35

24

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

TRADUCCION AUTOMATICA A FORMA CLAUSULAR

T E S I S

QUE PARA OBTENER EL TITULO DE

M A T E M A T I C O

P R E S E N T A

SONIA SABRE MARTINEZ ADAME

MEXICO, D. F. 1988

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE.

INTRODUCCION.

La lógica de primer orden es un lenguaje con un gran poder de expresión, por lo cual ha sido objeto de estudio desde su creación, por Frege, hasta nuestros días. En la lógica de primer orden, una fórmula expresa las relaciones que existen entre una o varias proposiciones. Sin embargo, la relación entre dos o más proposiciones puede ser escrita de varias maneras, dando lugar con esto, a varias fórmulas equivalentes. Algunas de ellas, en particular la forma clausular de una fórmula, son más adecuadas para ser manipuladas por una computadora.

Los trabajos de Skolem, alrededor de 1928, y de Herbrand, alrededor de 1930, dieron lugar a un algoritmo mediante el cual cualquier fórmula de primer orden puede ser traducida a forma clausular de modo que si la primera es verdadera en un dominio A, la segunda es verdadera en un dominio B e inversamente. Este algoritmo es tedioso y se puede dificultar al aumentar la complejidad de la fórmula. Además la forma clausular obtenida puede no ser óptima. Por estas razones se han hecho programas que llevan a cabo el proceso automáticamente y también se han creado variantes del algoritmo que generan mejores formas clausulares. En particular, Clocksin y Mellish presentan en [3], una parte de un programa en PROLOG que lleva a cabo la traducción a forma clausular.

El objetivo principal de esta tesis es hacer un estudio de las variantes del algoritmo de traducción planteado por Skolem y Herbrand que nos permita analizar los casos en los que una variante es mejor que otra. Para tal efecto, tomando la idea de Clocksin y Mellish, utilizamos el lenguaje PROLOG para la programación de las distintas variantes contando así con una herramienta computarizada para llevar a cabo el análisis de los distintos algoritmos. Por otro lado, a partir de una forma clausular es bastante sencillo, usando la negación extralógica de PROLOG, obtener el conjunto de cláusulas PROLOG correspondiente. Entonces, además de traducir una fórmula a forma clausular, nos pareció interesante automatizar el proceso para llevar cada cláusula de la forma clausular a una cláusula PROLOG.

En el capitulo I, haremos una presentación de los conceptos de lógica de primer orden que son necesarios para un mejor entendimiento del proceso de traducción y definiremos la forma clásica del algoritmo de traducción. A continuación, en el

capítulo II, describiremos y analizaremos dos variantes del proceso de traducción presentado en el capítulo I. La primera variante es el producto de haber conjuntado una serie de ideas que dieron lugar a un proceso con el cual las fórmulas resultantes son mejores que con el algoritmo clásico; la segunda variante se debe a Andrews [1] y también optimiza las formas clausulares . En el capítulo III explicaremos algunos de los predicados en PROLOG que se hicieron para los dos algoritmos presentados en el capítulo II. En el capítulo IV, la intención es describir la estructura global de los programas que traducen a formas clausulares, así como del proceso que traduce una forma clausular al conjunto de cláusulas PROLOG correspondiente. Finalmente, el capítulo V, proporcionará al usuario una guía de utilización del programa y mostrará algunos ejemplos.

En los anexos 1, 2 y 3 expondremos una versión conjunta del programa PROLOG descrito en los capítulos III y IV. En el anexo 1, presentaremos la estructura global del sistema de traducción; en el anexo 2, ofrecemos la codificación de algunas de las utilerías de apoyo de la estructura global y de algunos predicados cuya definición , por tediosa, omitimos en el capítulo III. El anexo 3 contiene la definición de los predicados correspondientes al preprocesamiento de la fórmula a traducir y a la impresión de las formas clausulares. Por último, el anexo 4 contendrá una lista de las equivalencias lógicas que se utilizaron en el proceso de traducción.

CAPITULO I.

Método Clásico de Traducción a Forma Clausular.

En este capítulo explicaremos el proceso que transforma una fórmula de lógica de primer orden a forma clausular. Este proceso se apoya en un teorema de la lógica de primer orden que enunciaremos más adelante. En una primera parte del capítulo definiremos el alfabeto, las fórmulas y el lenguaje de la lógica de primer orden. En una segunda parte daremos la definición formal de varios conceptos que son necesarios para poder explicar el proceso mediante el cual llevamos una fórmula de primer orden a forma clausular. En una tercera y última parte explicaremos cada paso del proceso de transformación y después daremos la fundamentación lógica de cada uno de ellos.

I.1 El alfabeto y las fórmulas del lenguaje de primer orden.

Los símbolos del alfabeto son los siguientes : variables, conectivos, cuantificadores, símbolos de puntuación, constantes, símbolos funcionales y predicativos.

. **Las variables** se utilizan para denotar objetos indeterminados y generalmente se denotarán con letras minúsculas como :

u, u1, u2,...
v, v1, v2,...
w, w1, w2,...
x, x1, x2,...
y, y1, y2,...
z, z1, z2,...

. **Los conectivos** son los símbolos que estructuran una fórmula :

~ para la negación.
V para la disyunción.
& para la conjunción.
→ para la implicación.
↔ para la equivalencia.

. **Los cuantificadores** se denotarán de la manera siguiente :

∀x para la cuantificación universal (para toda x).
∃x para la cuantificación existencial (existe x).

. **Los símbolos de puntuación** o auxiliares son los paréntesis y la coma :

( ) ,

. **Las constantes** se utilizan para denotar objetos determinados y se escriben como:

c1,c2,c3,c4,...,cn,...

- Los símbolos funcionales, que permiten definir objetos del lenguaje, serán denotados generalmente con símbolos del tipo :

$$f,\ f_1,\ f_2,\ldots$$
$$g,\ g_1,\ g_2,\ldots$$
$$h,\ h_1,\ h_2,\ldots$$

y tendrán asociado implícitamente un grado o número de argumentos mayor que cero.

- Los símbolos predicativos, que definen las relaciones entre objetos, se denotarán normalmente con símbolos del tipo :

$$P,\ P_1,\ P_2,\ldots$$
$$Q,\ Q_1,\ Q_2,\ldots$$
$$R,\ R_1,\ R_2,\ldots$$
$$S,\ S_1,\ S_2,\ldots$$

que nuevamente tendrán asociado un grado mayor que cero.

Términos.
Los símbolos funcionales aplicados a las variables y a los símbolos constantes determinan los términos del lenguaje y se definen recursivamente como sigue :

i. Una constante es un término.
ii. Una variable es un término.
iii. Si f_i es un símbolo funcional de grado n y $t_1,\ldots,t_n$ son n términos entonces

$$f_i(t_1,t_2,\ldots,t_n) \text{ es un término.}$$

iv. Los únicos términos del lenguaje son los definidos en i., ii. e iii.

Veamos un ejemplo de término; sea x una variable, c una constante, f un símbolo funcional de grado 1 y g un símbolo funcional de grado 2, entonces :

. x y c son términos por los incisos i. e ii. de la definición anterior.
. f(c) es término por los incisos iii. e i.
. g(f(c),x) también es término por los incisos iii., ii. e i.
. fc(no es término por los incisos iv. e i., ii., iii.

Fórmulas Atómicas.
Los símbolos predicativos aplicados a los términos generan las fórmulas atómicas y se definen como expresiones de la forma :

$$R_i(t_1,t_2,\ldots,t_n)$$

donde los ti's son términos y la Ri es un símbolo predicativo de grado n.

No todas las cadenas de símbolos del lenguaje son fórmulas de la lógica de primer orden, a continuación definiremos formalmente que cadenas de símbolos son fórmulas.

Fórmula.
Una fórmula de la lógica de primer orden se define recursivamente como sigue :

i. Las fórmulas atómicas son fórmulas del lenguaje.
ii. Si F es una fórmula entonces (~F) es una fórmula.
iii. Si F y G son fórmulas, entonces :

(F V G)
(F & G)
(F → G)
(F ↔ G)

también lo son.

iv. Si F es una fórmula y x es una variable, entonces :

(∃x F) (existe x tal que F)
(∀x F) (para toda x sea F)

son fórmulas . Decimos que x es la variable asociada al cuantificador existencial o universal según sea el caso.

v. Una cadena de símbolos es una fórmula del lenguaje sólo si cumple con alguna de las definiciones de i. a iv.

Lenguaje de primer orden.
El lenguaje de primer orden L, consistirá de todas las fórmulas construidas a partir de los símbolos del alfabeto.

Un ejemplo de fórmula de la lógica de primer orden es el siguiente: Si f es un símbolo funcional de grado 1, R es un símbolo predicativo de grado 2, x y y son variables y c es una constante, entonces

$$\forall x(\exists y((R(x,f(y)) \& R(c,f(x))) \rightarrow R(x,f(y)))) \qquad (I.1)$$

es una fórmula que se lee como : para toda x, existe y tal que R(x,f(y)) y R(c,f(x)) implican a R(x,f(y)); la fórmula I.1 es de primer orden ya que:

. x, y, c, f(x) y f(y) son términos.

. R(x,f(y)) y R(c,f(x)) son fórmulas atómicas por el inciso i. de la definición de fórmula.

. R(x,f(y)) & R(c,f(x)) es una fórmula por iii.

. ((R(x,f(y)) & R(c,f(x))) → R(x,f(y)) es una fórmula por el inciso iii.

. $\exists y((R(x,f(y)) \& R(c,f(x))) \rightarrow R(x,f(y)))$ es fórmula por el inciso iv.

. $\forall x(\exists y((R(x,f(y)) \& R(c,f(x))) \rightarrow R(x,f(y))))$ es fórmula por el inciso iv.

Para que una fórmula sea más fácil de leer y escribir, abusaremos de la notación y omitiremos ciertos paréntesis. Para esto adoptaremos la siguiente convención de precedencia para los conectivos lógicos : $\sim$, $\forall$, &, $\rightarrow$ y $\leftrightarrow$. Esto quiere decir que primero se aplica el $\sim$ a la fórmula más pequeña que lo siga; después, el & y el $\forall$ relacionan las fórmulas más pequeñas que los rodean y por último, el $\rightarrow$ y el $\leftrightarrow$, nuevamente relacionan las fórmulas más pequeñas que los rodeen. Por lo que la fórmula I.1 se podrá escribir de manera abreviada como :

$$\forall x \exists y(R(x,f(y)) \& R(c,f(x)) \rightarrow R(x,f(y)))$$

I.2 Conceptos fundamentales.

Ahora daremos las definiciones de ciertos conceptos que serán necesarios para poder explicar y fundamentar el proceso de transformación de una fórmula de primer orden a su forma clausular, forma que definiremos más adelante. En la primera parte hablaremos de los conceptos asociados a los cuantificadores y fórmulas; en la segunda de los conceptos relativos a modelo, validez y equivalencia de fórmulas.

I.2.1 Cuantificadores y fórmulas.

Dados los cuantificadores universal y existencial ($\forall$ y $\exists$), definiremos algunas nociones asociadas a ellos como son: la de alcance, la de variable acotada y libre, la de cerradura, etc.

Alcance de un Cuantificador.
Se dice que la fórmula F esta bajo el alcance de un cuantificador, si F ocurre en una fórmula del tipo :

$$\forall x\ F \quad \text{ó} \quad \exists x\ F.$$

Por ejemplo, en la fórmula :

$$\forall y \forall x(R(x,w) \& R(x,y)) \vee R(y,y) \vee \exists z P(z) \qquad (I.2)$$

la fórmula [R(x,w) & R(x,y)] esta bajo el alcance de los cuantificadores $\forall x$ y $\forall y$, la fórmula P(z) esta bajo el alcance del cuantificador existencial $\exists z$ y la fórmula R(y,y) esta libre del alcance de cuantificadores.

Ocurrencia Acotada.
La ocurrencia de una variable x está acotada en una fórmula, si y sólo si, x es la variable asociada a un cuantificador del tipo $\forall x$ o $\exists x$, o bien x está bajo el alcance de un cuantificador $\forall x$ o $\exists x$. Por ejemplo, en la fórmula I.2 : todas las ocurrencias de x están acotadas universalmente, la primera ocurrencia de y en R(x,y) está acotada universalmente y la ocurrencia de z, en p(z), está acotada existencialmente.

Ocurrencia Libre.
La ocurrencia de una variable x está libre si no está acotada. Por ejemplo en I.2, la ocurrencia de w en R(x,w) está libre y las ocurrencias de y en R(y,y) están libres.

Variables Libres y Acotadas en una fórmula.
i. Una variable es libre en una fórmula F, si y sólo si tiene ocurrencias libres en F.
ii. Una variable es acotada en una fórmula si y sólo si tiene ocurrencias acotadas universales o existenciales dentro de la misma.

Nótese que una misma variable puede tener ocurrencias libres y acotadas dentro de una fórmula. Por ejemplo, en la fórmula I.2, la variable w es libre y las variables x y z son acotadas mientras que la variable y es acotada y libre ya que tiene una ocurrencia acotada en R(x,y) y dos ocurrencias libres en R(y,y).

Cuantificador Vacuo.
Un cuantificador $\forall x$ F o $\exists x$ F es vacuo si la variable x del cuantificador no ocurre dentro de la fórmula F acotada por el cuantificador. Por ejemplo, en la fórmula :

$$\forall x \exists y (R(z,y) \rightarrow R(y,z))$$

el cuantificador $\forall x$ es vacuo, ya que la variable x, no ocurre en la fórmula $R(z,y) \rightarrow R(y,z)$.

Fórmula Abierta.
Una fórmula es abierta si no contiene ningún cuantificador. Es decir si todas las ocurrencias de todas las variables de la fórmula son libres. Por ejemplo, la fórmula

$$R(x,y) \rightarrow S(y,x)$$

es abierta.

Fórmula Cerrada.
Una fórmula es cerrada si todas las ocurrencias de sus variables son acotadas dentro de la fórmula. Por ejemplo :

$$\forall x \forall y (R(x, f(x)) \vee R(y, f(y)))$$

es una fórmula cerrada.

Cerradura Universal.
Si F es una fórmula con variables libres :

$$x_1, x_2, \ldots, x_n$$

entonces, la cerradura universal de F se define como la fórmula :

$$\forall x_1 \forall x_2 \ldots \forall x_n \, F$$

La cerradura universal de una fórmula F se denotará como : $\forall F$. Por ejemplo, la cerradura universal de la fórmula I.2 es la fórmula

$$\forall w \forall y \forall x ((R(x, w) \,\&\, R(x, y)) \vee R(y, y) \vee \exists z P(z))$$

ya que las variables w y y que aparecían libres en R(x,w) y en R(y,y) fueron acotadas por $\forall w$ y $\forall y$.

