGUE1.

Una vez concluído esto, se pasa el control a una función que solicita al usuario las especificaciones necesarias para crear la gramática, esta función recibe el nombre de CREAREL (la cual será descripta más adelante).

Por último, en CREATIREL, se almacenan junto al TIPO:

1- Un vector SGN (que contiene el número de argumentos y la especificación de cada uno de ellos y de la relación).
   SGN tendrá la forma: (n A0 A1 ... An), donde n es el número de argumentos.

2- La gramática creada para ese TIPO.

3- Los nombres de relaciones involucradas en TIPO.

En todos los almacenamientos se hace uso de las listas de propiedades.

Si se quiere hacer uso de este tipo de relaciones posteriormente, para traducir enunciados del castellano a la forma clausular, es necesario contar con toda esta información. Por eso es útil resguardarla.

Finalmente CREATIREL pasa el control a otra función llamada INTER. La descripción de INTER se hará en el capítulo 6.

A continuación describiremos qué hace la función CREAREL mencionada anteriormente.

Esta función, haciendo uso de otras funciones, es la que tiene como objetivo obtener una nueva gramática.

Se mencionan a continuación los pasos que se siguen para concretar la creación de una gramática.

Paso 1: Se pide al usuario que ingrese cuantos argumentos tendrá la relación (n).

Paso 2: Para facilitar la comunicación con el usuario construye un vector (REP) con la forma (rel arg_1 ... arg_n), donde n es el número de argumentos y el arg_i representa al argumento de la posición i. La creación de REP está a cargo de la función CONST.

Paso 3: Como el nombre del predicado no necesariamente debe venir al inicio del enunciado, se pide al usuario que ingrese un índice $i \leq n$ que indique en qué posición, con respecto a los argumentos, vendrá la definición de la relación. Es decir si llamamos "rel" a la especificación de la relación, el vector REP queda:

$$(arg_1 \ldots arg_i \text{ rel } arg_{i+1} \ldots arg_n)$$

Paso 4: Una vez ingresado el índice i, se llama a otra función CONST2 que permite ordenar el vector REP, de tal manera que los arg_i y rel aparezcan en el orden adecuado.

Paso 5: Se llama a una función DEFINOM, utilizada para definir un nombre para la nueva gramática. Más adelante describiremos esta función.

Paso 6: Comienza a construirse la gramática (de manera transparente al usuario), considerando los conectivos explícitos que pueden preceder a una proposición: la implicación (SI) y la negación (NO ES CIERTO QUE), y los cuantificadores: PARA TODO variable, EXISTE UN variable TAL QUE.
Esta función está a cargo de la función CONST7.

Paso 7: Se le pasa el control a la función CONST3, que construye el resto de la gramática. La gramática resultante de esta función se almacena con el nombre obtenido en DEFINOM.

Ahora daremos una breve descripción de la función DEFINOM. Esta función se utiliza para establecer un nombre para la nueva gramática. Esta función trabaja con una lista de nombres de gramáticas que ya existen (lista que recibe el nombre de NOM_GRAMATICAS), y con un valor numérico (que recibe el nombre de CONTADOR) que inicialmente está en cero.
Como ya se mencionó en el capítulo 3, la función despliega un mensaje en la pantalla preguntando al usuario si quiere dar un nombre a la gramática a crearse. La respuesta puede ser SI o NO.

Por SI : el sistema, haciendo uso de NOM_GRAMATICAS, muestra al usuario los nombres de gramáticas que ya existen, y le pide que el nombre que ingrese sea distinto de ellos. Una vez ingresado el nuevo nombre, se lo incorpora a la lista NOM_GRAMATICAS y se lo adopta como nombre de la gramática a crearse.

Por NO : el sistema forma un nombre para gramática concatenando la palabra GRAM y el valor numérico contenido en CONTADOR. Por ejemplo: GRAM0.

El usuario no debe utilizar la palabra GRAM para especificar sus nombres. Es una palabra reservada para el sistema.

Una vez generado el nuevo nombre, se lo incorpora a la lista NOM_GRAMATICAS y se incrementa el valor de CONTADOR en uno (CONTADOR = CONTADOR + 1).

A continuación presentaremos las funciones CONST7 y CONST8.

Las funciones CONST7 y CONST8 se utilizan para agregar automáticamente distintas alternativas a la gramática.

Para lograr una manera general de hacerlo, se definieron dos variables globales: CONECLIS1 y CONECLIS2.

1) En CONECLIS1 se almacenaron (en forma de listas de listas) conectivos explícitos y cuantificadores que pueden preceder a la oración.
2) En CONECLIS2 se almacenaron (en forma de listas de listas) los conectivos explícitos que pueden aparecer entre dos proposiciones.

La función CONST7 es convocada al inicio y al final de la creación de la gramática. Cuando es llamada al inicio, se incorpora a la especificación de la gramática los elementos de la lista CONECLIS1. En cambio, cuando es llamada al final se incorporan a la definición los elementos de la lista CONECLIS2.

CONST7 necesita de tres argumentos, dos nombres de variables globales (que almcenan partes de la gramática) y un valor numérico

que indique si se tomarán los elementos de CONECLIS1 o de CONECLIS2.

Si el número = 1 entonces llama a CONST8 con los nombres recibidos y con CONECLIS2.

Si el número ≠ 1 entonces llama a CONST8 con los nombres recibidos y con CONECLIS1.

Como salida de estas funciones se obtiene que la gramática contemple, de manera transparente al usuario, la posibilidad de recibir conectivos y cuantificadores en las oraciones de entrada.

Continuaremos con la descripción del Editor, presentando la función CONST3. Esta función es muy sencilla, porque hace uso de otras funciones que son las que realmente crean la gramática.

Las dos funciones llamadas por CONST3 son: FUNCAUX y CONST4.

A FUNCAUX la convoca una vez por cada elemento del vector REP, para permitir que el usuario los vaya definiendo.

Como ya dijimos, el Editor trabaja con variables globales.

El usuario puede definir más de una estructura por cada elemento de REP. El sistema utiliza una variable global por cada elemento de REP. Es decir, que en cada variable global guarda todas las alternativas para un elemento de REP.

Se deben crear nombres distintos de variables para que se le puedan ir asignando las definiciones. Con los nombres se identifican y además, por medio de ellos, se van a poder ir conectando las distintas partes de la gramática.

Para crear nombres para las variables globales, se hace uso de la función CREANOM, la cual describiremos más adelante.

Una vez que todos los elementos de REP fueron procesados por

FUNCAUX, la función CONST3 le pasa el control a CONST4.
Al finalizar la ejecución de CONST4, se puede dar por terminado el proceso de creación de la gramática. Entonces, el sistema le pasa nuevamente el control a la función CREAREL y de ésta a CREATIREL.

Para terminar con la descripción del Editor de Gramáticas, presentaremos las funciones que mencionamos. Primero trataremos a FUNCAUX, luego a CONST4 y por último CREANOM.

A- La función FUNCAUX tiene como entrada a un elemento de REP, llamémosle ARG, que se quiere definir y un lista iniciada en NIL (vacío, en LISP), en la cual se irán alamacenando las definiciones dadas por el usuario.
FUNCAUX hace uso de otras funciones: FUNC, TRANSFORMA y FUNCAUX1.

La función FUNCAUX opera de la sigiente manera:
Paso 1: pide al usuario que defina a ARG. El usuario deberá ingresar la definición tal como se especificó en el capítulo 2.
Paso 2: pasa el control a la función FUNC. La salida de esta función es la definición del argumento, solamente con símbolos terminales (que luego se reemplazarán por constantes o variables).
Paso 3: pide al usuario cual es la palabra clave que identifica a ARG. Para ello, despliega todas las clases gramáticales utilizadas para definir a ARG, y espera a que el usuario ingrese cual es la que va a dar a ARG.
Paso 4: con los datos obtenidos en el Paso 3 y en el Paso 4, se pasa el control a la función TRANSFORMA. La salida de esta función es la definición de ARG, expresada en listas.
Paso 5: si ARG = rel se le agrega automáticamente la opción de la

negación implícita.

Paso 6: si ARG = arg_i, i=1,...,n; se le agrega automáticamente la opción de manejo de *variables libres o cuantificadas* en el lugar del arg_i. Es decir, se permite al usuario que se refiera a cada arg_i por medio de variables libres o cuantificadas. Este proceso se cumple con la ayuda de la variable global CONECLIS3, que contiene estas definiciones.

Paso 7: pasa el control a la función FUNCAUX1. Esta función se utiliza para preguntar al usuario si quiere ingresar alguna otra forma de describir a ARG. Si el usuario dice SI, entonces se llama nuevamente a FUNCAUX. Si el usuario dice NO, entonces el control regresa a CONST3.

A continuación describiremos las funciones más importantes usadas por FUNCAUX.

1)- FUNC : esta función recibe como entrada la especificación de ARG dada por el usuario. La salida será esa definición pero expresada con símbolos terminales.

A su vez, esta función hace uso de otras funciones, ellas son: FUNC1, FUCN2, FUNC3, PONEAST, ASTERIS, QUITAST, AVERIGUA, VERIFICA y REEMPLAZA.

No describiremos detalladamente a cada una de ellas. Todas sirven directa o indirectamente a FUNC, y permiten que el usuario pueda usar estructuras complejas para definir los argumentos y la relación y luego utilizar esas mismas estructuras en otras definiciones sin tener que volver a especificarlas.

Ilustraremos mejor esta idea con un ejemplo.

Supongamos que queremos definir una gramática que nos reconozca frases como:

[El alumno]$_{Frase1}$ Garaycochea toma [el curso]$_{Frase1}$ de física

Esta frase representa la instancia de una relación de dos argumentos: el *alumno* y la *materia*.

Se puede ver que al definir los dos argumentos, habrá una estructura que es común a ambos: un artículo seguido de un sustantivo (Frase1).