Cláusula.
A toda fórmula atómica o la negación de una fórmula atómica se le denomina literal. Una cláusula se definirá entonces como una disyunción de literales. Formalmente, una cláusula es una fórmula del tipo

$$L_1 \vee L_2 \vee \ldots \vee L_n$$

en donde cada L_i, $i = 1, \ldots, n$, es una literal.

Por ejemplo, la fórmula $R(x,y) \vee \neg P(y,z) \vee Q(x,z)$ es una cláusula con literales $L_1 = R(x,y)$, $L_2 = \neg P(y,z)$ y $L_3 = Q(x,z)$.

Forma Normal Conjuntiva (FNC).
Esta forma se conoce como la forma canónica de una fórmula y se define como:

$$(L_{11} \vee \ldots \vee L_{1n_1}) \,\&\, (L_{21} \vee \ldots \vee L_{2n_2}) \,\&\, \ldots\ldots \,\&\, (L_{k1} \vee \ldots \vee L_{kn_k})$$

en donde cada L_{ij} es una literal.

Nótese que una forma normal conjuntiva es una conjunción de cláusulas. Entonces, si $C_1, C_2, \ldots, C_k$ son cláusulas, la forma normal conjuntiva puede escribirse como :

$$C_1 \,\&\, C_2 \,\&\, \ldots\ldots \,\&\, C_k$$

Por ejemplo, $(R(x,y) \vee Q(x,z)) \,\&\, (P(x) \vee \neg S(y))$ es una forma normal conjuntiva, en donde $C_1 = (R(x,y) \vee Q(x,z))$ y

$C_2 = (P(x) \vee \sim S(y))$ son las cláusulas que la conforman.

Forma Clausular.
La forma normal conjuntiva de una fórmula puede verse también como el conjunto de cláusulas que intervienen en la conjunción y en este caso se denominará forma clausular. Por ejemplo, el conjunto $\{ R(x,y) \vee Q(x,z) , P(x) \vee \sim S(y) \}$ es la forma clausular asociada a la forma normal conjuntiva $(R(x,y) \vee Q(x,z)) \& (P(x) \vee \sim S(y))$.

I.2.2 Dominios de Interpretación.

El concepto de verdad de una fórmula no es absoluto, es relativo a su significado; así para poder decidir si una fórmula es verdadera o falsa, es necesario dar un significado a los símbolos que aparecen en ella. Los cuantificadores y los conectivos tienen un significado fijo; sin embargo el significado que les asociamos a las constantes, a los símbolos funcionales y a los símbolos predicativos de una fórmula puede variar. Una interpretación consiste entonces en : a) un dominio D, sobre el cual variarán las variables y sobre el cual tomarán significado los símbolos de la fórmula, b) una asociación que asigna un individuo del dominio D a cada constante de la fórmula, c) una asociación que defina una función de D en D para cada símbolo funcional de la fórmula y d) una asociación que asigne una relación en D a cada símbolo predicativo de la fórmula.

Entonces para cada fórmula, una interpretación especifica dará a cada símbolo de la fórmula, un significado particular. Por lo que formalmente una interpretación se definirá como sigue.

Interpretación.
Una interpretación I, de un subconjunto S del lenguaje de primer orden consiste en :

i. Un conjunto no vacío D, llamado el dominio de la interpretación.

ii. La asociación de un elemento en D, a cada constante de S.

iii. La asociación de una función de D^n en D, a cada símbolo funcional de grado n en S.

iv. La asociación de una relación sobre D^n, a cada símbolo predicativo de grado n en S.

Por ejemplo, sea I una interpretación del conjunto S definido como

$$S = \{f, g, c_1, c_2, \ldots\}$$

tomemos como dominio de I al conjunto de los números naturales ($D=\mathbb{N}$). La asociación definida por I, a los símbolos f, g y c_i será como sigue:

$I(f) = +$ f con la operación suma.

$I(g) = *$ g con la operación multiplicación.

$I(c_i) = i$ c_i con el número natural i.

Con esta interpretación, estamos asociando los símbolos funcionales del conjunto S con operaciones en los números naturales y las constantes con los números mismos.

Si al conjunto S, con una interpretación I, le agregamos nuevas constantes, nuevos símbolos predicativos o símbolos funcionales, tendremos una nueva interpretación I', que denominaremos una expansión de I. Por ejemplo, si al conjunto del ejemplo anterior le agregamos el símbolo predicativo R de grado 2, obtendremos una expansión I' de I tal que :

$I'(f) = I(f)$

$I'(g) = I(g)$

$I'(c_i) = I(c_i)$

$I'(R) =$ '>' la relación "mayor que" en los números naturales.

Con una interpretación le estamos asignando entonces, un significado específico a las constantes, símbolos funcionales y símbolos predicativos del lenguaje. Faltaría darle un significado a las variables y a los términos; esto lo hacemos mediante las siguientes definiciones.

Asignación de Variables.
Sea I una interpretación del lenguaje L de primer orden. Una asignación de variables con respecto a I es una función A, que asigna a cada variable de L un elemento del dominio de I.

Asignación de Términos.
Sea I una interpretación con dominio D de un subconjunto S del lenguaje de primer orden y sea A una asignación de variables. La asignación de términos de S con respecto a I y a A se define como:

i. Cada variable tiene una asignación de acuerdo a A.

ii. Cada constante de S tiene una asignación de acuerdo a I.

iii. Si $t_1', \ldots, t_n'$ son los elementos en D asignados a $t_1, \ldots, t_n$ bajo I y A, y si f' es la asignación de f de acuerdo a I, entonces

$$f'(t_1', \ldots, t_n')$$

es el elemento en D asignado a f(t1,...,tn) con respecto a I y a A.

En este momento, dada una interpretación, una asignación de variables y una fórmula F, podemos dar un significado a todos los símbolos que aparecen en F. Ahora nos interesa saber si con ese significado la fórmula es verdadera o falsa por lo que a continuación definiremos el concepto de valor de verdad de una fórmula bajo una interpretación.

Concepto de Verdad.
Sea I una interpretación, con dominio D, de un subconjunto S del lenguaje de primer orden; y sea A una asignación de variables. Entonces, a una fórmula de S le podemos asignar un valor de verdad, verdadero o falso, con respecto a I y a A de la siguiente manera:

i. Si la fórmula R(t1,...,tn), es atómica entonces el valor de verdad se obtiene calculando el valor R'(t1',...,tn') donde R' es la asignación I(R) con respecto a I y t1',...,tn' son los elementos asignados a los términos t1,...,tn con respecto a I y a A.

ii. Si la fórmula tiene la forma ~F, F & G, F V G, F → G, F ↔ G entonces el valor de verdad estará dado por el siguiente cuadro :

F	G	~F	F & G	F V G	F → G	F ↔ G
v	v	f	v	v	v	v
v	f	f	f	v	f	f
f	v	v	f	v	v	f
f	f	v	f	f	v	v

en donde v es la abreviación de verdadero y f de falso.

iii. Si la fórmula tiene la forma ∃x F, entonces el valor de verdad de la fórmula es verdadero si existe una d∈D, tal que F toma el valor verdadero con respecto a I y a A(x/d). La asignación de variables A(x/d) es igual a A con la excepción de que a x le asigna el valor d. De lo contrario el valor de verdad es falso.

iv. Si la fórmula tiene la forma ∀x F, entonces el valor de verdad de la fórmula es verdadero si para toda d∈D, tenemos que F toma el valor verdadero con respecto a I y a A(x/d). De lo contrario el valor de verdad es falso.

De los últimos dos incisos de la definición podemos notar que el valor de verdad de una fórmula cerrada no depende de la asignación de variables, por lo que de aquí en adelante

hablaremos del valor de verdad de una fórmula cerrada solamente con respecto a una interpretación, sin tomar en cuenta la asignación de variables.

Nótese que una interpretación puede hacer verdadera a una fórmula particular. Si este es el caso, la interpretación se llama modelo de la fórmula. También una interpretación puede hacer verdadero a un conjunto de fórmulas, en cuyo caso se denomina modelo de todo el conjunto.

Modelo.
Sea F una fórmula de lógica de primer orden y C un conjunto de fórmulas de primer orden. Entonces

i. F tiene un modelo si existe una interpretación I (independiente de cualquier asignación) que la hace verdadera.
ii. C tiene modelo si existe una interpretación I que hace verdadera a cada fórmula de C.

Se dice que un conjunto de fórmulas es válido, si toda interpretación bajo la cual se pueda interpretar el conjunto, hace verdadera a cada fórmula del conjunto. A continuación definimos formalmente el concepto de validez.

Validez.
Sea F una fórmula cerrada del lenguaje L de primer orden y C un conjunto de fórmulas cerradas. Decimos que :

i. C es válido si toda interpretación de C es modelo de C.
ii. F es válida si toda interpretación de F es modelo de F. En este caso la fórmula F es válida independientemente de la interpretación y se denota como : $\models F$.

Para una fórmula cualquiera diremos que es válida si su cerradura universal lo es. Por último definiremos cuando es que dos fórmulas son equivalentes.

Equivalencia entre fórmulas.
Si F y G son fórmulas, decimos que F y G son lógicamente equivalentes si la fórmula $F \leftrightarrow G$ es válida y escribiremos :

$$\models F \leftrightarrow G \qquad \text{o bien} \qquad F \equiv G$$

I.3 Proceso de Transformación.

En este momento tenemos ya las definiciones necesarias para poder explicar el proceso mediante el cual una fórmula F de lógica

de primer orden es llevada a su forma clausular $\langle C_1, C_2, \ldots, C_m \rangle$, en donde cada C_i es una cláusula. Nótese que para poder obtener la forma clausular de F, es necesario llevar primero la fórmula F a una fórmula F' del tipo :

$$F' = \forall x_1 \ldots \forall x_k \; G(x_1, \ldots, x_n) \qquad (I.3)$$

en donde $n \geq k$, $x_1, \ldots, x_n$ son todas las variables que ocurren en F y $G(x_1, \ldots x_n)$ es una forma normal conjuntiva del tipo $C_1 \& \ldots \& C_m$. Es importante fijarse en que F' no contiene cuantificadores existenciales, ni conectivos del tipo $\rightarrow$ o $\leftrightarrow$ y además las variables $x_{k+1}, \ldots, x_n$, si las hay, son libres.

A continuación explicaremos de manera muy general los pasos necesarios del proceso de transformación de una fórmula a su forma clausular.

1. Paso a Forma Normal Prenex.
   Para poder transformar una fórmula cualquiera F a una forma del tipo I.3 es necesario que todos los cuantificadores, universales y existenciales, que se encuentren dentro de la fórmula F salgan hasta el principio de la misma, obteniendo así, una fórmula que llamaremos forma normal prenex. La forma normal prenex será equivalente a la fórmula F y tendrá la forma QF_1, en donde Q es una secuencia de cuantificadores y F_1, una fórmula abierta. Al proceso de sacar los cuantificadores de la fórmula hasta el principio de ella se le llama prenexar la fórmula.

2. Paso a Forma Normal Conjuntiva.
   El siguiente paso del proceso es transformar la fórmula abierta F_1 a su forma normal conjuntiva, obteniendo una fórmula F_2 en la cual los conectivos $\rightarrow$ y $\leftrightarrow$ han sido eliminados y las negaciones sólo afectan a fórmulas atómicas. Después, utilizando un lema de sustitución de la lógica de primer orden, sustituiremos F_1 por su equivalente F_2 en QF_1 obteniendo QF_2.

3. Paso a Forma Normal de Skolem.
   Ahora debemos eliminar los cuantificadores existenciales de la secuencia de cuantificadores Q que ocurren en QF_2, de la siguiente manera : cada variable asociada a un cuantificador existencial será sustituida por una función que haga explícita la existencia de la variable. A estas funciones las llamaremos funciones de skolem. Después simplemente eliminaremos los cuantificadores existenciales. La fórmula que obtenemos en este paso es una fórmula del tipo Q_1F_3, donde Q_1 es una secuencia de cuantificadores universales y F_3 es el resultado de sustituir en F_2 las variables que estaban acotadas existencialmente por las funciones de skolem correspondientes.

4. Paso a Forma Clausular.
   En este momento ya tenemos la forma normal de skolem, Q_1F_3, del tipo :

$$\forall x_1 \ldots \forall x_k \, (C_1(x_1,\ldots,x_n) \;\&\; \ldots \;\&\; C_m(x_1,\ldots,x_n)) \qquad (I.4)$$

en donde $n \geq k$ y cada $C_i(x_1,\ldots,x_n)$ es una cláusula. Omitiendo los cuantificadores universales en la fórmula I.4, obtenemos la fórmula $C_1(x_1,\ldots,x_n) \;\&\; \ldots \;\&\; C_m(x_1,\ldots,x_n)$ y de ella la forma clausular

$$\{\, C_1(x_1,\ldots,x_n), \ldots, C_m(x_1,\ldots,x_n) \,\}$$

Durante el proceso anterior, fue necesario sustituir varias veces la ocurrencia de una fórmula G por otra fórmula equivalente G' en una fórmula F. Al hacer esta sustitución obtuvimos una fórmula F' equivalente a F. El primer lema que enunciaremos justifica formalmente esta sustitución de fórmulas.

Lema 1. Sustitución de fórmulas.
Sean F, F', G, G' fórmulas. Si $\models G \leftrightarrow G'$ y F' es el resultado de sustituir cada ocurrencia de G por G' en F entonces $\models F \leftrightarrow F'$.

Ahora expondremos con detalle los cuatro pasos de transformación antes mencionados, enunciando los lemas y teoremas que los fundamentan. Las demostraciones de estos lemas y teoremas al igual que la del lema 1, se pueden encontrar en las referencias [6] y [7].

I.3.1 Paso a Forma Normal Prenex.

Primero definiremos formalmente lo que es una fórmula prenexada.

Forma Normal Prenex.
Una fórmula en forma normal prenex es una fórmula QF en donde F es una fórmula abierta y Q es una secuencia :

$$Q_0 x_0 \ \ldots \ Q_{n-1} x_{n-1}$$

con $Q_i = \forall$ o $\exists$ y x_i una variable. Decimos que Q es el prefijo de QF y F la matriz. Por ejemplo, la fórmula :

$$\forall x \exists y \forall z \, (R(x,y) \;\&\; \sim S(z,x))$$

es una fórmula en forma normal prenex con prefijo : $\forall x \exists y \forall z$ y matriz : $R(x,y) \;\&\; \sim S(z,x)$.

Al prenexar una fórmula F de lógica de primer orden, i.e. al sacar los cuantificadores hasta el principio de la misma, obtenemos una nueva fórmula F' que es lógicamente equivalente a F. Esto queda enunciado en el siguiente teorema.

Teorema 1.
Cualquier fórmula tiene una fórmula lógicamente equivalente en forma normal prenex y con exactamente las mismas variables libres.

Supongamos que F es una fórmula de primer orden. Para pasar F a forma normal prenex, necesitamos sacar hasta el principio de F todos los cuantificadores. Es más sencillo sacar los cuantificadores en una fórmula que sólo contiene los conectivos & y V. Por lo tanto, antes de prenexar una fórmula, es conveniente eliminar de ella los conectivos de la forma → y ↔, y reducir las negaciones que aparezcan afectando a cuantificadores.

Con el siguiente lema enunciamos las equivalencias lógicas que utilizaremos para que una fórmula cumpla con las características necesarias para poder ser prenexada y las equivalencias lógicas que utilizaremos para poder prenexarla.

Lema 2.
Si P, Q y F son fórmulas de primer orden y x una variable, entonces

i. $\models P \to Q \leftrightarrow \neg P \vee Q$
ii. $\models P \leftrightarrow Q \leftrightarrow (\neg P \wedge \neg Q) \vee (P \wedge Q)$
iii. $\models \neg\exists x\, F \leftrightarrow \forall x\, \neg F$
iv. $\models \neg\forall x\, F \leftrightarrow \exists x\, \neg F$

v. Si x no es libre en F entonces

$\models (F \wedge \exists x\, G) \leftrightarrow \exists x(F \wedge G)$.
$\models (F \wedge \forall x\, G) \leftrightarrow \forall x(F \wedge G)$.

vi. Si x no es libre en F, entonces

$\models (F \vee \exists x\, G) \leftrightarrow \exists x(F \vee G)$.
$\models (F \vee \forall x\, G) \leftrightarrow \forall x(F \vee G)$.

vii. Si x no ocurre en F, entonces
$\models \forall x\, F \leftrightarrow F$
$\models \exists x\, F \leftrightarrow F$

donde que x no ocurra libre en F significa que x no ocurre en F o bien que x ocurre acotada en F. La prueba es directa de las definiciones.

El lema 1 y los dos primeros incisos del lema anterior nos permiten eliminar los conectivos → y ↔, ya que:

- Si tenemos la fórmula P → Q, ésta puede ser escrita como ¬PVQ, debido a que ambas fórmulas son lógicamente equivalentes.
- Si tenemos la fórmula P ↔ Q, podemos eliminar el ↔ escribiendo la fórmula como (¬P & ¬Q) V (P & Q), ya que ésta última es lógicamente equivalente a P ↔ Q.

Con los incisos iii) y iv) del lema anterior reducimos el alcance de la negación en subfórmulas que contengan la negación de cuantificadores, ya que la negación de un cuantificador existencial, ¬∃x F, es lógicamente equivalente a la cuantificación

universal de la negación de F; y la negación de un cuantificador universal, ~∀x F, es lógicamente equivalente a la cuantificación existencial de la negación de F. Las equivalencias v. y vi. serán utilizadas para sacar los cuantificadores al principio de la fórmula y su uso se explicará más adelante. La última equivalencia del Lema 2 permite eliminar los cuantificadores vacuos de una fórmula.

Por ejemplo, utilizando el inciso i) del lema 2, transformaremos la fórmula

$$\forall y(\exists x(P(x,y)) \rightarrow R(x))$$

en

$$\forall y(\sim\exists x(P(x,y)) \vee R(x)) \qquad (I.5)$$

Al haber eliminado los conectivos → y ↔, obtendremos una fórmula F1 equivalente a F. Pero ahora sucede que F1 puede contener cuantificadores negados (como en el ejemplo anterior). Para poder prenexar la fórmula necesitamos que los cuantificadores no esten negados. Para eliminar estas negaciones utilizaremos los incisos iii) y iv) del lema 2. Al eliminar los cuantificadores negados de la fórmula I.5 obtenemos :

$$\forall y(\forall x(\sim P(x,y)) \vee R(x)) \qquad (I.6)$$

Entonces para poder prenexar una fórmula F, es necesario :

1. Eliminar los conectivos → y ↔ de F, si es que existían, generando F1.
2. Eliminar los cuantificadores negados de F1 generando F2.

En seguida veremos que F2 todavía debe de cumplir con otra característica antes de poder ser prenexada, F2 debe ser una fórmula rectificada.

Fórmula Rectificada.
Una fórmula F es rectificada si y sólo si

- F no contiene ocurrencias acotadas y libres de una misma variable.
- F no contiene dos cuantificadores que acoten ocurrencias de una misma variable.
- F no contiene cuantificadores vacuos.

Analicemos los siguientes tres ejemplos para entender porque una fórmula debe ser rectificada antes de ser prenexada.

Caso a.
Una fórmula de lógica de primer orden puede contener ocurrencias libres y acotadas de una misma variable, por ejemplo , en la siguiente fórmula la variable x ocurre acotada en P(x) y libre en Q(x):

$$\forall x(P(x) \vee R(z,x)) \ \& \ Q(x) \qquad (I.7)$$

en realidad al hablar de la ocurrencia de x en P(x) y de la ocurrencia de x en Q(x) estamos hablando de variables diferentes. Al tratar de prenexar esta fórmula nos encontraremos con ciertos problemas. Supongamos, aunque esto no sea válido, que sacamos el cuantificador universal al principio de la fórmula y obtenemos:

$$\forall x(P(x) \vee R(z,x) \ \& \ Q(x))$$

el significado de esta última fórmula no es el mismo que el de la fórmula I.7, ya que la x para la cual se cumple P(x) debe ser la misma que la x para la cual se cumple Q(x), mientras que en la fórmula I.7, la x de P(x) y la x de Q(x) eran variables diferentes, realmente la x de Q(x) es variable mientras que la x de P(x) es parte de una afirmación universal.

Caso b.

Una fórmula también puede contener dos cuantificadores que acoten ocurrencias de una misma variable, por ejemplo :

$$\forall x(P(x) \vee \exists x(R(x,y))) \qquad (I.8)$$

en este caso al igual que en el anterior, la x de P(x) y la x de R(x,y) no son la misma variable. Al tratar de prenexarla, nuevamente tendremos problemas. Supongamos, aunque esto no sea válido, que sacamos el cuantificador existencial y obtenemos :

$$\forall x \exists x(P(x) \vee R(x,y))$$

el significado de esta última fórmula no es el mismo que el de la fórmula I.8, ya que nuevamente, la x para la cual se cumple P(x) y la x para la cual se cumple R(x,y) debe ser la misma variable, mientras que en la fórmula I.8 no lo eran.

Caso c.

También puede suceder que la fórmula contenga un cuantificador vacuo, por ejemplo :

$$\forall x(P(x) \ \& \ \forall z \ R(x,y)) \qquad (I.9)$$

El cuantificador $\forall z$ es vacuo ya que z no ocurre en R(x,y). La fórmula I.9 es lógicamente equivalente a $\forall x(P(x) \ \& \ R(x,y))$, ya que el cuantificador $\forall z$ se puede eliminar debido al Lema 2.vii.

Por lo tanto para poder sacar los cuantificadores en una fórmula F, ésta no debe contener cuantificadores que acoten ocurrencias de la misma variable, tampoco debe contener ocurrencias libres y acotadas de la misma variable y tampoco debe contener cuantificadores vacuos; i.e., para poder prenexar la fórmula F, necesitamos que F sea rectificada.

De los tres casos anteriores, el más sencillo de solucionar es el Caso c). Lo que haremos al encontrarnos con fórmulas como la I.9 será eliminar el cuantificador vacuo y obtendremos una fórmula equivalente a la original, esto queda justificado con el inciso vii del Lema 2 y con el Lema 1 de sustitución de fórmulas. Lo que haremos al encontrarnos con casos como el a) y el b) será lo siguiente :

> Sustituiremos la variable x asociada al cuantificador que queremos mover por otra variable x_i que no ocurra en la fórmula y sustituiremos todas las ocurrencias de x que esten bajo el alcance del cuantificador por x_i. Al hacer esto obtendremos una fórmula equivalente ya que las variables acotadas de una fórmula pueden cambiar sin alterar el significado de la misma. Esto se formaliza con el lema 3.

Lema 3. Sustitución de variables acotadas.
Sea x una variable que no ocurre en la fórmula F y sea F' el resultado de sustituir las ocurrencias acotadas de la variable y por la variable x. Entonces

$$\models F \leftrightarrow F'$$

Para ilustrar el proceso de rectificación de una fórmula, veamos que sucede con la fórmula I.6 del ejemplo :

$$\forall y((\forall x(\neg P(x,y)) \vee R(x))$$

El cuantificador $\forall x$ no puede salir y acotar a R(x), ya que x es libre en R(x) y el inciso v) del lema 2 pide que x no sea libre. Entonces sustituiremos la variable x acotada por el cuantificador universal, por otra variable w que no ocurra en F; al sustituir todas las ocurrencias acotadas de x en el alcance del cuantificador universal por w, obtendremos una fórmula lógicamente equivalente a la fórmula I.6 debido al lema 3, ésta será :

$$\forall y((\forall w \neg P(w,y)) \vee R(x))$$

En este momento el cuantificador universal $\forall w$ puede salir a la izquierda de la fórmula ya que la variable w no es libre en R(x). Al sacar el cuantificador obtendremos una nueva fórmula lógicamente equivalente a I.5 y I.6:

$$\forall y \forall w((\neg P(w,y)) \vee R(x)).$$

Como hemos visto, para que una fórmula F pueda ser transformada a forma normal prenex, ésta debe de cumplir con los siguientes requisitos :

i. F no debe contener los conectivos $\rightarrow$ y $\leftrightarrow$.
ii. F no debe contener cuantificadores negados.
iii. F debe ser rectificada.

Luego de esto, la fórmula se prenexa completando así el primer paso de la transformación.

I.3.2 Paso a Forma Normal Conjuntiva.

Lo que haremos ahora será llevar la matriz M de la fórmula prenexada QM, a su forma normal conjuntiva. Para esto utilizaremos las equivalencias lógicas del siguiente lema.

Lema 4.
Si P, Q y R son fórmulas de primer orden, entonces

i. $\models P \vee (Q \wedge R) \leftrightarrow (P \vee Q) \wedge (P \vee R)$
ii. $\models \sim\sim P \leftrightarrow P$
iii. $\models \sim(P \vee Q) \leftrightarrow \sim P \wedge \sim Q$
iv. $\models \sim(P \wedge Q) \leftrightarrow \sim P \vee \sim Q$

La prueba puede obtenerse directamente de las definiciones de verdad y validez. El inciso i) del lema anterior es la equivalencia lógica principal del proceso ya que es la que nos permite transformar una disyunción de conjunciones en una conjunción de disyunciones, es decir, en una forma normal conjuntiva. Las otras tres equivalencias del lema anterior serán utilizadas para reducir el alcance de las negaciones cuando éstas aparezcan aplicadas a fórmulas no atómicas; su utilización será explicada más adelante.

Como mencionamos al principio de esta tercera parte del capítulo, al transformar una fórmula abierta a su forma normal conjuntiva, obtenemos una fórmula equivalente a la primera. Este resultado puede enunciarse como el siguiente teorema.

Teorema 2.
Toda fórmula abierta tiene una forma normal conjuntiva equivalente.

La prueba de este teorema puede ser obtenida de manera inmediata a partir del Lema 4 y del Lema 1 de sustitución. Ahora bien, la matriz M de una forma normal prenex QM es una fórmula abierta y por lo tanto tiene una fórmula equivalente M' en forma normal conjuntiva. Si sustituimos M' por M en la forma normal prenex QM, obtendremos otra forma normal prenex QM' equivalente a QM, por el Lema 1.

En la práctica, para transformar una fórmula abierta a su forma normal conjuntiva utilizaremos la equivalencia i) del Lema 4. Para esto necesitamos que la negación no este aplicada a los conectivos & y V. Entonces antes de pasar una fórmula a su forma normal conjuntiva debemos reducir el alcance de la negación cuando ésta afecte a los conectivos & y V, para obtener fórmulas en las cuales la negación solamente este aplicada a fórmulas atómicas.

Los incisos ii), iii) y iv) del Lema 4 nos permiten reducir el alcance de la negación. Por ejemplo, en la fórmula

$$\sim(P(x) \vee Q(x,y)) \vee R(w,x)$$

la subfórmula [P(x) V Q(x,y)] esta negada, y por lo tanto no podemos pasar la fórmula a forma normal conjuntiva usando directamente el inciso i) del lema 4. Entonces, aplicamos primero el inciso iii) del Lema y obtenemos :

$$(\sim P(x) \wedge \sim Q(x,y)) \vee R(w,x).$$

En este momento las negaciones solamente están aplicadas a fórmulas atómicas y ya podemos utilizar el inciso i) del Lema 4 para pasar la fórmula a forma normal conjuntiva, obteniendo :

$$(\sim P(x) \vee R(w,x)) \wedge (\sim Q(x,y) \vee R(w,x)).$$

Los incisos ii) y iv) del Lema 4 son utilizados de manera análoga para reducir el alcance de la negación.

Una fórmula en la cual las negaciones han sido reducidas puede ser llevada a su forma normal conjuntiva sin problema. Por ejemplo dada la fórmula

$$S(x) \wedge (\underbrace{(S(y)}_{Q} \wedge \underbrace{\sim S(f(x,y))}_{R}) \vee \underbrace{(\sim S_4(x,z) \vee S(z))}_{P})$$

aplicando el inciso i) del lema 4 y el lema 1 a las fórmulas P, Q y R, obtenemos la fórmula :

$$S(x) \wedge (S(y) \vee \sim S_4(x,z) \vee S(z)) \wedge (\sim S(f(x,y)) \vee \sim S_4(x,z) \vee S(z)).$$

Al terminar el paso 2, tenemos una fórmula QM', en la cual, M' tiene una forma normal conjuntiva acotada por una secuencia de cuantificadores Q.

I.3.3 Paso a Forma Normal de Skolem.

Lo que haremos en este paso del proceso será eliminar los cuantificadores existenciales de la secuencia de cuantificadores Q, que prenexan la forma normal conjuntiva QM'. Para eliminarlos, como ya hemos dicho, sustituiremos las variables asociadas a los cuantificadores existenciales dentro del alcance de los mismos, por funciones de skolem y los cuantificadores simplemente serán eliminados de la fórmula. A este proceso se le llama proceso de skolemización y la fórmula resultante se denominará forma normal de skolem. Intuitivamente, las funciones de skolem deberán depender de las variables acotadas universalmente que ocurran antes del cuantificador existencial y de las variables libres que se encuentren bajo el alcance del mismo.

La forma normal de skolem que obtenemos en este paso ya no es lógicamente equivalente a la F', resultante del paso 2. Sin embargo, como habíamos mencionado al principio del capítulo, existe un teorema, que formalizaremos más adelante, que afirma que la forma normal de skolem tiene modelo si y sólo si la fórmula F' lo tiene. Es importante notar que hasta antes de este paso veníamos obteniendo fórmulas lógicamente equivalentes a la fórmula original. Con este paso dejamos de tener la equivalencia lógica, sin embargo la noción de tener modelo es suficiente para nuestros propósitos.