Entonces, en el momento de definir arg_1, lo podemos hacer de la siguiente manera:

‹arg_1› ::= (‹Frase1› :NOMBRE_PROPIO›)

Cuando el sistema nos pregunte si ‹Frase1› es terminal le decimos que NO. Ahora, el sistema nos preguntará como queremos definir a ‹Frase1›, le decimos:

‹Frase1› ::= (‹ARTDET› ‹SUSCOM›)

El sistema preguntará si los elementos de la lista con la cual especificamos a ‹Frase1› son terminales. Por cada uno de ellos le decimos que SI. Como toda la definición quedó expresada con símbolos terminales, el sistema la usa para reemplazar a ‹Frase1›. Luego sigue el análisis con ‹NOMBRE_PROPIO›. Decimos que es terminal, y la definición final de arg_1 quedará:

(‹ARTDET› ‹SUSCOM› ‹NOMBRE_PROPIO›)

Además, ‹Frase1› se guarda junto con su especificación, para posibles usos posteriores.

Definiremos a arg_2 como:

‹arg_2› ::= (‹Frase1› ‹PREP› ‹SUSCOM›)

Cuando el sistema encuentre a ‹Frase1›, automáticamente la reemplazará por la expresión obtenida durante la definición anterior. Considerando que ‹PREP› y ‹SUSCOM› son terminales,

arg_2 quedará definido de la siguiente manera:

(‹ARTDET› ‹SUSCOM› ‹PREP› ‹SUSCOM›)

Resumiendo, podemos decir que la función FUNC y sus auxiliares se utilizan para especificar estructuras complejas empleadas en la definición de los elementos de REP.
Esto nos permite utilizar estructuras complejas a distintos niveles, y volver a usarlas en otras definiciones sin necesidad de especificarlas nuevamente.

z)- TRANSFORMA : esta función recibe como entrada la salida de FUNC y la palabra clave (ingresada por el usuario) para reconocer a cada ARG. Esta función hace uso de otras dos: FORMAGRAM y CARGACLA.
La primera tiene como entrada una expresión escrita entre angulares " ‹ › ", y su salida es la misma expresión pero escrita como listas de listas.
Por ejemplo, dada la entrada: (‹ARTDET› ‹SUSCOM›), se obtendrá la salida: ((ARTDET) (SUSCOM)).
Su modo de operación es sencillo. Va recorriendo la lista, y a medida que detecta angulares los reemplaza por paréntesis, formando así una lista de listas.
La segunda función que usa ,CARGACLA, tiene como entrada la definición de ARG y de la palabra clave (ambos escritos como listas de listas).
Recorre la lista que corresponde a la definición de ARG. Cuando detecta que una de sus listas es igual a la que corresponde a la palabra clave, agrega junto a esa lista las siguientes listas: (0) y (CARGA). La primera de estas listas, (0), le indica al

Intérprete de la Gramática que la lista que le sigue contiene el nombre de una función que debe ejecutarse. La lista (CARGA) especifica el nombre de una función a ejecutarse. La función CARGA permite que la palabra clave correspondiente a cada ARG se *cargue* o almacene en una pila, para poder obtener luego la representación del enunciado que se está analizando.

Aquí finalizamos la descripción de FUNCAUX y las funciones que dependen de ella. Ahora pasamos a la descripción de la función CONST4.
En CONST3, a medida que se van recibiendo resultados de FUNCAUX, se los van guardando en una lista (RESG).
Cuando todos los elementos de REP fueron definidos, se le pasa el control a CONST4, llevando como argumento a la lista RESG.

B- La función CONST4 hace uso, directa o indirectamente, de las funciones CONST5, CONST6, CONST6_1 y CONST6_2.
El objetivo de este conjunto de funciones es generar la gramática adecuada, según las especificaciones dadas por el usuario, de tal manera que el Intérprete pueda reconocerla.
Para cada uno de los elementos de REP se crea una variable global para que contenga todas sus especificaciones.
Además, para cada elemento de REP se agrega automáticamente la opción para tratar conectivos implícitos. Es decir, se agregará una clase gramatical que permita reconocer conectivos uniendo dos instancias de un mismo argumento o dos definiciones de relaciones. También se incluirá automáticamente una función (MANCON) como parte de la gramática, que procese los conectivos. El tratamiento de los conectivos, ya sean implícitos o explícitos, se presentará

en el capítulo 6.

Para concluir con esta sección presentaremos la función CREANOM.

C- La función CREANOM recibe como entrada una cadena (PAL) y un número (NRO). El objetivo de esta función es crear un nombre nuevo, dada la cadena y el número. Se procede de la siguiente manera:

Si NRO = 0 entonces la salida es la concatenación de la cadena recibida con el número 1. Por ejemplo, sea PAL = "GRAMO" y NRO = 0, la salida será: GRAMO1.

Si NRO ≠ 0 entonces la salida es la cadena recibida donde su último carácter (que es un número) fue incrementado en uno. Por ejemplo, sea PAL = "GRAMO1" y NRO ≠ 0, la salida será: GRAMO2.

Esta función es muy útil para generar nuevos nombres, tanto para variables globales como para renombrar variables (caso que se describe en el capítulo 6).

El tipo de numeración elegido para acompañar a los nombres de variables, se debe al hecho de mantener cierto orden entre los nombres que se vayan generando.

En el caso de las *variables globales* que contienen partes de la gramática resulta muy importante, ya que entre ellas se va transfiriendo el control del proceso a medida que se va recorriendo la gramática.

La enumeración establece, en cierto modo, el orden en el cual se va visitando cada variable global a medida que el recorrido por

la gramática avanza, para realizar el análisis del enunciado.

6- ANALIZADOR SINTACTICO

En este capítulo presentaremos al Analizador Sintáctico.
El Analizador Sintáctico, es la parte del sistema que realiza el análisis sintáctico de los enunciados y obtiene su representación clausular.
A medida que va recorriendo el enunciado, y lo va analizando según la gramática construída como vimos en el capítulo anterior, va procesando los conectivos y los cuantificadores.
Con los conectivos implícitos se obtienen diferentes instancias de una relación.
Al finalizar el análisis del enunciado, se obtiene una representación final en forma de clausulas. Cada clausula tendrá la forma descripta en el capítulo 4. Además a la representación clausular de la proposición le acompañará un vector de variables cuantificadas.
A continuación trataremos más detalladamente los procesos de *análisis sintáctico*, *manejo de conectivos* y de *cuantificadores*.

6.1- Analizador sintactico

Mostraremos el proceso del análisis sintáctico que se efectua sobre el enunciado ingresado por el usuario.
El análisis sintáctico se lleva a cabo según la gramática asociada al tipo de relaciones que se está manejando. La gramática fue creada por el Editor de Gramáticas.

El analizador, además utiliza un *diccionario*. El diccionario se encuentra organizado de la siguiente manera:

Cada palabra es almacenada como una " propiedad " (característica de Lisp). Junto a cada palabra se guarda una lista con sus características gramaticales: clase gramatical, género (o modo), número (o tiempo) y persona. Las características son importantes para una posterior implementación, cuando habrá de considerarse la concordancia entre los accidentes de las palabras.
El hecho de que cada palabra se almacene como una propiedad, proporciona rapidez al sistema. La búsqueda es directa y además, se pueden seguir incorporando nuevas palabras sin afectar la velocidad del sistema.

Recordemos que la gramática queda formada por distintas variables globales, las que a su vez están constituidas por listas de listas. Cada una de las variables globales contiene la definición de un argumento o de la relación.
También dijimos que el usuario puede definir de más de un modo a cada argumento o a la relación. Todas las definiciones alternativas se almacenan en una misma variable global, y en esa variable global se almacenan las alternativas que el sistema agrega automáticamente (para el tratamiento de conectivos y cuantificadores).

Consideramos de interés dar estas aclaraciones antes de entrar en la descripción del Analizador Sintáctico.

A continuación presentaremos un esquema de las funciones que intervienen en el proceso de análisis:

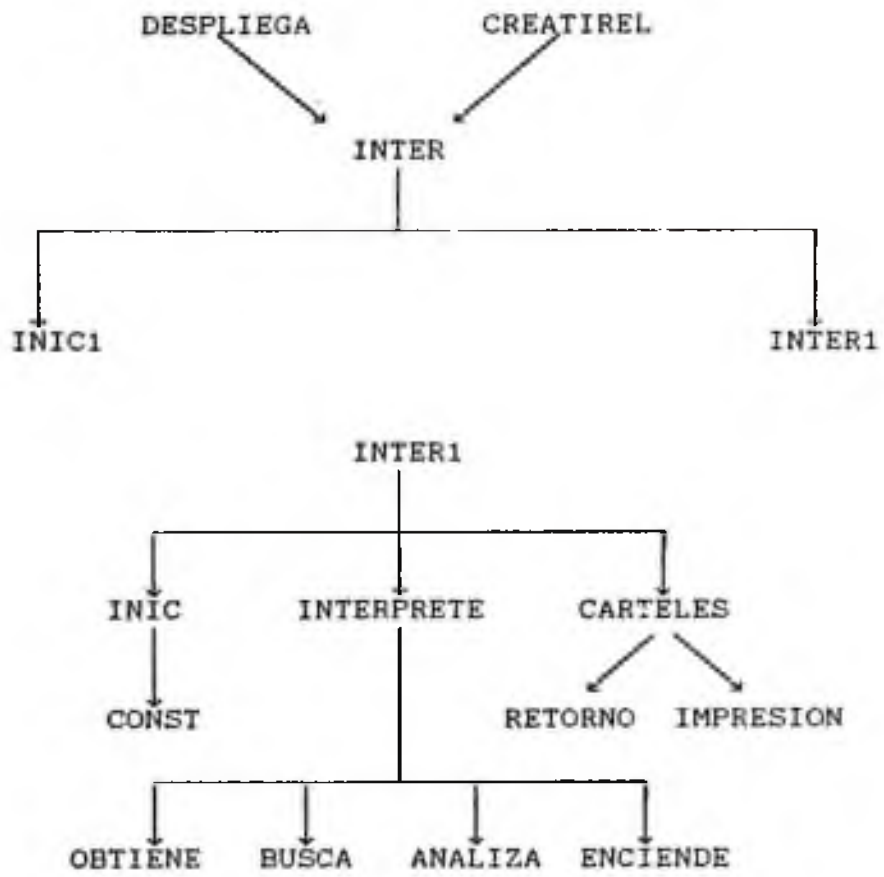


Fig. 6.1

En la figura 6.1, se muestra que a la función INTER se llega desde las funciones DESPLIEGA y CREATIREL.