En la práctica el paso a forma normal de skolem consta de los siguientes pasos :

Para cada cuantificador existencial de una fórmula dada :

- construimos la función de skolem correspondiente.
- sustituimos las ocurrencias de la variable asociada al cuantificador, que se encuentren bajo el alcance del mismo, por las funciones de skolem.
- eliminamos el cuantificador existencial.

Para explicar mejor como se skolemiza una fórmula, analizaremos 3 ejemplos ilustrativos :

a. La fórmula afectada por el cuantificador existencial a eliminar, sólo contiene variables acotadas.
b. La fórmula afectada por el cuantificador existencial a eliminar, contiene variables libres y acotadas.
c. La fórmula afectada por el cuantificador existencial a eliminar, no contiene ni variables libres ni acotadas.

Caso a. Fórmula con variables acotadas.

Sea la fórmula

$$\forall x \exists y \; R(x,y) \qquad (I.10)$$

en donde la existencia de la variable y depende del valor de cada x. Para cada x tendremos la existencia de una y particular; por lo que y puede definirse como una función de x, que lleva a cada x en la y correspondiente :

$$\begin{aligned} x_1 &\dashrightarrow y_1 \\ x_2 &\dashrightarrow y_2 \\ &\vdots \end{aligned}$$

Para hacer explícita esta dependencia introducimos un símbolo funcional f, que no ocurra en la fórmula dada. Esta función f, será la función de skolem que sustituirá cada ocurrencia de la variable y bajo el alcance del cuantificador $\exists y$ por $f(x)$. Para eliminar el cuantificador existencial $\exists y$ en I.10, sustituimos

la ocurrencia de la variable y en la fórmula R(x,y) por la función de skolem f(x), obteniendo la fórmula :

$$\forall x\ R(x, f(x)).$$

Caso b. Fórmula con variables libres y acotadas.

Consideremos la siguiente fórmula

$$\forall x \exists y\ R(f(x,y),z) \qquad (I.11)$$

en donde la variable z es libre y la variable x acotada. Retomemos la interpretación I del ejemplo de las pags. 11 y 12, en donde el dominio de I son los números naturales, la interpretación del símbolo funcional f es la operación suma y la interpretación de R es la relación "mayor que". Entonces la fórmula I.11 se traduce como sigue: para todo número natural x, existe otro natural y, tal que x+y es mayor que z. Tratemos ahora de eliminar el cuantificador $\exists y$. Semanticamente, la variable y depende de la variable x, que aparece acotada universalmente, pero es claro que también depende de la variable libre z, por lo que, al hacer explícita esta dependencia con la función de skolem, tenemos que la función de skolem g deberá depender de x y z. Al eliminar el cuantificador existencial $\exists y$ obtenemos la fórmula :

$$\forall x\ R(f(x,g(x,z)),z).$$

Caso c. Fórmula que no contiene variables libres ni acotadas.

Supongamos que la fórmula que vamos a skolemizar tiene la forma:

$$\exists x\ P(x).$$

La función de skolem no tendrá argumentos, ya que no existen variables acotadas universalmente antes del cuantificador existencial, ni variables libres dentro del alcance del mismo. Una función de skolem con cero argumentos es, en realidad, una constante cualquiera que no ocurra en la fórmula. Por lo que la skolemización de la fórmula anterior será:

$$P(c)$$

donde c es una constante.

A continuación definiremos formalmente el proceso de skolemización de una fórmula.

Proceso de Skolemización.
Para definir formalmente el proceso de skolemización consideraremos dos fórmulas tipo :

i. Si la fórmula F, a skolemizar es del tipo

$$\forall x_1 \forall x_2 \ldots \forall x_n \exists x \; G(x_1, \ldots, x_n, z_1, \ldots, z_k) \; ,$$

y f es un símbolo funcional de grado n+k que no ocurre en F y $z_1, \ldots, z_k$ es la lista de variables libres de F, entonces la fórmula

$$\forall x_1 \forall x_2 \ldots \forall x_n \; G(x_1, \ldots, x_n, f(x_1, \ldots x_n, z_1, \ldots, z_k), z_1, \ldots, z_k)$$

es el resultado de skolemizar el cuantificador existencial $\exists x$ en F.

ii. Si la fórmula F es del tipo

$$\exists x \; G(x)$$

en donde ningún cuantificador universal esta prefijando al $\exists x$ y si c es un símbolo constante que no ocurre en F y G(x/c) es el resultado de sustituir en R todas las ocurrencias de x por c, entonces

$$G(x/c)$$

es el resultado de skolemizar el cuantificador $\exists x$ en F.

Es importante notar que si F es una forma normal prenex y skolemizamos todos sus cuantificadores existenciales, obtendremos una fórmula que solamente contendrá cuantificadores universales. Si una fórmula contiene solamente cuantificadores universales diremos que es universal y utilizaremos este concepto para definir formalmente la forma normal de skolem.

Forma Normal de Skolem.
Sea F una forma normal prenex y sean F1, F2, ..., Fn una secuencia de fórmulas tales que :

i. F1 = F.
ii. Fn es universal.
iii. Fi+1 es el resultado de skolemizar un cuantificador existencial en la fórmula Fi.

Entonces decimos que Fn es una forma normal de skolem de F, del tipo $\forall x_1 \ldots \forall x_k \; (C_1 \; \& \ldots \& \; C_m)$.

Por ejemplo si F es la siguiente forma normal prenex :

$$\exists x \forall z \exists y (R(x) \; \& \; (P(x,y) \; V \; Q(y,z,w)))$$

entonces

$$F1 : \forall z \exists y (R(f_1(w)) \; \& \; (P(f_1(w), y) \; V \; Q(y,z,w)))$$

es el resultado de skolemizar el cuantificador $\exists x$ en F y

$$F2 : \forall z(R(f_1(w)) \ \& \ (P(f_1(w), f_2(z,w)) \ \vee \ Q(f_2(z,w), z, w)))$$

es una forma normal de skolem de F, que resulta de skolemizar el cuantificador $\exists y$ en F_1.

A continuación enunciamos el teorema que fundamenta esta parte del proceso.

Teorema 3.
Sea F una fórmula de la lógica de primer orden en forma normal prenex con matriz en forma normal conjuntiva. El proceso de skolemización genera una forma normal de skolem G, tal que

F tiene modelo si y sólo si G tiene modelo.

Es importante notar, como ya habíamos mencionado, que después de los primeros dos pasos del proceso: paso a forma normal prenex y paso a forma normal conjuntiva, tenemos una fórmula F2 lógicamente equivalente a la fórmula original; después de aplicar el proceso de skolemización obtenemos la forma normal de skolem que no es lógicamente equivalente a F2 ya que :

$$\models G(c) \dashrightarrow \exists x \, G(x)$$

pero la implicación inversa no es cierta

$$\not\models G(c) \dashleftarrow \exists x \, G(x).$$

Entonces, después de skolemizar, solamente podemos garantizar que la forma normal de skolem F3 tiene modelo, si y sólo si, la fórmula F2 lo tiene. Pero esto, como ya habíamos dicho, es suficiente para nuestros propósitos.

I.3.4 Paso a forma Clausular.

Lo último que haremos será transformar la fórmula F3, en forma normal de skolem, a su forma clausular correspondiente. Lo único que tenemos que hacer para ello es eliminar los cuantificadores universales de F3. Cuando los hayamos eliminado, asumiremos que todas las variables de la fórmula resultante están acotadas universalmente. Eliminar los cuantificadores universales de una fórmula se justifica con el siguiente lema, que afirma que dada una interpretación I, I hace verdadera a F, si y sólo si, I hace verdadera a $\forall$ F.

Lema 5.
Para toda fórmula F y para toda interpretación I,

I es modelo de F si y sólo si I es modelo de $\forall$ F.

Entonces sea F3 una forma normal de skolem del tipo :

$$F3 = \forall x_1 \forall x_2 \ldots \forall x_k \; G(x_1, \ldots, x_n)$$

en donde G es una fórmula abierta en forma normal conjuntiva y n≥k, utilizando el Lema 5 tenemos que :

$$I = \forall x_1 \ldots \forall x_k \; G(x_1, \ldots, x_n) \quad \text{--->} \quad I = G(x_1, \ldots, x_n).$$

De aquí en adelante trabajaremos con G en lugar de $G(x_1, \ldots, x_n)$ y asumiremos que todas las variables que ocurren en las cláusulas que conforman a G estan acotadas universalmente; esto es posible ya que sabemos que todo lo que sea verdadero para la cerradura universal de G lo es para G. Ahora obtendremos a partir de G, el conjunto de cláusulas que conforman su forma clausular. La fórmula G es de la forma

$$G = C_1 \; \& \; C_2 \; \& \ldots \& \; C_m$$

en donde cada C_i es una cláusula para $i = 1, \ldots, m$; la forma clausular FC, será el conjunto :

$$FC = \{C_1, C_2, \ldots, C_m\}.$$

Veamos con el siguiente ejemplo como obtener la forma clausular de una fórmula dada. Si la forma normal de skolem es

$$\forall z(R(c) \; \& \; (P(c, f(z,w)) \; V \; Q(f(z,w), z, w)))$$

al eliminar los cuantificadores universales obtendremos :

$$R(c) \; \& \; (P(c, f(z,w)) \; V \; Q(f(z,w), z, w))$$

y la forma clausular correspondiente es el conjunto :

$$\{ \; R(c) \; , \; P(c, f(z,w)) \; V \; Q(f(z,w), z, w) \; \}$$

Claramente, cualquier interpretación es modelo del conjunto de cláusulas obtenido, si y sólo si, es modelo de la forma normal de skolem; lo que queda enunciado en este último teorema.

Teorema 4.
Si el conjunto FC es la forma clausular correspondiente a la forma normal de skolem F3, entonces cualquier modelo de FC es modelo de F3 e inversamente.

I.4 Ejemplos de transformación.

Para terminar, ilustraremos con un ejemplo todo el proceso de transformación de una fórmula a su forma clausular. Sea F la siguiente fórmula :

$$\forall x(\forall y(P(y) \rightarrow R(y,x)) \rightarrow K(x)) \qquad (I.12)$$

Los cuatro pasos del proceso son :

Paso 1. Obtención de la forma normal prenex.

1.a. $\forall x(\sim\forall y(P(y) \rightarrow R(y,x)) \vee K(x))$	eliminación de la 2da implicación.
1.b. $\forall x(\sim\forall y(\sim P(y) \vee R(y,x)) \vee K(x))$	eliminación de la 1a implicación.
1.c. $\forall x(\exists y\sim(\sim P(y) \vee R(y,x)) \vee K(x))$	eliminación de los cuantificadores negados.
1.d. $\forall x\exists y(\sim(\sim P(y) \vee R(y,x)) \vee K(x))$	prenexación.

Forma Normal Prenex F1 :

$$\forall x\exists y(\sim(\sim P(y) \vee R(y,x)) \vee K(x))$$

Paso 2. Paso a forma normal conjuntiva.

2.a. $\forall x\exists y((P(y) \& \sim R(y,x)) \vee K(x))$	reducción del alcance de las negaciones.
2.b. $\forall x\exists y((P(y) \vee K(x)) \& (\sim R(y,x) \vee K(x)))$	paso a FNC.

Forma Normal Conjuntiva F2 :

$$\forall x\exists y((P(y) \vee K(x)) \& (\sim R(y,x) \vee K(x)))$$

Paso 3. Paso a Forma Normal de Skolem.

3. $\forall x((P(f(x)) \vee K(x)) \& (\sim R(f(x),x) \vee K(x))$	skolemización del cuantificador $\exists y$.

Forma Normal de Skolem F3 :

$$\forall x((P(f(x)) \vee K(x)) \& (\sim R(f(x),x) \vee K(x))$$

Paso 4. Paso a forma clausular.

4.a. $((P(f(x)) \vee K(x)) \& (\sim R(f(x),x) \vee K(x))$	eliminación de los cuantificadores universales.
4.b. $FC = \{ P(f(x)) \vee K(x) , \sim R(f(x),x) \vee K(x) \}$	conjunto de cláusulas.

Forma clausular F4 :

$$F4 = \{ P(f(x)) \vee K(x) , \sim R(f(x),x) \vee K(x) \}$$

Las fórmulas I.12, F1 y F2 son lógicamente equivalentes. F2 tiene modelo, si y sólo si, F3 lo tiene. Además, cualquier interpretación es modelo de F3, si y sólo si, es modelo de F4. Por lo tanto, I.12 tiene modelo, si y sólo si, F4 lo tiene.

CAPITULO II.

Dos Variantes del Método Clásico.

En el capítulo anterior describimos el método clásico de transformación de una fórmula a forma clausular, al cual denominaremos método 1. Al skolemizar una fórmula, este método puede generar funciones de skolem con argumentos superfluos, ya que no analizamos cuales variables deben formar parte de los argumentos de las funciones de skolem y cuales no. El objetivo de este capítulo es describir y analizar dos variantes del método 1 que serán denominadas método 2 y método 3 respectivamente. Veremos que estos dos nuevos métodos optimizan el proceso de skolemización reduciendo el número de argumentos de las funciones de skolem, analizando cuales variables son las que realmente deben ser argumentos de las funciones.

Antes de presentar las dos variantes veamos cómo el método 1 efectivamente puede generar funciones de skolem con argumentos superfluos. Sea F la siguiente fórmula :

$$\forall x(\forall z(P(x,z)) \vee \exists w\, R(w,y)) \qquad (II.1)$$

aplicando el método 1 paso a paso tenemos la siguiente secuencia :

$\forall x\forall z\exists w(P(x,z) \vee R(w,y))$ al pasar a forma normal prenex.

$\forall x\forall z\exists w(P(x,z) \vee R(w,y))$ al pasar a forma normal conjuntiva.

$\forall x\forall z(P(x,z) \vee R(f(x,z,y),y))$ al pasar a forma normal de skolem.

$\{\ P(x,z) \vee R(f(x,z,y),y)\ \}$ al pasar a forma clausular.

La función de skolem que sustituyó a la variable w, es una función de 3 argumentos f(x,z,y), ya que el cuantificador existencial estaba bajo el alcance de los cuantificadores $\forall x$ y $\forall z$ en el momento de la skolemización y la variable y era libre dentro del alcance del $\exists w$. En realidad, esta función f podría tener menos argumentos pues la w en la fórmula II.1 no depende realmente de x y z, que sólo son argumentos del predicado P. Veamos que variando el procedimiento de skolemización del método 1 podemos mejorar la función de skolem.