Se llega desde DESPLIEGA, cuando el usuario elige en el menú principal la opción 1(REVISAR TIPOS DE RELACIONES DEFINIDOS), y decide trabajar con alguno de ellos.

Se llega desde CREATIREL, cuando el usuario elige en el menú principal la opción 2 (CREAR UN TIPO DE RELACIONES). Luego de creada la gramática se llama a INTER.

INTER no realiza el análisis sintáctico. Se lo usa como una función que *comunica* al Analizador Sintáctico con otras funciones del sistema.

INTER despliega un submenú, donde se pregunta al usuario si quiere realizar la traducción de algún enunciado o bien, volver al menú principal.
Si el usuario decide volver al menú principal, entonces se pasa el control a TRESLOG.
Si el usuario decide realizar la traducción a clausulas, de enunciados en castellano, INTER llama a INIC1 y luego a INTER1.

En INIC1 se inician variables que se usarán durante el proceso de análisis y representación clausular de la oración.

En INTER1, primero se llama a INIC. Luego se pide al usuario que ingrese la oración a traducir. Una vez ingresada, se pasa el control a una función llamada INTERPRETE.
Según la salida de INTERPRETE, que será un número, se llama a la función CARTELES.
A continuación explicaremos cada una de estas funciones:

A)- INIC : la función INIC, al igual que la función INIC1, se utiliza para iniciar variables que se usarán durante el resto del proceso. Si bien estas dos funciones cumplen tareas similares, es necesario que operen separadamente ya que no todas las variables necesitan ser iniciadas al mismo tiempo.

B)- INTERPRETE : esta función junto a las que llama, constituyen el cuerpo del Analizador Sintáctico.
INTERPRETE recibe como entrada la oración dada por el usuario y la gramática sobre la cual se hará el análisis.

La salida de INTERPRETE es un número. Dicho número será usado por la función INTER1 como argumento para la función CARTELES.
INTERPRETE opera recorriendo la oración *palabra por palabra*.
Cada vez que una palabra se corresponde con la gramática, la oración se recorta y se sigue analizando el resto.
Cuando se llega al fin de la oración, se da por terminado el proceso de análisis.
Si durante el análisis se produce algún error, entonces se detiene el proceso y se da un mensaje de error.

Supongamos que una oración está formada por n palabras. Llamaremos $palabra_i$ con $i=1,\ldots,n$, a la palabra de la posición i.
Explicaremos de la manera más sencilla posible, cómo opera la función INTERPRETE.

Paso1: Si $palabra_i$ no pertenece al diccionario entonces:

- ← Llamar a OBTIENE
- ← Si la salida de OBTIENE es 0 entonces:

  detener el proceso y la salida es = 1.
- ← En otro caso volver a Paso1

Si $palabra_i$ pertenece al diccionario entonces:

- ← Ir a Paso2.

Paso2: Sea VAR1 la variable en la cual cargamos la clase gramatical de $palabra_i$.
Con VAR1 y la gramática llamar a BUSCA. Esta función recorre la gramática, para determinar si la clase gramatical se ajusta o no a la gramática.
Si la salida de BUSCA es F entonces:

detener el proceso y la salida es = 2.

En otro caso ir a Paso3.

Paso3: Con la salida de BUSCA y VAR1 llamar a ANALIZA.

Si la salida de ANALIZA es F entonces:

detener el proceso y la salida es = 3.

Si la salida de ANALIZA es T y palabra$_{i+1}$ = λ entonces:

el proceso finaliza y la salida es = 4.

En otro caso ir a Paso4.

Paso4: ×– Redefinir el valor de la gramática, asignándole la salida de ANALIZA.

×– Llamar a ENCIENDE.

×– Volver a Paso1..

Antes de presentar a la función CARTELES, trataremos a las funciones que utiliza INTERPRETE:

B.1)– OBTIENE: recibe como argumento la palabra$_i$ que no pertenece al diccionario. Pregunta al usuario si quiere ingresar la palabra desconocida al diccionario.

Si el usuario dice SI, se le piden las características gramaticales de palabra$_i$.

Si el usuario dice NO, la salida será la letra Q.

B.2)– BUSCA: recibe como argumentos a la gramática y a la clase gramatical de la palabra en curso.

La función BUSCA se utiliza para encontrar la parte de la gramática que se corresponde con la clase gramatical.

Como ya dijimos la gramática se encuentra almacenada en distintas variables globales.

BUSCA recorre la gramática. Cuando encuentra que la clase gramatical se corresponde con la gramática, la salida será la lista de la variable global donde se encontró la correspondencia. Si la clase gramatical no se corresponde con la gramática, la salida será F.

B.3)- ANALIZA: recibe como argumentos la salida de BUSCA, llamémosle VAR3, y la clase gramatical de la palabra en curso. VAR3 tendrá la forma de listas de listas (ℓ_1 ℓ_2 ... ℓ_n), donde la primer lista contiene a la clase gramatical.

ANALIZA procede de la siguiente manera:

Si ℓ_2 = (0) entonces se evalúa la función que se indica a continuación. En este caso, ℓ_3 contendrá el nombre de una función. La lista (0) siempre va precediendo una lista que contiene un nombre de acción. Cada vez que el intérprete encuentra una lista (0), ejecuta la lista que le sigue.

Si al ejecutar la función, existe algún error la salida será F.

En caso contrario, se llama recursivamente a ANALIZA con VAR3 recortada (ℓ_3 ... ℓ_n) y la clase gramatical.

Si ℓ_2 = (1) entonces se pasa el control a la variable global, cuyo nombre se encuentra en la lista que sigue a ℓ_2. En este caso ℓ_3 contendrá el nombre de una variable global.

La lista (1) siempre va precediendo una lista que contiene un nombre de variable global. Cada vez que el intérprete encuentra una lista (1), pasa el control a dicha variable global.

La salida de la función será el contenido de la variable global a la cual se le pasó el control.

Si ℓ_3 = (2) entonces se pasa el control a la función BUSCA, llevando como primer argumento la evaluación de ℓ_3 (que será el nombre de una variable global). Con la salida de BUSCA, se llama nuevamente a ANALIZA.

La lista (2) siempre va precediendo una lista que contiene un nombre de variable global.

La diferencia entre (1) y (2), es que en el primer caso se pasa el control a la variable global con palabra$_{i+1}$, mientras que en el segundo caso se pasa el control a la variable global con palabra$_i$. Para cualquier otro valor de ℓ_2, la salida de ANALIZA es la lista VAR3 recortada (ℓ_2 ... ℓ_n).

B.4)- ENCIENDE: cuando existe negación implícita enciende una bandera. Esta bandera se utiliza posteriormente, en el momento de efectuar la *carga* de la palabra clave correspondiente a la relación.

C)- CARTELES: esta función recibe como argumento un número (NUM). Dependiendo del valor de NUM se mostrará un determinado mensaje en la pantalla:

Si NUM = 1 , el usuario no quiso ingresar la palabra$_i$ al diccionario. En la pantalla aparecerá el siguiente mensaje:

SE INTERRUMPE EL ANALISIS YA QUE UD.NO INGRESO LA PALABRA

Si NUM = 2 , se detectó algún error durante el análisis de la oración. En la pantalla aparecerá el siguiente mensaje:

SU ORACION NO CONCUERDA CON LA GRAMATICA

Si NUM = 3 y AVISO3 = 1, una palabra$_i$ que se definió como VARIABLE no comienza con mayúsculas. En la pantalla aparecerá el siguiente mensaje:

SU VARIABLE NO COMIENZA CON MAYUSCULA

Si NUM = 3 y AVISO3 ≠ 1, el usuario no quiso ingresar el nombre de la relación, durante el proceso de análisis. En la pantalla aparecerá el siguiente mensaje:

EL ENUNCIADO NO SE TRADUCE

Si NUM = 4 , el análisis se llevó a cabo con éxito. En la pantalla aparecerá el siguiente mensaje:

LISTA DE CLAUSULAS FORMADAS

y se llamará a la función IMPRESION para que imprima las cláusulas. Luego se imprime el vector de variables cuantificadas.

Si NUM = 5 , aparecerá en la pantalla el siguiente mensaje:

SU ELECCION NO ES APROPIADA

En todas las funciones del sistema donde se dan al usuario distintas opciones para que elija una, y éste ingresa una opción no contemplada se llama a CARTELES con el 5.

En cada una de las alternativas de CARTELES , se llama a la función RETORNO.

A continuación presentaremos las funciones llamadas desde CARTELES.

C.1)- IMPRESION: recibe como entrada la lista de cláusulas formadas durante el proceso de análisis.

Es una función muy simple, toma el primer elemento de la lista de entrada y lo imprime. Luego recorta la lista y repite el proceso hasta que la lista quede vacía.

C.2)- RETORNO: es una función usada para que el usuario indique que el proceso debe continuar.

Muestra en la pantalla el siguiente mensaje:

PARA CONTINUAR OPRIMA LA TECLA S :

y se queda esperando hasta que se pulse la tecla S.

Hasta aquí presentamos cómo se efectúa el análisis sintáctico de enunciados, y qué funciones intervienen en dicha operación. Este analizador sintáctico es muy similar al presentado en [1], que se utilizó para procesar frases en castellano sobre Química.
A continuación, presentaremos cómo se tratan los conectivos durante el proceso de análisis.