Si en lugar de prenexar (sacar todos los cuantificadores hasta la izquierda de la fórmula) antes de skolemizar, tratamos de reducir el alcance de los cuantificadores introduciéndolos lo más que se pueda, podríamos transformar II.1 en :

$$\forall x\forall z(P(x,z)) \vee \exists w\, R(w,y) \qquad (II.2)$$

Esto es posible ya que la variable x en $\forall x$ no ocurre en la segunda fórmula de la disyunción [$\exists w\, R(w,y)$]. En general, para cualquier fórmula F(x) y D una fórmula que no contiene libre a x, la siguiente equivalencia es válida :

$$\forall x\,(F(x) \vee D) \quad \equiv \quad \forall x\, F(x) \vee D.$$

Entonces si ahora skolemizamos sobre II.2 obtenemos :

$$\forall x \forall z (P(x,z)) \quad V \quad R(f(y),y))$$

en donde la función de skolem f(y) tiene solamente un argumento ya que el cuantificador existencial no estaba bajo el alcance de ningun cuantificador universal y la única variable libre dentro del alcance del existencial era la y. Podemos decir entonces que la idea fundamental de este método, al que denominaremos **método 2** y que será descrito con detalle más adelante, es reducir el alcance de los cuantificadores y después skolemizar, logrando con esto que las funciones de skolem dependan de menos argumentos que con el método 1.

Otra forma de lograr reducir el número de argumentos en las funciones de skolem consiste en skolemizar sin mover los cuantificadores. A este método lo denominaremos **método 3**. Por ejemplo, si skolemizamos la fórmula II.1 directamente, obtendremos

$$\forall x (\forall z (P(x,z)) \ V \ R(f(y),y))$$

en donde la función f, solamente depende de la variable y ya que la variable x del $\forall x$ que esta afectando a $[\exists w \ R(w,y)]$ no ocurre realmente en la fórmula R(w,y). La escencia de este método consiste en analizar si las variables de los cuantificadores que estan afectando a la fórmula a skolemizar ocurren realmente o no dentro de ella, sin necesidad de transformar la fórmula.

La regla del método 3 sería entonces la siguiente :

a) toda variable x acotada por un cuantificador universal, anterior al existencial a skolemizar, será argumento de la función de skolem, si y sólo si, x ocurre dentro del alcance del cuantificador existencial.

b) toda variable libre bajo el alcance del cuantificador existencial a skolemizar y no acotada por cuantificadores universales anteriores, será argumento de la función de skolem.

Nótese que el conjunto formado por las variables mencionadas en a) y en b) puede ser visto como el conjunto de las variables libres dentro del alcance del cuantificador, por lo que con este método, los argumentos de la función de skolem serán exactamente las variables libres bajo el alcance del cuantificador existencial a skolemizar.

Dentro de esta primera parte del capítulo hemos descrito los primeros pasos de los dos nuevos métodos que transforman una fórmula a su forma clausular. A continuación explicaremos con detalle los métodos 2 y 3 y después analizaremos los casos en los cuales es mejor aplicar el método 2 que el método 3 y viceversa.

II.1 El Método 2.

Cuando describimos el método 1 en el capítulo I, vimos que antes de prenexar una fórmula era necesario tener una fórmula rectificada (I.3.1,pag.18). Para poder skolemizar con el método 2 también es necesario que la fórmula este rectificada; por lo que supondremos que antes de empezar con el proceso de transformación, las variables acotadas de la fórmula han sido renombradas y los cuantificadores vacuos eliminados. El proceso de transformación con este método constará entonces de los siguientes pasos :

1. Introducción de los cuantificadores.
2. Paso a forma normal de skolem.
3. Paso a forma normal conjuntiva y después a forma clausular.

Las diferencias escenciales entre el método 2 y el método 1 son las siguientes :

a) Con el método 1 sacamos los cuantificadores hasta el principio de la fórmula. Con el método 2 hacemos exactamente lo contrario, introducimos los cuantificadores en la fórmula para reducir el alcance de los mismos , logrando con esto que al skolemizar las funciones de skolem dependan de menos argumentos que con el método 1.

b) Con el método 1, después de prenexar la fórmula tenemos una fórmula de la forma QF, en donde Q es una secuencia de cuantificadores universales y existenciales, y F es una fórmula *abierta*; por esta razón, podemos transformar F a forma normal conjuntiva antes de skolemizar. Con el método 2 introducimos los cuantificadores dentro de la fórmula, por lo que la fórmula ya no es abierta y por esto no podemos pasar a forma normal conjuntiva antes de skolemizar.

A continuación explicaremos cada uno de los pasos del proceso de transformación.

Paso 1. Introducción de los cuantificadores.

Antes de skolemizar una fórmula, reduciremos el alcance de los cuantificadores haciendo así, que la función de skolem dependa de un número menor de variables.

Para poder utilizar las equivalencias, que nos permiten reducir el alcance de los cuantificadores, es necesario que la fórmula no contenga conectivos de la forma $\rightarrow$ y $\leftrightarrow$ y que las negaciones afecten únicamente a fórmulas atómicas. Entonces el primer paso consistirá en eliminar los conectivos $\rightarrow$ y $\leftrightarrow$ de la fórmula y reducir el alcance de las negaciones de manera análoga a como lo hicimos en el capítulo I (I.3.1,pags.17,18 y I.3.2,pags.21,22).

Las equivalencias lógicas que utilizaremos serán las siguientes :

$$\forall x(F(x) \,\&\, G(x)) \equiv \forall x\, F(x) \,\&\, \forall x\, G(x).$$
$$\exists x(F(x) \vee G(x)) \equiv \exists x\, F(x) \vee \exists x\, G(x).$$
$$\forall x(F(x) \,\&\, D) \equiv \forall x\, F(x) \,\&\, D.$$
$$\exists x(F(x) \vee D) \equiv \exists x\, F(x) \vee D.$$
$$\forall x(F(x) \vee D) \equiv \forall x\, F(x) \vee D.$$
$$\exists x(F(x) \,\&\, D) \equiv \exists x\, F(x) \,\&\, D.$$
$$\forall x\forall y\, F(x,y) \equiv \forall y\forall x\, F(x,y).$$
$$\exists x\exists y\, F(x,y) \equiv \exists y\exists x\, F(x,y).$$

donde $F(x)$, $G(x)$ y $F(x,y)$ son fórmulas cualesquiera y D es una fórmula en la cual x no ocurre libre.

Las primeras seis equivalencias pueden resumirse diciendo que un cuantificador universal que afecta a una fórmula con conectivo & siempre se distribuye y que un cuantificador existencial que afecta a una fórmula con conectivo V también se distribuye; mientras que un cuantificador universal $\forall x$, que afecta a una disyunción, se distribuye solamente si la x ocurre en sólo uno de los argumentos de la disyunción, y un cuantificador existencial $\exists x$, que afecta a una conjunción, se distribuye sólo si la x ocurre en uno de los argumentos de la conjunción.

La séptima equivalencia lo que nos dice es que los cuantificadores universales son conmutativos, es decir, que dados dos cuantificadores universales, podemos cambiar su orden y esto no afecta a la fórmula. La octava equivalencia es la dual de la séptima ya que habla de la conmutatividad de cuantificadores existenciales.

La séptima equivalencia la utilizaremos cuando una fórmula contenga dos o más cuantificadores universales seguidos y el alcance del cuantificador, de más a la derecha, no pueda ser reducido. Por ejemplo, si tenemos la fórmula

$$\forall x\forall y(R(x,y) \vee \exists z\, P(y,z))$$

el alcance del cuantificador $\forall y$ no puede reducirse, por lo que intercambiamos los cuantificadores utilizando la séptima equivalencia y obtenemos la siguiente fórmula :

$$\forall y\forall x(R(x,y) \vee \exists z\, P(y,z)) \qquad (II.3)$$

Ahora, el alcance del $\forall x$ puede reducirse utilizando la cuarta equivalencia y así obtenemos una fórmula equivalente a la II.3 :

$$\forall y(\forall x(R(x,y)) \vee \exists z\, P(y,z))$$

Ya que hemos reducido el alcance de los cuantificadores definiremos el proceso de skolemización para el método 2.

Paso 2. Paso a Forma Normal de Skolem.

Lo primero que haremos será describir el proceso mediante el cual skolemizamos un cuantificador existencial de una fórmula. Después del paso 1 del proceso tenemos una fórmula F rectificada, no abierta, que hay que skolemizar. Sea $\exists y\, D$ el primer cuantificador existencial con el que nos encontramos al recorrer la fórmula F de izquierda a derecha. La subfórmula $\exists y\, D$ puede estar bajo el alcance de $\forall x_1 \ldots \forall x_m$ con $m \geq 0$. Sean $z_1, \ldots, z_n$ las variables libres de $(\forall x_1 \ldots \forall x_m \exists y\, D)$ con $n \geq 0$ y sea f un símbolo funcional de grado n+m que no ocurra en F, entonces

i. Si $m>0$ ó $n>0$,

$$\forall x_1 \ldots \forall x_m\, D(y/f(x_1,\ldots,x_m,z_1,\ldots,z_n))$$

es el resultado de skolemizar el cuantificador $\exists y$; en donde $D(y/f(x_1,\ldots,x_m,z_1,\ldots,z_n))$ es el resultado de sustituir todas las ocurrencias de la variable y por $f(x_1,\ldots,x_m,z_1,\ldots,z_n)$ en D.

ii. Si $m=0$ y $n=0$,

$$D(y/c)$$

es el resultado de la skolemización sustituyendo todas las ocurrencias de la variable y por una constante c que no ocurra en F.

Después de ver como se lleva a cabo la skolemización de un cuantificador existencial, definiremos la forma normal de skolem, que se obtiene eliminando todos los cuantificadores existenciales de la fórmula obtenida en el paso 1.

Forma Normal de Skolem.

Sea F una fórmula rectificada que no contiene conectivos de la forma $\rightarrow$ y $\leftrightarrow$, en la cual las negaciones afectan sólo a fórmulas atómicas y el alcance de los cuantificadores ha sido reducido al mínimo y sea F1,...,Fk una secuencia de fórmulas tales que

i. F1 = F.
ii. Fk es universal.
iii. Fi+1 es el resultado de skolemizar el primer cuantificador existencial que nos encontramos al recorrer la fórmula Fi de izquierda a derecha.

entonces decimos que Fk es una forma normal de skolem de F.

Veamos el paso a forma normal de skolem con el siguiente ejemplo. Sea

$$\forall x \exists w \forall u (\exists z\ P(w,z) \ \&\ (M(x,u,w) \ \vee\ \exists y\ Q(u,y))) \qquad (II.4)$$

la fórmula original. Aplicando el paso 1 para la reducción de los cuantificadores obtenemos la siguiente fórmula :

$$\forall x \exists w (\exists z\ P(w,z) \ \&\ \forall u (M(x,u,w) \ \vee\ \exists y\ Q(u,y)))$$

Ahora, iniciemos el paso 2 skolemizando el cuantificador $\exists w$ que aparece en la fórmula, para obtener la nueva fórmula :

$$F2 : \forall x (\exists z\ P(f_1(x),z) \ \&\ \forall u (M(x,u,f_1(x)) \ \vee\ \exists y\ Q(u,y)))$$

La función f_1 depende solamente de x ya que ésta es la única variable acotada universalmente anterior al cuantificador existencial y no existen variables libres en el alcance del $\exists w$. Ahora skolemicemos el $\exists z$ para obtener la fórmula :

$$F3 : \forall x (P(f_1(x),f_2(x)) \ \&\ \forall u (M(x,u,f_1(x)) \ \vee\ \exists y\ Q(u,y)))$$

la función de skolem $f_2(x)$ que sustituyó a la variable z sólo depende de x, ya que nuevamente ésta es la única variable anterior al $\exists z$ acotada universalmente y no existen variables libres dentro del alcance del existencial. Por último, al skolemizar $\exists y$ obtenemos :

$$F4 : \forall x (P(f_1(x),f_2(x)) \ \&\ \forall u (M(x,u,f_1(x)) \ \vee\ Q(u,f_3(x,u)))) \qquad ...(II.5)$$

en donde la variable y fué sustituida por $f_3(x,u)$ ya que las variables u y x eran las variables asociadas a los cuantificadores universales anteriores al existencial.

Si la fórmula II.4 la hubieramos skolemizado con el método 1, hubiésemos obtenido lo siguiente :

$$\forall x \forall u (P(f_1(x),f_2(x,u)) \ \&\ (M(x,u,f_1(x)) \ \vee\ Q(u,f_3(x,u)))) \qquad ...(II.6)$$

Si comparamos las funciones de skolem de las fórmulas II.5 y II.6, podemos notar que la función $f_2(x)$ que sustituye a la variable z en el método 2 depende solamente de x, mientras que la $f_2(x,u)$ obtenida con el método 1 depende de x y de u.

Al igual que con el método 1, la forma normal de skolem obtenida con el método 2 tiene modelo, si y sólo si, la fórmula original lo tiene.

Paso 3. Paso a forma clausular.

En este momento, la fórmula F2 con la que estamos trabajando solamente contiene cuantificadores universales y conectivos & y V.

Para llevar F2 a forma clausular es necesario transformar primero F2 a forma normal conjuntiva y para esto tenemos que sacar los cuantificadores universales hasta el principio de la fórmula y eliminarlos. Sabemos que cualquier fórmula es lógicamente equivalente a una forma normal prenex que tenga exactamente las mismas variables libres (teorema 1, Cap. I, pag.18); por lo que al sacar los cuantificadores universales hasta el principio de F2 obtenemos una fórmula F2' equivalente a F2. Además, sabemos por el Lema 5, del capítulo anterior, que de una fórmula cuantificada universalmente podemos obtener la fórmula sin la cuantificación, eliminando los cuantificadores y que todo lo que sea verdadero para la fórmula cuantificada lo es para la fórmula sin la cuantificación y viceversa. Por lo que de F2' podemos obtener una fórmula F2'' que no contenga cuantificadores universales; supondremos F2'' cerrada universalmente.

Es importante aclarar que al automatizar este paso, en lugar de sacar los cuantificadores universales hasta el principio de la fórmula y después eliminarlos, los eliminaremos del lugar en el que se encuentran, que para fines prácticos es exactactamente lo mismo.

Ahora transformaremos F2'' a su forma normal conjuntiva usando la siguiente equivalencia lógica :

$$(P \& Q) \vee R \equiv (P \vee R) \& (Q \vee R)$$

que distribuye la disyunción sobre la conjunción.

Si queremos llevar la fórmula II.5 a forma normal conjuntiva, primero sacamos los cuantificadores universales hasta el principio de la fórmula, los eliminamos y obtenemos :

$$P(f_1(x),f_2(x)) \& (M(x,u,f_1(x)) \vee Q(u,f_3(x,u))) \qquad (II.7)$$

En este caso, la fórmula ya se encuentra en forma normal conjuntiva por lo que no es necesario distribuir las disyunciones sobre las conjunciones.