6.2- Manejo de Conectivos

Durante el proceso de creación de la gramática, se agregan acciones para el tratamiento de conectivos, que se irán ejecutando a medida que el Analizador Sintáctico recorra la gramática.
Antes de entrar en la descripción del manejo de conectivos implícitos, mencionaremos las acciones relacionadas:

CARGA: esta función se utiliza para ir *guardando* las palabras claves a medida que se va recorriendo la oración. Por cada argumento y por cada relación habrá una palabra clave que los identifique, serán esas palabras claves las que se irán almacenando en una pila (REL) durante el proceso de análisis.

MANCON: esta función se utiliza para manejar los conectivos implícitos encontrados en la oración.
Con los conectivos y con las palabras claves forma una pila por cada argumento y una por cada relación. Cada pila será una lista de listas, y se interpretará que dichas listas están conectadas por *disyunciones*, y los elementos dentro de una sub_lista están

conectados por *conjunciones*.

Los conectivos implícitos reconocidos son: y, e (conjunción), o , u (disyunción) y * (para reemplazar la *coma*, que es una palabra reservada de LISP).

A continuación presentaremos una descripción del algoritmo de esta función.

MANCON trabaja con dos pilas: CL y CM.

En CM se van guardando las palabras claves que aparecen antes de una coma " * ".

En CL se van guardando las sub_listas.

También trabaja con una lista PI, que contiene nombres de variables: Pi_1 , ... , Pi_{n+1}. A cada una de estas variables, a medida que avanza el proceso, se les va asignando la lista de palabras claves correspondientes a un argumento o a la relación.

Además trabaja con una variable auxiliar, PILAUX.

Sea $palabra_i$ la palabra en curso. El algoritmo es el siguiente:

Si $palabra_i$ = * entonces:

- Poner el contenido de REL en CM.
- REL = λ

Si ($palabra_i$ = y) ∨ ($palabra_i$ = e) entonces:

- CL= CM
- CM = λ
- Poner el contenido de REL en CL
- REL = λ

Si ($palabra_i$ = o) ∨ ($palabra_i$ = u) entonces:

- Poner el contenido de REL en CL
- REL = λ
- Si CM ≠ λ entonces:
  - Poner cada uno de los elementos de CM en PILAUX
  - Poner CL en PILAUX

-CL = λ
-CM = λ

✗- En otro caso:
-Poner CL en PILAUX
-CL = λ

En otro caso: ✗- Poner el contenido de REL en CL
✗- REL = λ
✗- Si PILAUX = λ entonces:

-Asignar al primer elemento de PI,CL
-CL = λ
- Recortar PI

✗- En otro caso:

-Poner el contenido de CL en PILAUX.
-Asignar al primer elemento de PI,
PILAUX
-CL = λ
-PILAUX = λ

Suponiendo que estamos trabajando con una relación de n argumentos. Luego de procesarse todos los conectivos implícitos que pudieran aparecer en la oración, se obtendrán n+1 listas.

En la variable global CONECLIS2, utilizada por la función CONST7, existen ciertas funciones incluidas junto a la descripción de la gramática.
Esas funciones son usadas para obtener la representación de las proposiciones (CARGAREP) y además, manejar los conectivos explícitos (MANCON1).
Cada vez que se encuentra un conectivo explícito y al finalizar el análisis de la oración, se llama a la función CARGAREP para que construya la representación correspondiente a la proposición y posteriormente a MANCON1, para que opere sobre las representaciones según el conectivo explícito encontrado.
A continuación presentaremos a estas dos funciones.

A)- CARGAREP: esta función es llamada durante el análisis. Forma parte de la gramática. En el momento de ejecutarse pasa el control a otra función llamada FORMAPROPOS. Si durante la ejecución de este proceso se produce algún error, la salida será F. Lo que producirá finalmente una llamada a CARTELES con un 3.

Si no se originan errores, se van guardando las salidas de FORMAPROPOS en una variable (REP) para luego usarse en la representación final del enunciado.

Presentemos en una gráfica la manera en la que se distribuyen y relacionan las funciones manejadas a partir de CARGAREP.

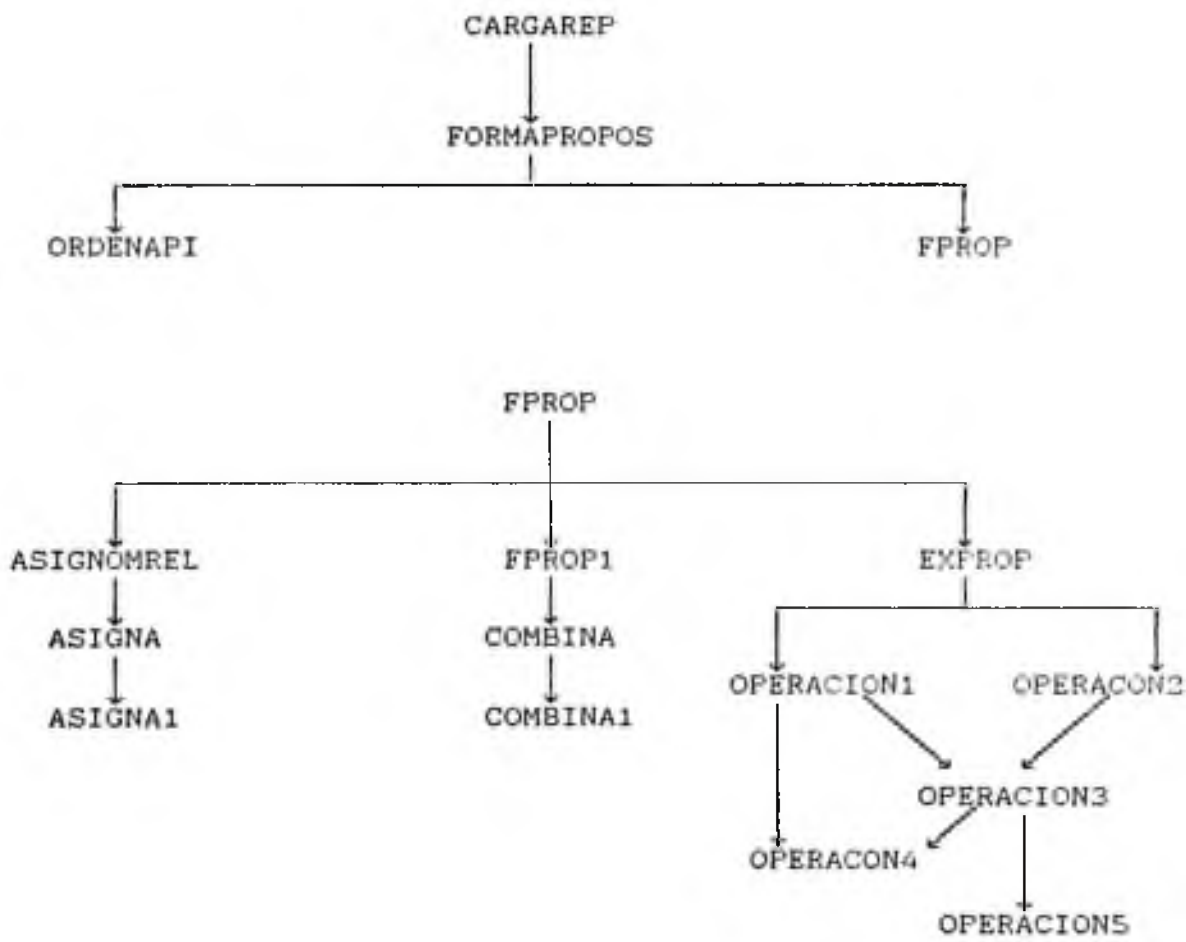


Fig. 6.2

Trataremos ahora la función FORMAPROPOS.

A.1)-FORMAPROPOS: esta función hace uso de otras dos funciones: ORDENAPI y FPROP.

En ORDENAPI se altera el orden del vector PI. El vector PI tiene la forma (Pi_1 ... Pi_{n+1}), donde cada Pi_j se utiliza para guardar las palabras claves de la relación y de los argumentos de la misma.

La salida de ORDENAPI es un nuevo vector PI con la forma: (Pi_{i+1} Pi_1 ... Pi_{n+1}). El primer elemento de PI será el nombre de la pila donde se almacenaron las palabras claves que dan los nombres de la relación.

Una vez reordenado PI, se lo usa como entrada para la función FPROP.

El objeto de la función FPROP es crear la representación de las distintas instancias de una relación. Para ello, hace uso de las listas obtenidas por MANCON y del vector PI (que le dará el orden de los argumentos de la relación).

Cada proposición atómica tendrá la forma (R a_1...a_n), donde R es el nombre de la relación y los a_i son los argumentos de la misma.

La representación será en una forma normal conjuntiva. Al finalizar el análisis de toda la oración se obtendrá su representación clausular.

Tal como se muestra en la Fig.6.2, FPROP hace uso de otras funciones: ASIGNOMREL, FPROP1 y EXPROP.

1)- ASIGNOMREL: esta función recibe como argumento una lista con palabras claves correspondientes a la relación. Con cada una de

las palabras claves llama a la función ASIGNA.

ASIGNA opera de la siguiente manera: dada una palabra clave, averigua a cual relación hace referencia. La salida será el nombre de dicha relación.

La función ASIGNA hace uso de ASIGNA1, cuando la palabra clave en cuestión no tiene asociado ningún nombre de relación.

ASIGNA1 da al usuario las siguientes opciones:

1- Ingresar el nombre de la relación correspondiente a la palabra clave.

2- Ingresar algún sinónimo de la palabra clave, que pueda ayudar a encontrar el nombre de relación.

3- Interrumpir el proceso.

Si el usuario elige la primer opción, se le pide que ingrese el nombre de la relación. Este nombre será la salida de la función y además se almacenará con los nombres de relaciones definidos para el TIPO que se está tratando.

Si el usuario elige la segunda opción, se le pide que ingrese un sinónimo de la palabra clave. Con este sinónimo se llama a ASIGNA. Si la salida de ASIGNA es la palabra vacía, se vuelve a llamar a ASIGNA1. En caso contrario, se guarda la palabra clave con el nombre de relación encontrado a partir del sinónimo, y dicho nombre constituirá la salida de la función.