En este momento tenemos una fórmula en forma normal conjuntiva del tipo $C_1 \& C_2 \& \ldots \& C_m$, donde cada C_i es una cláusula; entonces la forma clausular correspondiente será :

$$\{ C_1, C_2, \ldots, C_m \}$$

Siguiendo el ejemplo, la forma clausular correspondiente a la fórmula II.7 es :

$$\{ P(f_1(x),f_2(x)), \quad M(x,u,f_1(x)) \vee Q(u,f_3(x,u)) \}.$$

II.2 El Método 3.

Para poder transformar una fórmula F a forma clausular con este método, al igual que con los métodos 1 y 2, es necesario que la fórmula esté rectificada, lo que daremos por hecho. El proceso de transformación consta ahora solamente de los dos pasos siguientes :

1. Paso a forma normal de skolem.
2. Paso a forma normal conjuntiva y después a forma clausular.

En este método a diferencia de los otros dos, los cuantificadores no se mueven de lugar; la fórmula simplemente es skolemizada y después transformada a forma clausular. El paso 2 es idéntico al paso 3 del método 2, por lo que nos limitaremos a describir el paso a forma normal de skolem.

Paso 1. Paso a Forma Normal de Skolem.

Para poder skolemizar, como ya hemos dicho, la fórmula no debe contener conectivos de la forma $\rightarrow$ y $\leftrightarrow$ y además las negaciones deben afectar solamente a fórmulas atómicas, para lograr esto al igual que con el método 2, eliminaremos los conectivos $\rightarrow$ y $\leftrightarrow$ de la fórmula y reduciremos el alcance de las negaciones (I.3.1,pags.17,18 y I.3.2, pags.21,22).

A continuación, definiremos el proceso de skolemización correspondiente a este método 3. Sea F la fórmula a skolemizar; sea $\exists y$ D, el primer cuantificador existencial con el que nos encontramos al recorrer la fórmula de izquierda a derecha; sean $z_1,\ldots,z_n$ con $n\geq 0$ las variables libres de $\exists y$ D y f un símbolo funcional de grado n que no ocurra en F, entonces

i. Si $n>0$, la fórmula

$$D(y/f(z_1,\ldots,z_n))$$

es el resultado de skolemizar el cuantificador $(\exists y\ D)$ en la fórmula F; donde $D(y/f(z_1,\ldots,z_n))$ es el resultado de sustituir todas las ocurrencias de la variable y por la función $f(z_1,\ldots,z_n)$ en D.

ii. Si $n=0$, la fórmula

$$D(y/c)$$

es el resultado de la skolemización sustituyendo todas las ocurrencias de la variable y por una constante c que no ocurra en F.

Para ilustrar la importancia que tiene el hecho de skolemizar de izquierda a derecha, obtengamos la forma normal de skolem de la siguiente fórmula:

$$\forall z \exists y \exists w \, (P(z,y) \vee P(y,w) \vee P(w,z)) \qquad \text{(II.8)}$$

como la variable z es la única variable libre en la fórmula $\exists y \exists w \, (P(z,y) \vee P(y,w) \vee P(w,z))$, tenemos que :

$$F2 : \forall z \exists w \, (P(z,f_1(z)) \vee P(f_1(z),w) \vee P(w,z))$$

y como nuevamente z es la única variable libre bajo el alcance del $\exists w$, entonces la fórmula

$$F3 : \forall z \, (P(z,f_1(z)) \vee P(f_1(z),f_2(z)) \vee P(f_2(z),z))$$

es la forma normal de skolem de la fórmula II.8.

Es importante notar que si en la fórmula II.8 hubieramos eliminado primero el existencial $\exists w$, entonces la variable w hubiera sido reemplazada por $g_1(z,y)$ ya que la variable y es libre en $[\exists w \, (P(z,y) \vee P(y,w) P(w,z))]$ y después por $g_1(z,g_2(z))$ al eliminar el cuantificador $\exists y$. La simplicidad de $f_1(z)$ comparada con $g_1(z,g_2(z))$, es la razón por la cual eliminamos los cuantificadores existenciales de izquierda a derecha al skolemizar una fórmula.

Ahora veamos que sucede al skolemizar con el método 3 el ejemplo que hemos utilizado a lo largo del capitulo (fórmula II.4) y que a continuación reescribimos :

$$F1 : \forall x \exists w \forall u (\exists z \, P(w,z) \, \& \, (M(x,u,w) \vee \exists y \, Q(u,y)))$$

primero skolemicemos $\exists w$ obteniendo la fórmula :

$$F2 : \forall x \forall u (\exists z \, P(f_1(x),z) \, \& \, (M(x,u,f_1(x)) \vee \exists y \, Q(u,y)))$$

ahora skolemicemos $\exists z$ obteniendo :

$$F3 : \forall x \forall u (P(f_1(x),f_2(x)) \, \& \, (M(x,u,f_1(x)) \vee \exists y \, Q(u,y)))$$

y por último skolemicemos $\exists y$ obteniendo :

$$F4 : \forall x \forall u (P(f_1(x),f_2(x)) \, \& \, (M(x,u,f_1(x)) \vee Q(u,f_3(u)))) \qquad \text{...(II.9)}$$

que es la forma normal de skolem de II.4 bajo el método 3.

Si comparamos este resultado con la forma normal de skolem (II.8) obtenida como resultado de skolemizar con el método 1, notamos que nuevamente las funciones de skolem se optimizan con el método 3, ya que dependen de menos argumentos que con el método 1.

Para terminar con el método 3, diremos que al igual que con los métodos 1 y 2, la forma normal de skolem obtenida con este método tiene modelo, si y sólo si, la fórmula original lo tiene (1) y que la forma clausular tiene modelo, si y sólo si, la fórmula original lo tiene.

II.3 Análisis comparativo de los métodos 2 y 3.

Lo que haremos ahora será comparar los procesos de skolemización de los métodos 2 y 3, analizando las ventajas de uno u otro en su caso. Para esto compararemos primero las funciones de skolem obtenidas al aplicar los métodos 2 y 3 a la fórmula II.4 :

$$\forall x \exists w \forall u (\exists z\ P(w,z) \ \&\ (M(x,u,w) \ \vee\ \exists y\ Q(u,y))) \qquad (II.4)$$

El resultado del método 2 es :

$$\forall x (P(f_1(x), f_2(x)) \ \&\ \forall u (M(x,u,f_1(x)) \ \vee\ Q(u,f_3(x,u)))) \qquad (II.5)$$

y el resultado del método 3 es :

$$\forall x \forall u (P(f_1(x), f_2(x)) \ \&\ (M(x,u,f_1(x)) \ \vee\ Q(u,f_3(u)))) \qquad (II.6)$$

Analicemos el siguiente cuadro :

Variables Skolemizadas	Método 2	Método 3
w	$f_1(x)$	$f_1(x)$
z	$f_2(x)$	$f_2(x)$
y	$f_3(x,u)$	$f_3(u)$

Nótese que la única variante aparece con la sustitución de la variable y. Analicemos que sucedió con el método 2 :

> Tratamos de reducir el alcance de los cuantificadores en la fórmula II.4 : El alcance del cuantificador $\forall u$ pudo ser reducido y éste quedó cuantificando solamente al segundo argumento de la conjunción; pero el alcance del cuantificador existencial $\exists w$ no pudo ser reducido ya que el conectivo principal de la fórmula es una conjunción y la variable w ocurría en ambos argumentos de ella, entonces tampoco pudo reducirse el alcance del $\forall x$, por lo que este cuantificador universal queda afectando a toda la fórmula y entonces la x aparece como argumento de la función de skolem f_3.

Supongamos que introducimos los cuantificadores como acabamos de explicar arriba, y skolemizamos $\exists w$; la fórmula que obtendríamos después de esto sería :

$$\forall x (\exists z\ P(f_1(x), z) \ \&\ \forall u (M(x,u,f_1(x)) \ \vee\ \exists y\ Q(u,y)))$$

si ahora introducimos nuevamente el cuantificador $\forall x$ obtenemos :

$$\forall x \exists z (P(f_1(x), z)) \ \& \ \forall u (\forall x (M(x, u, f_1(x))) \ V \ \exists y \ Q(u, y))$$

y es claro que si en este momento skolemizamos el $\exists y$, la función de skolem solamente dependerá de u, ya que ésta es la única variable asociada a un cuantificador universal que se encuentra afectando al $\exists y$. Este proceso que acabamos de discutir y que consiste en introducir los cuantificadores, skolemizar un existencial y volver a introducir los cuantificadores antes de seguir skolemizando es el que se encuentra "escondido" detrás del método 3 y por esto la función de skolem con el método 3 solamente depende de la variable u.

Tratemos ahora de mostrar algunas fórmulas típicas para las cuales el método 2 no es óptimo. Sean

$$P1 : \forall x \exists y \ [F(x, y) \ \& \ (G(y, x) \ V \ \exists u \ H(u))]$$

$$P2 : \forall x \exists y \ [F(x, y) \ \& \ (G(y, x) \ \& \ \exists u \ H(u))]$$

$$P3 : \forall x \exists w \forall y \ [F(y, w, x) \ V \ \exists u \ G(y, u)]$$

tales que F, G y H son fórmulas cualesquiera. Es importante notar que en P1 y en P2, la variable y del cuantificador existencial $\exists y$ no ocurre bajo el alcance del otro cuantificador existencial $\exists u$; si ocurriera, el resultado de skolemizar con el método 2 sería el mismo que con el método 3. En P1, P2 y P3 la skolemización con el método 2 no será óptima ya que el alcance del cuantificador externo de más a la derecha ($\exists y$ y $\forall y$) no puede ser reducido; esto ocasiona que el alcance de los demás cuantificadores tampoco pueda serlo provocando que las funciones de skolem dependan de argumentos superfluos. En estos casos el método 3 generará mejores funciones de skolem.

Sin embargo el método 3 no siempre es mejor que el método 2. Analicemos el siguiente ejemplo :

$$\forall x \exists w \forall u (\exists z \ P(w, z) \ V \ (M(x, u, w) \ \& \ \exists y \ Q(u, y))) \qquad (II.10)$$

al introducir los cuantificadores y skolemizar con el método 2 obtenemos :

$$\exists w \exists z \ P(w, z) \ V \ (\forall x \exists w \forall u \ M(x, u, w) \ \& \ \forall u \exists y \ Q(u, y))$$

$$\exists z \ P(c_1, z) \ V \ (\forall x \exists w \forall u \ M(x, u, w) \ \& \ \forall u \exists y \ Q(u, y))$$

$$P(c_1, c_2) \ V \ (\forall x \exists w \forall u \ M(x, u, w) \ \& \ \forall u \exists y \ Q(u, y))$$

$$P(c_1, c_2) \ V \ (\forall x \forall u \ M(x, u, f_1(x)) \ \& \ \forall u \exists y \ Q(u, y))$$

$$\boxed{P(c_1, c_2) \ V \ (\forall x \forall u \ M(x, u, f_1(x)) \ \& \ \forall u \ Q(u, f_2(u)))}$$

al skolemizar con el método 3 obtenemos :

$$\forall x \forall u (\exists z\ P(f_1(x), z) \ \vee\ (M(x, u, f_1(x)) \ \&\ \exists y\ Q(u, y)))$$

$$\forall x \forall u (P(f_1(x), f_2(x)) \ \vee\ (M(x, u, f_1(x)) \ \&\ \exists y\ Q(u, y)))$$

$$\boxed{\forall x \forall u (P(f_1(x), f_2(x)) \ \vee\ (M(x, u, f_1(x)) \ \&\ Q(u, f_2(u))))}$$

Nótese que la fórmula obtenida con el método 2 es óptima con respecto a la del método 3. Veamos que sucedió :

. Al introducir los cuantificadores con el método 2 pudimos diferenciar la variable w del predicado P de la variable w del predicado M. Por lo que podríamos pensar la fórmula como:

$$\exists w_1 \exists z\ P(w_1, z) \ \vee\ (\forall x \exists w \forall u\ M(x, u, w) \ \&\ \forall u \exists y\ Q(u, y))$$

lo cual al skolemizar w_1 nos da $P(c_1, z)$ y al skolemizar w obtenemos $M(x, u, f_1(x))$.

. Ahora con el método 3, al analizar la fórmula, la w de P y de M es considerada como la misma variable y como la w de M si depende de x la función de skolem $f_1(x)$ es sustituida en ambas ocurrencias. La variable z, que es skolemizada después, se ve afectada por la x en $f_1(x)$ y por lo tanto la función de skolem que sustituye a z se hace depender de x.

Lo único que podemos concluir de esta comparación es que ninguno de los dos métodos es el mejor, puesto que depende de la forma original de la fórmula el que un método de mejores resultados que el otro.

Sin embargo, tenemos una proposición para mejorar el método 2 que creemos igualará los resultados del método 3, y como además existe un caso (fórmula II.10) para el cual el propio método 2 es mejor que el método 3, podemos pensar que esta mejora será el método óptimo para llevar a cabo la traducción.

La mejora consiste en la posibilidad de volver a introducir los cuantificadores después de la skolemización de un cuantificador existencial y antes de proseguir con la siguiente skolemización (ver el caso presentado en el ejemplo II.4 al analizar los métodos 2 y 3).

Para terminar analizaremos una fórmula para la cual ni el método 2, ni el método 3, ni la mejora al método 2 que acabamos de describir, generan la función de skolem óptima y propondremos una segunda mejora al método 2 para este tipo de fórmulas. Consideremos el siguiente ejemplo :

$$\forall z \exists x \forall y\ (P(y, y) \ \vee\ (Q(y, z) \ \&\ R(x, y))) \qquad \text{(II.11)}$$

Skolemicemos la fórmula con el método 2; como el alcance del cuantificador ∀y no puede ser reducido, entonces el alcance de los otros dos cuantificadores tampoco puede serlo y al skolemizar obtenemos :

$$\forall z \forall y (P(y,y) \vee (Q(y,z) \& R(f(z),y)))$$

Si skolemizamos con el método 3 obtenemos la fórmula :

$$\forall z \forall y (P(y,y) \vee (Q(y,z) \& R(f(z),y)))$$

Con ambos métodos, la función que sustituye a la variable x es f(z). Si utilizamos el método 2 y analizamos la fórmula II.11, observamos que la razón por la cual el cuantificador externo ∀y no pudo ser introducido es la siguiente : el conectivo principal es la disyunción, y la variable y ocurre en ambos argumentos de ésta; sin embargo, si antes de introducir los cuantificadores distribuimos la disyunción en la fórmula, obtenemos :

$$\forall z \exists x \forall y [(P(y,y) \vee Q(y,z)) \& (P(y,y) \vee R(x,y))]$$

Ahora el cuantificador ∀y si puede ser introducido ya que el conectivo principal es la conjunción; aplicando las equivalencias 1, 6 y 3 (pag. 34) podemos introducir los cuantificadores obteniendo :

$$\forall z \forall y (P(y,y) \vee Q(y,z)) \& \exists x \forall y (P(y,y) \vee R(x,y)) \quad (II.12)$$

skolemizando el ∃x en la fórmula II.12, tenemos que la función de skolem que sustituye a la variable x es simplemente una constante c :

$$\forall z \forall y (P(y,y) \vee Q(y,z)) \& \forall y (P(y,y) \vee R(c,y)).$$

Lo que hicimos en este último ejemplo para que el cuantificador pudiera ser introducido, fue "cambiar" el conectivo principal de la fórmula. La segunda mejora que proponemos entonces para el método 2, cuando el cuantificador externo de más a la derecha no pueda ser introducido a la fórmula, es la siguiente :

1. Si el conectivo principal es una disyunción, y si una de las fórmulas que son argumentos de la disyunción contiene una conjunción, tratamos de distribuir la disyunción para obtener finalmente una conjunción de disyunciones. Este es el caso de la fórmula II.11 que hemos analizado.