Si el usuario elige la tercer opción, el proceso se interrumpirá y se llamará a CARTELES con un 3.

La función ASIGNA1, con la opción 1 permite seguir incorporando nombres de relación a un TIPO ya definido. Y con la opción 2, permite incrementar el número de palabras claves que identifiquen a una misma relación.

2)- FPROP1: Junto con las funciones COMBINA y COMBINA1 tiene por finalidad realizar combinaciones entre los elementos de las Pi_j, para obtener todas las instancias de una relación.

Cada elemento de Pi_1 se combina con cada elemento de Pi_2, y el resultado de esta operación se combina con cada elemento de Pi_3 y así sucesivamente.

Además, según la forma de cada Pi_j se originarán distintas listas durante las combinaciones. Esto dependerá si se trata de conjunciones o de disyunciones.

3)- EXPROP: una vez obtenida la salida de FPROP1, que será una lista que contenga las combinaciones entre los elementos de las Pi_j, se llama a EXPROP. Esta función, tiene por objeto construir una representación en forma normal conjuntiva de las proposiciones analizadas.

Cuando presentamos a MANCON, dijimos que las palabras claves unidas por conjunciones formaban parte de una misma lista, mientras que las que se encontraban conectadas por disyunciones formaban parte de listas distintas.

Manteniendo este criterio, en FPROP1 se forman las combinaciones entre las palabras claves dando origen a listas de listas en el caso de las conjunciones y a listas con elementos atómicos en el caso de las disyunciones.

Por ello, para decidir cual proceso aplicar en EXPROP se considera la longitud de cada sub-lista de la lista de entrada.

Si la longitud es igual a uno, estamos en presencia de una disyunción. En caso contrario, estamos en presencia de una conjunción.

Llamemos REPRES a la variable en la cual se irán almacenando las

representaciones de las proposiciones, y LISTA a la lista formada en FPROP1, que es la entrada de EXPROP.

El algoritmo que sigue EXPROP es el siguiente:

```
INICIO
  ×—Mientras LISTA ≠ λ
       Hacer:
            ELEM = primer elemento de LISTA
            Si longitud de ELEM  = 1 entonces:
                        - Poner ELEM en REPRES
                        - Recortar LISTA
                        - Volver a INICIO
            Si longitud de ELEM  ≠ 1 entonces:
                        - Mientras ELEM ≠ λ
                             Hacer:
                                - Poner los dos primeros
                                  elementos de ELEM en REPRES.
                                - Llamar a OPERACION2.
                                - Recortar a ELEM en dos
                                  elementos.
                             Fin
                        - Recortar LISTA
                        - Volver a INICIO
       Fin
  ×—Llamar a OPERACION1
FIN
```

Como se ve en el algoritmo, la función EXPROP hace uso de dos funciones: OPERACION1 y OPERACION2, para obtener la representación de las proposiciones atómicas. Pero por ahora serán expresadas en

forma normal conjuntiva.
A OPERACION1 se la llama al finalizar el recorrido de la LISTA.
A OPERACION2 se la llama para que trate las conjunciones.
Estas dos funciones procesan las proposiciones atómicas. Si están negadas las expresan en la forma correspondiente.
Toda proposición atómica p se representa por la lista (λ p), y toda proposición atómica negada $\neg p$ se representa por la lista (p λ).

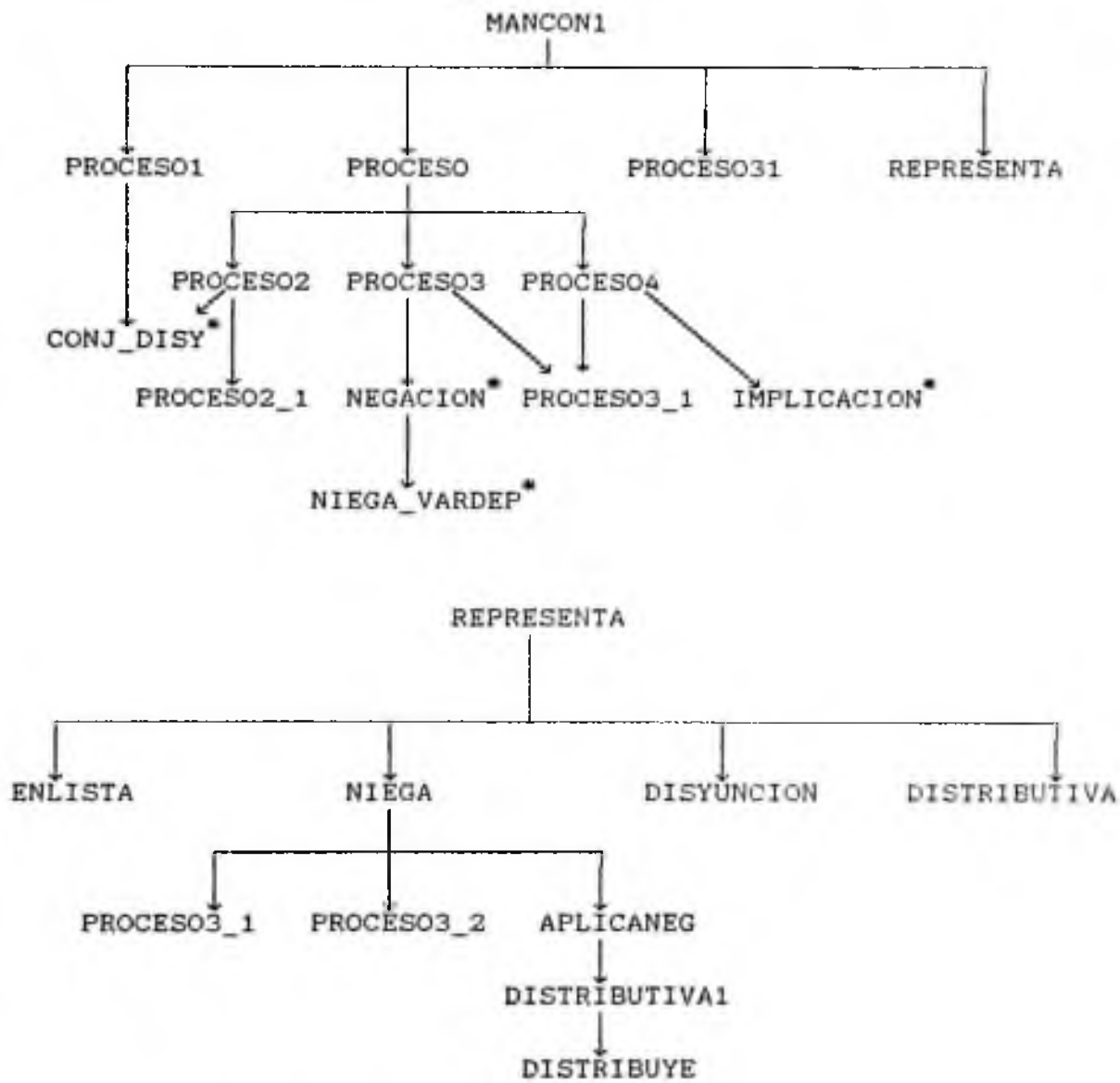
Hasta aquí tratamos de explicar cómo se manejan los conectivos implícitos de una oración, cuáles son las prioridades entre ellos y cuál es la representación que se obtiene.
A continuación y para terminar con esta sección, presentaremos cómo se manejan los conectivos explícitos.

B)- MANCON1: esta función forma parte de la gramática, y se usa para tratar los conectivos explícitos del enunciado. Los conectivos explícitos son: *conjunción* (Y, E), *disyunción* (O, U), *negación* (NO ES CIERTO QUE) e *implicación* (SI-ENTONCES).
Entre los conectivos existe cierta prioridad. La misma se corresponde, de mayor a menor, con el orden en el cual fueron enunciados.
Según el orden de prioridad, será el orden en el cual se los trate.
En nuestro caso el primer conectivo que se procesará, si existe, será la conjunción, luego la disyunción y así sucesivamente.
Para implementar estas acciones se definieron varias funciones, que operan con las representaciones de las proposiciones según los conectivos que se vayan encontrando durante el transcurso del

análisis.

Además, se llaman a funciones que tratan a los vectores de variables cuantificadas de acuerdo al conectivo. Estas últimas funciones las trataremos en la siguiente sección, cuando presentemos el manejo de variables y de cuantificadores.

A continuación presentaremos una gráfica, ilustrando la distribución de las funciones:



* Estas funciones se explicarán en la sección 6.3

Fig. 6.3

Cuando se llama a MANCON1 es porque la palabra en curso de la

oración es un conectivo explícito, o bien porque hemos llegado al fin de la oración. En este último caso debemos procesar los operadores que pudieran haber quedado pendientes y luego hallar la representación clausular.

Se trabaja con una pila, OPER en la cual iremos guardando los conectivos encontrados.

MANCON1 hace uso de otras funciones para poder obtener la representación adecuada según los conectivos.

Una vez llegado al término del análisis la proposición queda expresada aún en una forma normal conjuntiva. Pero luego se le pasa el control a la función REPRESENTA, la cual obtiene la representación clausular que puede ser usada por el traductor a Prolog.

El orden en el cual se procesan los conectivos tiene relación directa con la prioridad establecida entre ellos.

El algoritmo de MANCON1 es el siguiente:

Si (palabra$_i$ = Y) $\vee$ (palabra$_i$ = E) entonces:

- Llamar a PROCESO1
- Poner palabra$_i$ en OPER

Si (palabra$_i$ = O) $\vee$ (palabra$_i$ = U) entonces:

- Llamar a PROCESO con un 2

Si (palabra$_i$ = NO ES CIERTO QUE) entonces:

- Llamar a PROCESO31

Si (palabra$_i$ = SI) entonces:

- Poner palabra$_i$ en OPER

Si (palabra$_i$ = ENTONCES) entonces:

- Llamar a PROCESO con un 4

En otro caso:

- Mientras OPER ≠ λ

  Hacer:

  Llamar a PROCESO con un 5

  Fin

- Llamar a REPRESENTA

A continuación presentaremos una descripción de cada uno de estos procesos.

Sea REP la variable en la cual se fueron almacenando las salidas de FORMAPROPOS.

1)- PROCESO1: se utiliza para procesar las conjunciones explícitas. Su algoritmo es el siguiente:

- Si el primer elemento de OPER es una conjunción, entonces:

  X- Llamar a una función que opere con los vectores de variables cuantificadas.

  X- X1 = (pop REP)
  X2 = (pop REP)
  X = concatenación de X1 y X2
  Poner X en REP
  OPER = (cdr OPER)

  X- La salida será T

- En otro caso la salida será NIL (λ).

La representación de la conjunción de dos proposiciones es la concatenación de las respectivas representaciones.

2)- PROCESO31: se utiliza para procesar las negaciones explícitas. El algoritmo seguido por PROCESO31 es muy simple. Lo presentamos a continuación:

- Si el primer elemento de OPER es una negación, entonces:

  OPER = (cdr OPER)

- En otro caso: Poner palabra en OPER.

Si el último conectivo almacenado en OPER es una negación, se lo elimina. Es decir, si palabra$_4$ es una negación y está pendiente una negación sobre la misma proposición, por la equivalencia : $\neg(\neg p) \equiv p$, ambas se eliminan.

3)- PROCESO: esta función recibe como entrada un número entero De acuerdo a las prioridades entre los conectivos, se llaman en un cierto orden a determinados procesos. Estos procesos tendrán a su cargo el tratamiento de los conectivos almacenados en OPER.

Su algoritmo es el siguiente:

Si COD = 2 (palabra$_4$ = disyunción) entonces:

- Si (PROCESO1 OR PROCESO2) entonces Poner palabra$_4$ en OPER
- En otro caso: Poner palabra$_4$ en OPER

Si COD = 4 (palabra$_4$ = implicación) entonces:

- Si (PROCESO1 OR PROCESO2 OR PROCESO3 OR PROCESO4) entonces:
  - Entra nuevamente a Proceso con COD = 4.
- En otro caso: Poner palabra$_4$ en OPER.

Si COD = 5 entonces:

- Si (PROCESO1 OR PROCESO2 OR PROCESO3 OR PROCESO4) entonces:
  - La salida es T

El orden en el que aparecen los procesos en el OR tiene relacion directa con las prioridades establecidas entre los conectivos. Por ello el primer proceso a ejecutarse será el de la conjunción, luego el de la disyunción y así sucesivamente.

4)- PROCESO2: se utiliza para procesar las disyunciones explícitas. Toda disyunción es tratada como una conjunción, de acuerdo con la siguiente equivalencia: $(p \vee q) \equiv \neg(\neg p \wedge \neg q)$.

Es decir, toda disyunción puede expresarse como la negacion de la

conjunción de las proposiciones negadas.

El algoritmo es el siguiente:

Si el primer elemento de OPER es una disyunción, entonces:

- Llamar a una función que procese los vectores de variables cuantificadas.
- OPER = (cdr OPER)
  X1 = (pop REP)
  X2 = (pop REP)
- Llamar a PROCESO2_1. La salida de esta función es la negación de X1 y de X2.
  X = concatenación de la salida de PROCESO2_1.
- Poner X en REP
- La salida es T.

En caso contrario la salida será NIL (λ).

5)- PROCESO3: se utiliza para procesar las negaciones explícitas.

El algoritmo es el siguiente:

Si el primer elemento de OPER es una negación, entonces:

- Llamar a una función que procese los vectores de variables cuantificadas.
- OPER = (cdr OPER)
  X1 = (pop REP)
- Llamar a PROCESO3_1. La salida de esta función será la negación de X1.
- Poner la salida de PROCESO3_1 en REP.
- La salida es T

En otro caso la salida será NIL (λ).

6)- PROCESO4: se utiliza para procesar las implicaciones. Toda implicación puede ser expresada por medio de conjunciones y negaciones, de acuerdo a las siguientes equivalencias:

$$(p \rightarrow q) \equiv (\neg p \vee q) \equiv \neg(\neg\neg p \wedge \neg q) \equiv \neg(p \wedge \neg q)$$

El algoritmo es el siguiente:

Si el primer elemento de OPER es una implicación, entonces:

- ✗— Llamar a una función que procese los vectores de variables cuantificadas.
- ✗— X1 = (pop REP)
  X2 = (pop REP)
  OPER = (cddr OPER)
- ✗— Llamar a PROCESO3_1. La salida de esta función será la negación de X1. La representación quedará expresada finalmente, por medio de conjunciones y negaciones.
- ✗— La salida será T.

En otro caso la salida será NIL (λ).

En MANCON1, cuando OPER = λ se llama a la función REPRESENTA. Esta función trabaja sobre la representación (REP) obtenida durante el análisis de la oración.

REP se encuentra expresada en una forma normal conjuntiva. REPRESENTA junto a las funciones que utiliza, convierte esta representación a una forma clausular.

Cada cláusula tendrá la forma descrita en el capítulo 4.

Para obtener la forma clausular, a partir de la forma normal conjuntiva, se sigue el siguiente procedimiento, ya descrito en la sección 4.1.

1) Aplicar negación

2) Eliminar tautologías

3) Aplicar disyunción

Una vez efectuada la transformación, el enunciado ingresado en lenguaje natural queda traducido a cláusulas del predicado de primer orden.

6.3)- Manejo de cuantificadores

Presentaremos en esta sección el manejo de variables y de

cuantificadores.

Con respecto a las variables, se reconocen por comenzar con una letra mayúscula. Su tratamiento es similar al de cualquier objeto.

En las variables globales CONECLIS1 y CONECLIS3 ,están almacenadas todas las formas en las que pueden aparecer expresadas las variables (libres o cuantificadas). En CONECLIS3 se agregaron junto a cada aparición de la lista " (VARIABLE) ", las listas " (0) (ACCION1) ".

La primera de ellas, como ya se explicó, indica que la lista que le sigue contiene el nombre de una acción. En este caso, ACCION1 es el nombre de una función que en el momento de ejecutarse verificará si la variable precedente comienza con mayúsculas.

Si detecta que el primer carácter del nombre de la variable, no es una letra mayúscula enciende una bandera (AVISO3). Por lo tanto, la función CARTELES despliega el mensaje:

SU VARIABLE NO COMIENZA CON MAYUSCULA

Trataremos ahora los cuantificadores. Los cuantificadores usados son de dos tipo: los *universales*(UC) y los *existenciales*(EC).

UC : TODO, CUALQUIER, LOS, LAS, CUALQUIERA

EC : UN, UNO, UNA, NINGÚN, POCOS

En el momento de crear la gramática, el Editor de Gramáticas incorpora automáticamente las distintas opciones para manejar cuantificadores.

Los cuantificadores pueden aparecer al comienzo del enunciado junto a las variables, o bien dentro del enunciado.

En el primer caso, contemplado en CONECLIS1, se tiene la siguiente estructura:

((PUC) (UC) (VARIABLE) (0) (ACCION1) (0) (UC_FUNC) (0) (CARGA))

((PEC) (EC) (VARIABLE) (0) (ACCION1) (0) (EC_FUNC) (0) (CARGA))

((SEP) (PRON))

donde: PUC : PARA

PEC : EXISTE

SEP : TAL

PRON : QUE

En el segundo caso, contemplado en CONECLIS3, se tiene la siguiente estructura:

((VARIABLE) (0) (ACCION1) (0) (LIB_FUNC) (0) (CARGA))

((UC) (VARIABLE) (0) (ACCION1) (0) (UC_FUNC) (0) (CARGA))

((EC) (VARIABLE) (0) (ACCION1) (0) (EC_FUNC) (0) (CARGA))

En CONECLIS3, se consideran tres opciones para las variables:

a) que sean *libres*: las variables no están afectadas por cuantificadores.

b) que aparezcan cuantificadas *existencialmente*: las variables están precedidas por un cuantificador existencial.

c) que aparezcan cuantificadas *universalmente*: las variables están precedidas por un cuantificador universal.

En cada una de estas opciones se agrega una función que trate los cuantificadores y que permita ir formando el vector VAR_DEP.

Esas funciones son: LIB_FUNC, EC_FUNC y UC_FUNC.

A medida que se realiza el análisis del enunciado se van reconociendo las variables y se va formando el vector VAR_DEP.

Para ello se procede de la siguiente manera:

Sea X_i la variable de la posición i, entonces ...

$$VAR\_DEP(X_i) = \begin{cases} \text{LIBRE} & \text{si } X_i \text{ es una variable libre} \\ X_i & \text{si } X_i \text{ aparece cuantificada universalmente} \\ \text{Un conjunto de variables precedentes a } X_i \text{ cuantificadas universalmente} & \text{si } X_i \text{ aparece cuantificada existencialmente} \end{cases}$$

Toda fórmula se escribe en su forma PRENEX. Las variables cuantificadas existencialmente se escriben como una función SKOLEM de las variables precedentes cuantificadas universalmente.

Si la oración contiene variables, las acciones que aparecen en CONECLIS1 o en CONECLIS3 se van ejecutando a medida que avanza el análisis.

Al finalizar el análisis se habrá obtenido un vector VAR_DEP por cada proposición. Si existen proposiciones conectadas por medio de conectivos explícitos, entonces se opera sobre los vectores VAR_DEP para obtener un vector que sea representativo de todo el enunciado. Este proceso se explicará más adelante.

Antes de presentar las funciones que forman VAR_DEP durante el proceso de análisis, explicaremos el caso en el cual resulta necesario *renombrar* las variables.

Se establece que cada cuantificador aplicado a una variable tiene vigencia sobre ella hasta el final del proceso, o bien hasta que la misma variable aparezca afectada por otro cuantificador.

La generación de nombres nuevos se lleva a cabo por medio de la función CREANOM, la cual fue explicada en el capítulo 5.