2. Si el conectivo principal es una conjunción, y si una de las fórmulas que son argumentos de la conjunción contiene una disyunción, tratamos de distribuir la conjunción para obtener finalmente una disyunción de conjunciones.

Al cambiar el conectivo principal, el cuantificador externo de más

a la derecha podrá ser introducido dando así la posibilidad de que al skolemizar obtengamos una mejor función de skolem.

En [3], se propone un pedazo de programa que transforma una fórmula cerrada de primer orden a forma clausular; este programa se asemeja bastante a la automatización del método 1 y es por esto que no llevamos a cabo la implantación del primer método presentado. En el siguiente capítulo, describiremos la automatización de los métodos 2 y 3. Los programas que automatizan las mejoras al método 2 son un poco más complejos y no fueron desarrollados como parte de este trabajo, pero quedan abiertos para un futuro.

CAPITULO III.

El Predicado Central de la Implantación.

En este capítulo describiremos la realización en PROLOG de los métodos 2 y 3 presentados en el capítulo anterior. En una primera parte del capítulo definiremos la sintaxis aceptada por el programa para las fórmulas de primer orden. En una segunda parte presentaremos la codificación de la estructura principal de ambos métodos y por último, describiremos algunos procedimientos necesarios para la realización de ambos métodos y las diferencias entre ambos programas.

La programación de los métodos 2 y 3 fue hecha en PROLOG ya que la recursividad y la manipulación simbólica del lenguaje facilitan la implantación de los procesos. El intérprete usado para correr el programa fue Prolog-86.

III.1 Sintaxis utilizada en el programa.

Para cuantificar las fórmulas utilizaremos dos predicados, all y exists que definiremos como sigue :

. Una fórmula del tipo $\forall x$ F se denotará en PROLOG como

all(x,F)

. Una fórmula del tipo $\exists x$ F se denotará como

exists(x,F)

en ambos casos el primer argumento representa la variable x asociada al cuantificador y el segundo la fórmula bajo su alcance.

La variable x asociada al all y al exists, así como las variables de la fórmula F deberán ser letras minúsculas. Esta restricción se debe a que las letras mayúsculas tienen un significado específico en PROLOG. Para poder distinguir las variables de las constantes en una fórmula, tendremos la siguiente convención :

. Las constantes empezarán con la letra minúscula c o bien, la letra c seguida de cualquier otra secuencia de letras y números. Por ejemplo: c, c1, c2, cp, casa, etc. son constantes.

. Las variables podrán empezar con cualquier letra minúscula del alfabeto a excepción de la letra c. Por ejemplo : x1, y, a, m, etc. son variables.

. Los nombres de los predicados podrán formarse con cualquier sucesión de letras minúsculas. Por ejemplo : en p(x) y en mas(x,y,z), p y mas son los nombres de los predicados.

. Los conectivos lógicos se denotarán como sigue :

~	es el símbolo para la negación.
#	es el símbolo para la disyunción.
&	es el símbolo para la conjunción.
→	es el símbolo para la implicación.
↔	es el símbolo para la equivalencia.

Por ejemplo, la fórmula :

all(x, (p(x) # r) → ((p(x) & ~r) # (r & ~p(x))))

se lee: para toda x, p(x) o bien r, implica que, o bien p(x) y no r, o bien r y no p(x).

III.2 Estructura Principal del Programa.

A continuación presentaremos el algoritmo principal para la codificación de los métodos 2 y 3. En la traducción de una fórmula F a forma clausular, estos dos métodos tienen una parte en común que podemos resumir de la siguiente manera :

1. Eliminar los conectivos → y ↔ de la fórmula F, obteniendo como resultado una fórmula F1.
2. Introducir las negaciones en F1, obteniendo como resultado una fórmula F2.
3. Skolemizar F2 para obtener como resultado la fórmula F3, sin cuantificadores existenciales.
4. Pasar F3 a la forma clausular correspondiente obteniendo así, la representación en cláusulas de la fórmula original.

La realización de estos cuatro pasos estará definida por el predicado traduce(F,Cls), quien traduce la fórmula F de lógica de primer orden a su forma clausular Cls :

```
traduce(F,Cls) :- eli_imp(F,F1),
                  neg_dentro(F1,F2),
                  pre_skolem(F2,F3),
                  fcl(F3,Cls).
```

es decir, traduce(F,Cls) lleva a la fórmula F en una forma clausular Cls si

i) F1 es el resultado de eliminar en F los conectivos → y ↔.

ii) F2 es el resultado de introducir las negaciones a fórmulas atómicas en F1.

iii) F3 es el resultado de skolemizar en F2 los cuantificadores existenciales.

iv) Cls es la lista de cláusulas resultantes de la fórmula F3.

Los cuatro predicados que definen a *traduce* : *eli_imp*, *neg_dentro*, *pre_skolem* y *fcl*, corresponden a los cuatro pasos comunes de los métodos 2 y 3 arriba descritos. La codificación de los predicados *eli_imp*, *neg_dentro* y *fcl* será idéntica para ambos métodos. Sin embargo, la codificación del predicado *pre_skolem* varia.

En el capitulo II dijimos que antes de transformar una fórmula a forma clausular ya fuera con el método 2 o con el método 3, era necesario rectificarla, es decir :

1. renombrar las variables acotadas de la fórmula logrando con esto que la fórmula no contenga ocurrencias acotadas y libres de una misma variable y que no contenga dos o más cuantificadores que acoten ocurrencias de una misma variable.

2. eliminar los cuantificadores vacuos.

Supondremos que la fórmula F de *traduce*, es una fórmula cuyas variables acotadas ya han sido renombradas. El predicado que lleva a cabo este proceso será analizado en el siguiente capítulo. La eliminación de los cuantificadores vacuos se llevará a cabo dentro del predicado *pre_skolem* en ambos métodos, antes de skolemizar la fórmula.

Veamos como trabajan los predicados que son llamados por el predicado *traduce*. El primer argumento de *traduce* será una fórmula que sólo podrá tener alguna de las siguientes formas :

1)	all(x,P)	cuantificación universal de P.
2)	exists(x,P)	cuantificación existencial de P.
3)	P ↔ Q	equivalencia entre las fórmulas P y Q.
4)	P → Q	implicación.
5)	P & Q	conjunción de P y Q.
6)	P ∨ Q	disyunción de P y Q.
7)	~P	negación de P.
8)	P	fórmula atómica.

donde P y Q son variables que representan fórmulas cualesquiera.

Cada uno de los predicados que son llamados por *traduce* es recursivo y trabaja por casos según la forma de la fórmula. Por ejemplo, la definición del predicado correspondiente a *eli_imp* va a consistir de ocho reglas. Cada regla tomará en cuenta una de las formas mencionadas. Así, dependiendo de su forma, la fórmula unificará con la regla correspondiente. Comencemos ahora por analizar el predicado que elimina las implicaciones.

III.2.1.Eliminación de las implicaciones. (*eli_imp*).

Para la eliminación de los conectivos → y ↔ usaremos el predicado eli_imp(F,F1), en donde F1 es la fórmula que generamos al eliminar de F todos los conectivos → y ↔. Las definiciones del predicado según el tipo de la fórmula son las siguientes :

```
E1: eli_imp(P <-> Q, (P1 & Q1) # (~P1 & ~Q1)) :-
                        !,
                        eli_imp(P,P1),
                        eli_imp(Q,Q1).
E2: eli_imp(P -> Q, ~P1 # Q1) :-
                        !,
                        eli_imp(P,P1),
                        eli_imp(Q,Q1).
E3: eli_imp(all(X,P), all(X,P1)) :-
                        !,
                        eli_imp(P,P1).
E4: eli_imp(exists(X,P), exists(X,P1)) :-
                        !,
                        eli_imp(P,P1).
E5: eli_imp(P & Q, P1 & Q1) :-
                        !,
                        eli_imp(P,P1),
                        eli_imp(Q,Q1).
E6: eli_imp(P # Q, P1 # Q1) :-
                        !,
                        eli_imp(P,P1),
                        eli_imp(Q,Q1).
E7: eli_imp(~P, ~P1) :-
                        !,
                        eli_imp(P,P1).
E8: eli_imp(P, P).
```

Es importante notar que cada regla corresponde a una equivalencia lógica. Si la fórmula F tiene la forma P <-> Q, entonces la regla a utilizar será la primera, que corresponde a la siguiente equivalencia :

$$P \leftrightarrow Q \quad \equiv \quad (P \,\&\, Q) \,\#\, (\neg P \,\&\, \neg Q)$$

Esta equivalencia permite transformar P <-> Q en una fórmula sin implicaciones ni equivalencias. Pero P y Q pueden contener a su vez equivalencias o implicaciones que es necesario eliminar, por esto *eli_imp* es utilizada nuevamente con las fórmulas P y Q. Todo esto es equivalente a decir que

$$P \leftrightarrow Q \quad \equiv \quad (P1 \,\&\, Q1) \,\#\, (\neg P1 \,\&\, \neg Q1)$$

en donde P1 y Q1 son el resultado de eliminar recursivamente las implicaciones y las equivalencias de P y Q respectivamente.

Si la fórmula F tiene la forma P -> Q, utilizaremos la segunda regla de la definición para transformarla aplicándole la equivalencia correspondiente, lo que implica decir que

$$P \rightarrow Q \quad \equiv \quad \neg P1 \,\#\, Q1$$

donde P1 y Q1 son los resultados de eliminar recursivamente las implicaciones y las equivalencias de P y Q respectivamente.

```
E1: eli_imp(P ↔ Q, (P1 & Q1) # (~P1 & ~Q1)) :-
                        !,
                        eli_imp(P,P1),
                        eli_imp(Q,Q1).
E2: eli_imp(P → Q, ~P1 # Q1) :-
                        !,
                        eli_imp(P,P1),
                        eli_imp(Q,Q1).
E3: eli_imp(all(X,P), all(X,P1)) :-
                        !,
                        eli_imp(P,P1).
E4: eli_imp(exists(X,P), exists(X,P1)) :-
                        !,
                        eli_imp(P,P1).
E5: eli_imp(P & Q, P1 & Q1) :-
                        !,
                        eli_imp(P,P1),
                        eli_imp(Q,Q1).
E6: eli_imp(P # Q, P1 # Q1) :-
                        !,
                        eli_imp(P,P1),
                        eli_imp(Q,Q1).
E7: eli_imp(~P, ~P1) :-
                        !,
                        eli_imp(P,P1).
E8: eli_imp(P, P).
```

Es importante notar que cada regla corresponde a una equivalencia lógica. Si la fórmula F tiene la forma P ↔ Q, entonces la regla a utilizar será la primera, que corresponde a la siguiente equivalencia :

$$P \leftrightarrow Q \quad \equiv \quad (P \,\&\, Q) \;\#\; (\sim P \,\&\, \sim Q)$$

Esta equivalencia permite transformar P ↔ Q en una fórmula sin implicaciones ni equivalencias. Pero P y Q pueden contener a su vez equivalencias o implicaciones que es necesario eliminar, por esto *eli_imp* es utilizada nuevamente con las fórmulas P y Q. Todo esto es equivalente a decir que

$$P \leftrightarrow Q \quad \equiv \quad (P1 \,\&\, Q1) \;\#\; (\sim P1 \,\&\, \sim Q1)$$

en donde P1 y Q1 son el resultado de eliminar recursivamente las implicaciones y las equivalencias de P y Q respectivamente.

Si la fórmula F tiene la forma P → Q, utilizaremos la segunda regla de la definición para transformarla aplicándole la equivalencia correspondiente, lo que implica decir que

$$P \rightarrow Q \quad \equiv \quad \sim P1 \;\#\; Q1$$

donde P1 y Q1 son los resultados de eliminar recursivamente las implicaciones y las equivalencias de P y Q respectivamente.

Si la fórmula F tiene la forma all(x,P) o exists(x,P), utilizaremos la tercera y cuarta regla respectivamente, haciendo la siguiente transformación : Los cuantificadores all o exists, según sea el caso, no cambian, y la variable a la que cuantifican tampoco cambia. Lo que puede cambiar es la fórmula P, que se encuentra bajo el alcance del cuantificador, ya que P puede contener implicaciones o equivalencias que debemos eliminar. Por esta razón es necesario llamar recursivamente a *eli_imp* con la fórmula P y generar en esta llamada P1, que será el resultado de eliminar las implicaciones y equivalencias de P.

Si la fórmula F tiene la forma (P & Q) o (P # Q), utilizaremos la quinta y sexta regla respectivamente. En este caso tendremos que checar si los argumentos de la conjunción o de la disyunción según sea el caso, contienen implicaciones o equivalencias. Por lo que el resultado de eliminar las implicaciones o equivalencias en las fórmulas (P & Q) y (P # Q) será (P1 & Q1) y (P1 # Q1) respectivamente, en donde P1 y Q1 serán los resultados de llamar recursivamente a *eli_imp* dos veces.

Si la fórmula tiene la forma ~P , utilizaremos la séptima regla checando si P contiene a su vez equivalencias o implicaciones, por lo que el resultado de eliminar las implicaciones y equivalencias en la fórmula ~P es ~P1, en donde P1 es el resultado de llamar recursivamente a *eli_imp*.

Si la fórmula tiene la forma P, es decir, si la fórmula es una fórmula atómica, utilizaremos el último hecho del procedimiento. Este hecho nos dice que el resultado de eliminar las implicaciones y equivalencias de una fórmula atómica P, es la misma fórmula P.

Si analizamos las ocho definiciones de *eli_imp*, nos damos cuenta que las siete primeras reglas que lo componen tienen un corte. Los cortes refuerzan el determinismo haciendo al programa reducir el número de unificaciones. El término de la recursión está dado con el último hecho de la definición.

Es importante hacer notar que el lugar que ocupa el único hecho de la definición no puede cambiar, ya que si el hecho fuera el primero, cualquier fórmula unificaría con él antes de unificar con otra regla y por lo tanto no haríamos ninguna transformación.