El proceso de *renombrado* se lleva a cabo en las mismas funciones donde se van tratando los cuantificadores.

Ahora presentaremos a cada una de ellas y daremos también, una explicación más detallada de cómo se produce el cambio de nombre

de las variables.

La primera función que mostraremos será LIB_FUNC, destinada a tratar variables libres; seguiremos con EC_FUNC, que trata las variables cuantificadas existencialmente, y por ultimo presentaremos a UC_FUNC que trata las variables cuantificadas universalmente.

Estas funciones operan con las siguientes variables:

- LISTA_DE_VARIABLES: en la cual se van almacenando los nombres de variables que se van encontrando durante el análisis.
- NOMBRES_NUEVOS: en el caso que hubiera necesidad de renombrar a alguna variable, se almacenará en esta lista el nombre de la variable junto al nuevo nombre.
- VARNEG: cada vez que se detecte una negación explícita (las que pueden afectar a las variables cuantificadas) se enciende una bandera. A medida que se encuentran las variables, si la bandera está encendida también se almacenan en VARNEG.
- VAR_DEP: en esta variable se irán almacenando las variables junto a una expresión que depende del cuantificador que la está afectando.
- CUAN_LISTA: en esta variable se irán resguardando el VAR_DEP de cada proposición. Si existen conectivos explícitos se opera sobre los vectores almacenados en CUAN_LISTA.

1)- LIB_FUNC: esta función aparece junto a las variables que no están precedidas por cuantificadores, es decir son variables libres. El algoritmo seguido es el siguiente:

Si palabra $\notin$ LISTA_DE_VARIABLES entonces:

$\leftarrow$ VARCU = λ

× Poner "LIBRE" en VARCU
× Poner palabra en VARCU
× Poner VARCU en VAR_DEP
× Poner palabra en LISTA_DE_VARIABLES
Si SENEG = 1 entonces: Poner palabra en VARNEG
En otro caso va al fin

En otro caso va al fin.

Cuando se ejecuta esta función, en VAR_DEP se almacenará una lista con la siguiente forma: (X LIBRE), donde X es el nombre de la variable de la posición i y LIBRE indica que X es una variable libre.

2)- EC_FUNC: esta función aparece junto a las variables cuantificadas existencialmente. Como todas las variables cuantificadas existencialmente se deben expresar como una función SKOLEM de las variables precedentes cuantificadas universalmente. El algoritmo seguido es el siguiente:

INICIO

Si palabra no pertenece a LISTA_DE_VARIABLES entonces:

× Llamar a EC_FUNCAUX
× Ir a FIN

En otro caso:

× Llamar a BUSCA_NOMNUE, para que verifique si a partir de palabra se creó un nuevo nombre.

× Si la salida de BUSCA_NOMNUE = λ entonces:

× Llamar a CREANOM con palabra y un cero como argumentos.
× Poner el nuevo nombre en VARCU
× Poner palabra en VARCU
× Poner VARCU en NOMBRES_NUEVOS.
× Llamar a EC_FUNCAUX con el nuevo nombre.
× Ir a FIN

× En otro caso:

```
                    ×- Llamar a CREANOM con un uno y la
                       salida BUSCA_NOMNUE como argumentos.
                    ×- Poner el nuevo nombre en VARCU
                    ×- Poner palabra_i en VARCU
                    ×- Poner VARCU en NOMBRES_NUEVOS
                    ×- Llamar a EC_FUNCAUX con el nuevo
                       nombre.
                    ×- Ir a FIN
```

Daremos a continuación el algoritmo de EC_FUNCAUX: esta función recibe como entrada un nombre de variable (NOMVAR).

```
INICIO

     VARCU = λ
     Poner IVAR en VARCU
     Poner NOMVAR en VARCU
     Poner VARCU en VAR_DEP
     Poner NOMVAR en LISTA_DE_VARIABLES
     IVAR = λ
     Si SENEG = 1 entonces: - Poner NOMVAR en VARNEG

                            - Ir a FIN
FIN
```

Cuando se ejecuta esta función, en VAR_DEP se almacenará una lista con la siguiente forma: $(X_i\ (Y))$, donde X_i es el nombre de la variable de la posición i, e Y es una expresión igual a:

- la palabra vacía, si $X_j \ldots X_k$ $k < i$ no están cuantificadas universalmente.
- $X_j \ldots X_k$ si están cuantificadas universalmente.

g)- UC_FUNC: esta función aparece junto a las variables cuantificadas universalmente. Las variables se van almacenando en una variable llamada IVAR que será usada por EC_FUNCAUX, si existieran variables cuantificadas existencialmente.

El algoritmo seguido por UC_FUNC es el siguiente:

```
INICIO

     Si palabra_i no pertenece a LISTA_DE_VARIABLES entonces:

                    ×- Llamar a UC_FUNCAUX con palabra_i
```

```
            ×– Ir a FIN

      En otro caso:
            ×– VARCU = λ
            ×– Llamar a BUSCA_NOMNUE para que verifique si a
               partir de palabra se creó un nuevo nombre

            ×– Si la salida de BUSCA_NOMNUE = λ entonces:
                        ×– Llamar a CREANOM con palabra y un
                           cero como argumentos
                        ×– Poner el nuevo nombre en VARCU
                        ×– Poner palabra en VARCU
                        ×– Poner VARCU en NOMBRES_NUEVOS
                        ×– Llamar a UC_FUNCAUX con el nuevo
                           nombre
                        ×– Ir a FIN

            ×– En otro caso:
                        ×– Llamar a CREANOM con la salida de
                           BUSCA_NOMNUE y un uno como argumentos
                        ×– Poner el nuevo nombre en VARCU
                        ×– Poner palabra en VARCU
                        ×– Poner VARCU en NOMBRES_NUEVOS
                        ×– Llamar a UC_FUNCAUX con el nuevo
                           nombre
                        ×– Ir a FIN

FIN
```

Como vemos UC_FUNC es similar a EC_FUNC. La única diferencia entre ellas es la acción en ellas de UC_FUNCAUX y EC_FUNCAUX respectivamente.

Daremos a continuación el algoritmo de UC_FUNCAUX: esta función recibe como argumento el nombre de una variable (NOMVAR).

```
INICIO

      VARCU = λ
      Poner NOMVAR en VARCU
      Poner NOMVAR en VARCU
      Poner VARCU en VAR_DEP
      Poner NOMVAR en LISTA_DE_VARIABLES
      Poner NOMVAR en IVAR
      Si SENEG = 1  entonces:  – Poner NOMVAR en VARNEG
                               – Ir a FIN
FIN
```

Cuando se ejecuta UC_FUNC, en VAR_DEP se almacenará una lista con

la siguiente forma: (X_i X_i), donde X_i es el nombre de la variable de la posición i.

Para finalizar con esta sección, presentaremos cómo se opera sobre los vectores VAR_DEP cuando existen conectivos explícitos conectando dos o más proposiciones.

1) Si el conectivo es una *conjunción* o una *disyunción*, los VAR_DEP corrrespondientes a las proposiciones se concatenan.

2) Si el conectivo es una *negación*, el VAR_DEP correspondiente se obtendrá de la siguiente manera:

$$\text{VAR\_DEP}(X_i)_{\neg Y} = \begin{cases} X_i & \text{si VAR\_DEP}(X_i) \neq X_i \\ \text{el conjunto de variables precedentes a } X_i \text{, cuantificadas existencialmente.} & \text{en otro caso} \end{cases}$$

3)- Si el conectivo es una *implicación*, los VAR_DEP correspondientes a las proposiciones se tratan como una disyunción de la negación del antecedente con el consecuente: $(Y \rightarrow Z \equiv \neg Y \vee Z)$

Las funciones encargadas de realizar estas operaciones son invocadas desde los procesos utilizados para tratar los conectivos explícitos (fueron presentadas en la figura 6.3).

A continuación daremos una explicación de cada una de ellas:

1)- CONJ_DISY: se usa para tratar tanto *conjunciones* como *disyunciones*. La salida de esta función, será la concatenación de los vectores VAR_DEP de las proposiciones afectadas por el conectivo lógico.

Se procede de la siguiente manera:

REP1 = Pop CUAN_LISTA

REP2 = Pop CUAN_LISTA

Poner la concatenación de REP1 y REP2 en CUAN_LISTA.

2)- NEGACION: se usa para tratar la negación explícita. La salida de esta función será el vector VAR_DEP procesado de acuerdo a las variables afectadas por la negación. Se hace uso de VARNEG, y de una función llamada NIEGA_VARDEP.

Se procede de la siguiente manera:

Si VARNEG ≠ λ entonces:

- IVAR = λ
- REP1 = Pop CUAN_LISTA
- Llamar a NIEGA_VARDEP con REP1 y un cero
- Poner la salida de NIEGA_VARDEP en CUAN_LISTA

3)- IMPLICACION: se usa para tratar la implicación. A la implicación se la procesa como a una disyunción, por la equivalencia: $p \rightarrow q \equiv \neg p \vee q$. Para obtener la negación de p, hace uso de la función NIEGA_VARDEP.

se procede de la siguiente manera:

REP1 = Pop CUAN_LISTA

REP2 = Pop CUAN_LISTA

IVAR = λ

Llamar a NIEGA_VARDEP con REP2 y un uno

Poner la concatenación de la salida de NIEGA_VARDEP con REP1 en CUAN_LISTA

Ahora presentaremos a la función NIEGA_VARDEP, encargada de obtener la negación de un vector VAR_DEP.

Recibe como entrada el vector de variables cuantificadas que se quiera negar (llamémosle LISTA1), una lista iniciada con la

palabra vacía (llamémosle LISTA2) y un número entero (llamémosle BAND). En LISTA1 habrá listas con la forma: (X_i Y).

INICIO

Mientras LISTA1 ≠ λ, hacer:

VARCU = λ

Si X_i ∈ VARNEG ∧ BAND = 0 entonces:

- ✕– Poner el primer elemento de LISTA1 en LISTA2
- ✕– Si X_i = Y entonces: - Poner X_i en IVAR
  - LISTA1 = cdr_LISTA1
- ✕– En otro caso: - LISTA1 = cdr_lista1

Si X_i ∈ VARNEG entonces:

- ✕– Si X_i = Y entonces: - Poner IVAR en VARCU
  - Poner X_i en VARCU
  - IVAR = λ
  - Poner VARCU en LISTA2
  - LISTA1 = cdr_LISTA1
- ✕– En otro caso: - Poner X_i en VARCU
  - Poner X_i en VARCU
  - Poner X_i en IVAR
  - Poner VARCU en LISTA2
  - LISTA1 = cdr_LISTA1

FIN

En este capítulo hemos presentado cómo opera el Analizador Sintáctico.
Vimos que durante el análisis sintáctico de la oración, se van efectuando operaciones para tratar conectivos y cuantificadores.
Todos los procesos que se van ejecutando a medida que avanza el análisis, permiten finalmente obtener una representación clausular del enunciado.

Además, para cada enunciado se construirá un vector de variables cuantificadas.

En el siguiente capítulo, presentaremos algunas conclusiones acerca de este trabajo.

7- CONCLUSIONES

En este capítulo presentaremos algunas ventajas y desventajas del sistema, y un breve comentario acerca de futuros desarrollos del mismo.

El traductor presentado en este trabajo permite a cada usuario definir facilmente su propia gramática. Para su diseño no se consideró ningún tipo de discurso, facilitando así la generalización. El usuario sólo deberá reconocer las propiedades importantes y establecer las oraciones en castellano que las representen. Luego, siguiendo la forma normal de Backus definirá la gramática correspondiente a esas oraciones. La gramática definida se utilizará en la traducción de enunciados formulados en castellano a cláusulas de primer orden. Dicha traducción es bastante *rápida* pues durante el análisis sintáctico se construye la representación clausular.

El usuario puede definir una gramática tan amplia como lo desee, proporcionando así mucha generalidad al sistema.

Debido al tratamiento de conectivos y de cuantificadores implementado, se permite traducir cualquier proposición de primer orden por más compleja que sea. Cualquier oración (que se ajuste a la gramática) podrá ser traducida a la forma clausular. Las teorías de primer orden permiten la descripción precisa de importantes temas de matemáticas y de la vida real [5]. La característica principal de nuestro sistema es que es capaz de manejar toda la teoría de primer orden.

Una restricción importante del sistema es el espacio ocupado para la representación (listas de listas) de las cláusulas. El crecimiento del espacio ocupado es prácticamente exponencial. Cada vez que se agrega un conectivo a una proposición, el número de cláusulas formadas se incrementa notablemente, considerando la representación que hemos utilizado.

Por otro lado, al ser el sistema tan general, las máquinas de inferencia no pueden operar con todas las cláusulas obtenidas a partir del traductor. En el caso de Prolog, se hace necesario hacer una selección de aquellas que corresponden al tipo llamado *cláusulas de Horn*, y que no están cuantificadas existencialmente. En nuestro sistema sólo se traduce. No se reconocen a las cláusulas de Horn, y por lo tanto tampoco se determina qué cláusulas de Horn tienen información completa. Estas cláusulas son aquellas cuya información nos aseguran que el proceso de inferencia no caerá en un ciclo infinito.

Una futura continuación de este trabajo consiste en implementar un procedimiento que reduzca el espacio de almacenamiento de cláusulas.

Además, se pretende desarrollar un procedimiento de deducción para cláusulas que no son de Horn, y para proposiciones con estructuras complejas de cuantificadores.

REFERENCIAS

[1] Cairó, Osvaldo. " UN SISTEMA BASADO EN CONOCIMIENTOS ". Tesis de Maestría, Sección de Computación, Dpto. Ing. Eléctrica, CINVESTAV-IPN. México, 1987.

[2] Denning,P.; Dennis,J.; Qualitz,J. "MACHINES, LANGUAGES, AND COMPUTATION".Prentice-Hall. New Jersey, 1978.

[3] Korfhage, Robert. "LOGICA Y ALGORITMOS", Ed. Limusa, México, 1970.

[4] Kowalski, Robert. "LOGIC FOR PROBLEM SOLVING", Imperial College of Science and Technology University of London. New York, 1979.

[5] Mendelson, Elliot. "INTRODUCTION TO MATHEMATICAL LOGIC". Van Nostrand Company. Princeton, 1964.

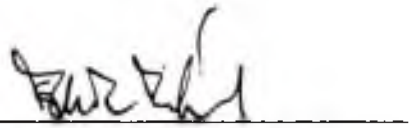
[6] Montes de Oca, Francisco. "LOGICA", Ed.Porrua, México, 1984.

[7] Morales.Guillermo; Guardati, Silvia." TRESLOG: Translating Spanish into Clausal Form Propositions ". (Por aparecer). Proceedings of the EXPERT SYSTEMS 87 Conference. British Computer Society. Brighton, Inglaterra. 1987.

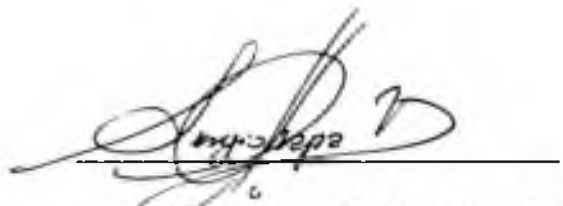
[8] Moreno,A. "ENTIENDA LA GRAMATICA MODERNA", Edición Larousse. México,1985.

[9] Wirth,N. "ALGORITMOS + ESTRUCTURAS DE DATOS = PROGRAMAS", Ediciones del Castillo, Madrid.

El jurado designado por la Sección de Computación del Departamento de Ingeniería Eléctrica del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó esta tesis el 22 de Setiembre de 1987.

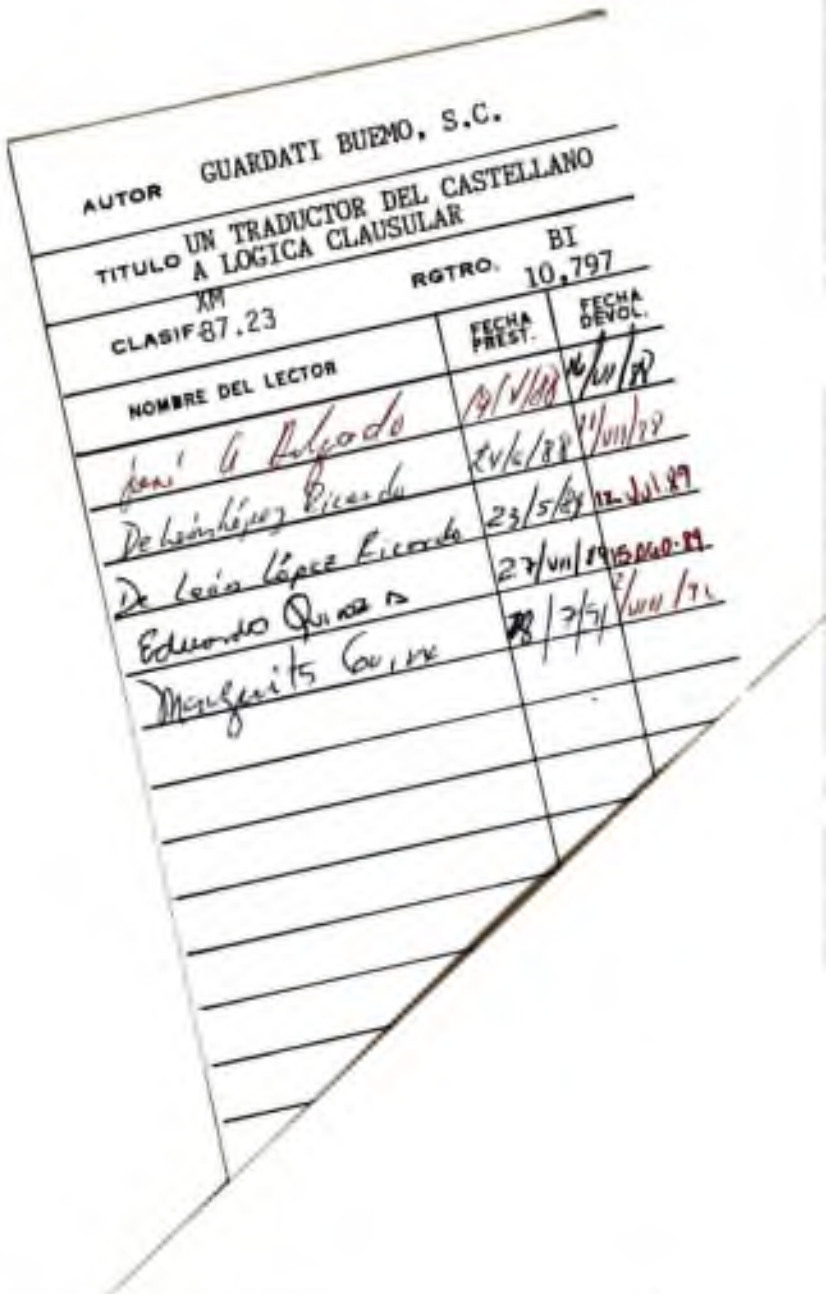


Dr. Zdenek Zdrahal



M. en C. Sergio Chapa Vergara

Dr. Guillermo Morales Luna



AUTOR GUARDATI BUEMO, S.C.

TITULO UN TRADUCTOR DEL CASTELLANO A LOGICA CLAUSULAR

XM

CLASIF 87.23 RGTRO. BI 10,797

NOMBRE DEL LECTOR	FECHA PREST.	FECHA DEVOL.
Jesús G Rolando	19/V/88	26/VII/88
De León López Ricardo	24/6/88	24/VIII/88
De León López Ricardo	23/5/89	12. VII 89
Eduardo Quiroz D.	27/VII/89	15 AGO. 89
Margarita Gomez	28/7/91	2/VIII/91