Lo que hemos explicado con respecto a los cortes y al orden de las reglas en este predicado será válido para todos los demás predicados que son llamados por *traduce*.

Veamos ahora como funciona *eli_imp* con el siguiente ejemplo. Si la fórmula F es :

$$all(x,\ all(y,\ p(y) \rightarrow r(y,x)) \rightarrow k(x)) \qquad \text{(III.1)}$$

y la llamada a *traduce* es la meta :

M: traduce(all(x, all(y, p(y) → r(y,x)) → k(x)), Cls)

la primera submeta generada será:

M1: eli_imp(all(x, all(y, p(y) → r(y,x)) → k(x)), F1)

que a su vez generará

M2: eli_imp(all(y, p(y) → r(y,x)) → k(x), G1)

y luego

M3: eli_imp(all(y, p(y) → r(y,x)), G2), eli_imp(k(x), G3)

La primera submeta M31 generará otra submeta eli_imp(p(y) → r(y,x), G4) y ésta a su vez dos más :

M4: eli_imp(p(y), G5) , eli_imp(r(y,x), G6)

las cuales unificarán con el último hecho E8 de la definición de *eli_imp* haciendo G8 igual a p(y) y G6 a r(y,x) y por lo tanto

G4 = ~p(y) # r(y,x)
G2 = all(y, ~p(y) # r(y,x))

La segunda submeta de M3, eli_imp(k(x),G3), unificará con el último hecho de la definición haciendo G3 = k(x). Finalmente tendremos G1 igual a

~all(y, ~p(y)#r(y,x)) # k(x)

y la fórmula resultante F1, será :

F1: all(x, ~all(y,~p(y)#r(y,x)) # k(x)) (III.2)

III.2.2 Introducción de las negaciones (*neg_dentro*).

Lo que haremos ahora es introducir las negaciones en la fórmula hasta que solamente queden negadas sus subfórmulas atómicas. Esto se lleva a cabo con el predicado neg_dentro(F1,F2), en donde F2 es el resultado de introducir las negaciones en F1. Recordemos que F1 es una fórmula que ya no contiene ni implicaciones ni equivalencias. Por esta razón, la definición de *neg_dentro* no considera estos dos casos y consiste de las siguientes reglas :

```
ND1: neg_dentro(~P, P1) :-
                        !,
                        neg(P,P1).
ND2: neg_dentro(all(X,P), all(X,P1)) :-
                        !,
                        neg_dentro(P,P1).
```

```
ND3: neg_dentro(exists(X,P), exists(X,P1)) :-
                        !,
                        neg_dentro(P,P1).
ND4: neg_dentro(P & Q, P1 & Q1) :-
                        !,
                        neg_dentro(P,P1),
                        neg_dentro(Q,Q1).
ND5: neg_dentro(P # Q, P1 # Q1) :-
                        !,
                        neg_dentro(P,P1),
                        neg_dentro(Q,Q1).
ND6: neg_dentro(P, P).
```

Con este predicado estamos "caminando" en la fórmula F1 hasta encontrar una negación; en el momento en el que la encontramos llamamos al predicado *neg*. Por lo que para cualquier tipo de fórmula, a excepción de una negación, *neg_dentro* se llama recursivamente hasta encontrar la negación de una subfórmula. Con el predicado neg(P,P1) estamos diciendo que P1 es la negación de P. *Neg* se encargará de introducir la negación hasta que tengamos solamente la negación de fórmulas atómicas y se definirá como sigue :

```
N1: neg(~P, P1) :-
                !,
                neg_dentro(P,P1).
N2: neg(all(X,P), exists(X,P1)) :-
                !,
                neg(P,P1).
N3: neg(exists(X,P), all(X,P1)) :-
                !,
                neg(P,P1).
N4: neg(P & Q, P1 # Q1) :-
                !,
                neg(P,P1),
                neg(Q,Q1).
N5: neg(P # Q, P1 & Q1) :-
                !,
                neg(P,P1),
                neg(Q,Q1).
N6: neg(P, ~P).
```

Estas definiciones están tomadas directamente de las siguientes equivalencias lógicas de la negación :

$$\sim\sim P \equiv P$$
$$\sim all(x,P) \equiv exists(x,\sim P)$$
$$\sim exists(x,P) \equiv all(x,\sim P)$$
$$\sim(P \,\&\, Q) \equiv \sim P \,\#\, \sim Q$$
$$\sim(P \,\#\, Q) \equiv \sim P \,\&\, \sim Q$$

Retomando el ejemplo de la sección anterior, apliquemos ahora el predicado *neg_dentro* a la fórmula III.2 resultante del predicado *eli_imp*. Sea entonces F1:

```
all(x, ~all(y,~p(y) # r(y,x))  #  k(x))
```

La primera llamada a *neg_dentro* es la submeta:

```
M5:     neg_dentro(all(x, ~all(y,~p(y)#r(y,x)) # k(x)), F2)
```

la cual unificará con la segunda regla ND2 de la definición de *neg_dentro* generando a su vez:

```
M6:     neg_dentro(~all(y, ~p(y)#r(y,x)) # k(x), G1)
```

Esta M6 unificará con la regla ND6 de la misma definición y generará dos submetas

```
M7:     neg_dentro(~all(y, ~p(y)#r(y,x)), G2),  neg_dentro(k(x),G3)
```

La primera submeta de M7 generará la siguiente llamada a *neg* :

```
M8:     neg(all(y, ~p(y)#r(y,x)), G2)
```

la cual generará

```
M9:     neg(~p(y) # r(y,x), G4)
```

que al unificar con la regla N3 de la definición de *neg* generará las submetas

```
M10:    neg(~p(y),G5),  neg(r(y,x),G6)
```

Estas dos últimas submetas de M10 unificarán con el hecho N6 de la definición haciendo : G5 = p(y) y G6 = ~r(y,x), y por lo tanto,

```
G4 = p(y) & ~r(y,x)
G2 = exists(y, p(y) & ~r(y,x))
```

La segunda submeta de M7, neg_dentro(k(x),G3), unificará con el último hecho ND6 de *neg_dentro* haciendo G3 = k(x). Entonces

```
G1 = exists(y, p(y) & ~r(y,x)) # k(x)
```

y la fórmula resultante F2 será :

```
F2: all(x, exists(y, p(y)&~r(y,x)) # k(x))   (III.3)
```

III.2.3. La Skolemización con el Método 2 (*skolem2*).

Ahora analizaremos la implantación de la skolemización utilizada en el método 2. El proceso estará dividido en dos partes :

a) En la primera parte introduciremos los cuantificadores hacia dentro de la fórmula para reducir al máximo el alcance de los mismos y eliminaremos aquellos que sean vacuos.
b) En la segunda parte llevaremos a cabo la skolemización de los existenciales.

El predicado utilizado para esto será pre_skolem(F2,F3), en donde F3 es el resultado de skolemizar la fórmula F2 obtenida en el paso anterior. *Pre_skolem* se definirá mediante tres predicados: el predicado *cuantif_dentro* para llevar a cabo la primera parte del proceso; *lee_funcion(S)* para leer de la pantalla el símbolo funcional con el que se generarán las funciones de skolem y *skolem2* para llevar a cabo la skolemización propiamente dicha :

```
pre_skolem(F2,F3) :- cuantif_dentro(F2,Fa),
                     lee_funcion(S),
                     skolem2(Fa,F3,[],[],Ls,S).
```

Las definiciones de *cuantif_dentro* y de *lee_funcion* las omitiremos en este capítulo por tediosas, pero pueden ser revisadas en el anexo 1. Aquí describiremos solamente el funcionamiento del predicado principal *skolem2* suponiendo que los procesos correspondientes a *cuantif_dentro* y a *lee_funcion* ya han sido llevados a cabo.

El predicado skolem2(Fa,F3,Us,Es,Ls,S) se interpretará de la siguiente manera: F3 es el resultado de skolemizar la fórmula Fa; Us, Es y Ls son respectivamente:la lista de variables acotadas universalmente, la lista de variables acotadas existencialmente y la lista de variables libres de la fórmula Fa, y S es el símbolo dado para generar las funciones de skolem.

Recordemos que Fa es una variante de F2, en la cual los cuantificadores han sido introducidos al máximo para reducir su alcance. Si el valor de S en *skolem2*, es por ejemplo la constante *f*, los nombres de las funciones de skolem que sustituirán a las variables acotadas existencialmente en Fa serán f_1, f_2, etc. y los argumentos de éstas se obtendrán como la unión de las listas Us y Ls. La llamada a *skolem2* contiene inicialmente las listas Us y Es vacías para ir acumulando en ellas las variables acotadas universalmente y existencialmente. Ls será instanciada con la lista de variables libres de Fa. Las tres listas generarán sus valores durante la ejecución del predicado *skolem2*.

La idea intuitiva de *skolem2* es la siguiente: recorremos la fórmula de izquierda a derecha acumulando en Us las variables que aparecen asociadas a cuantificadores universales hasta encontrar el primer cuantificador existencial. En este momento, creamos la lista de variables libres Ls que se encuentren bajo el alcance de este cuantificador y así con Us y Ls poder construir la función de skolem. Esta función sustituirá todas las ocurrencias de la variable cuantificada existencialmente dentro de la subfórmula bajo el alcance del cuantificador.

Recordemos que las fórmulas que vamos a skolemizar son fórmulas que solo contienen los conectivos &, V y la negación está aplicada solamente a fórmulas atómicas, por lo que las posibles formas de la fórmula Fa de *skolem2* son las siguientes :

```
all(x,P)
exists(x,P)
P & Q
P # Q
~P
P
```

el predicado *skolem2* se definirá entonces para cada uno de estos casos :

```
S1: skolem2(all(X,P),all(X,P1),Us,Es,Ls,S) :-
                                !,
                                skolem2(P,P1,[X|Us],Es,Ls,S).
S2: skolem2(exists(X,P),P2,Us,Es,Ls,S) :-
                                !,
                                skolem2(P,P1,Us,[X|Es],Ls,S),
                                concatena(Us,Ls,Args),
                                gensim(S,F),
                                Sk =.. [F|Args],
                                subst(X,Sk,P1,P2).
S3: skolem2(P & Q,P1 & Q1,Us,Es,Ls,S) :-
                                !,
                                skolem2(P,P1,Us,Es,Ls1,S),
                                skolem2(Q,Q1,Us,Es,Ls2,S),
                                concatena(Ls1,Ls2,Laux),
                                depura_lista(Laux,Ls,[]).
S4: skolem2(P # Q,P1 # Q1,Us,Es,Ls,S) :-
                                !,
                                skolem2(P,P1,Us,Es,Ls1,S),
                                skolem2(Q,Q1,Us,Es,Ls2,S),
                                concatena(Ls1,Ls2,Laux),
                                depura_lista(Laux,Ls,[]).
S5: skolem2(~P,~P1,Us,Es,Ls,S) :-
                                !,
                                skolem2(P,P1,Us,Es,Ls,S).
S6: skolem2(P,P,Us,Es,Ls,S) :-
                                !,
                                gen_var(P,Lps),
                                concatena(Us,Es,As),
                                diferencia(As,Lps,Ls).
```

La definición de *skolem2* para el caso del existe es la más compleja, ya que en ella eliminamos realmente los cuantificadores existenciales. Por esta razón la explicaremos primero.

Si Fa es de la forma exists(x,P) se aplica la definición S2 para: a) generar la función de skolem que sustituya todas las ocurrencias de la variable x bajo el alcance del exists y b) eliminar el cuantificador existencial.

La fórmula P que esta bajo el alcance del cuantificador existencial, puede contener a su vez otros existenciales que es necesario eliminar. Por esto, lo primero que hacemos es llamar a *skolem2* de P para obtener P1. Con ello eliminamos los cuantificadores existenciales de P, si es que estos existían, de adentro hacia afuera y obtenemos la lista Ls de variables libres que se encuentran bajo el alcance del existencial. Después de esto, se concatena Us y Ls para obtener los argumentos (Args) de la función de skolem Sk, que sustituirá a la variable x en P.

El nombre Nom de la función de skolem se obtiene con el predicado **gensim(S,Nom)** a partir del símbolo funcional S. El predicado univ (=..) genera la función Sk a partir de Args y Nom y el predicado **subst(x,Sk,P1,P2)** sustituye cada ocurrencia de x por Sk en P1 para obtener la fórmula P2 ya skolemizada.

Veamos ahora las definiciones restantes. Si la fórmula Fa está cuantificada universalmente sobre la variable x, definición S1, entonces x pasa a formar parte de la lista Us de variables acotadas universalmente y se vuelve a llamar a *skolem2* para procesar la fórmula P bajo el alcance del cuantificador.

Si la fórmula Fa es una conjunción de la forma P & Q, definición S3, llamamos a *skolem2* de P y a *skolem2* de Q para skolemizar a P y a Q. Para formar la lista Ls de variables libres de P & Q llamamos a **concatena(Ls1,Ls2,Las)**, que deja en Las la unión de Ls1, la lista de variables libres de P, y Ls2 , la lista de variables libres de Q. La lista Las es depurada de repeticiones por el predicado **depura_lista(Las,Ls,[])**, quedando en Ls la lista de variables libres de la conjunción P & Q ya depurada. El tratamiento de una disyunción, caso S4, es totalmente análogo al caso de una conjunción.

Si la fórmula Fa es la negación de una atómica, ~P, definición S5, llamamos a *skolem2* de P únicamente para obtener la lista Ls de variables libres de P. Por último, si la fórmula Fa es atómica, definición S6, sólo nos interesa generar la lista Ls de variables libres de P; para ello necesitamos generar la lista de variables de P con **gen_var(P,Lps)** y la lista As de variables acotadas concatenando las listas Us y Es que afectan a P. La lista Ls de variables libres se obtiene, entonces, con el predicado **diferencia(As,Lps,Ls)**.

Para terminar con el funcionamiento de *skolem2*, tomemos el ejemplo de la fórmula III.3 obtenida con la introducción de las negaciones y apliquémosle el predicado *skolem2*. La primera llamada a *skolem2*, después de haber introducido los cuantificadores y suponiendo que f es el símbolo funcional para las funciones de skolem es la siguiente submeta :

M11: **skolem2(all(x,exists(y,p(y)&~r(y,x))#k(x)), F3,[],[],Ls,f)**

que al unificar con la definición S1 de *skolem2* nos permitirá obtener :

M12: skolem2(exists(y,p(y)&~r(y,x))#k(x), G1,[x|Us],[],Ls,f)

que unificando con S2 nos dará las dos submetas siguientes :

M13: skolem2(exists(y,p(y)&~r(y,x)), G2,[x|Us],[],Ls1,f),
skolem2(k(x), G3,[x|Us],[],Ls2,f)

La primera submeta de M13 unificará con la segunda regla S2 de la definición generando con esto la meta

M14: skolem2(p(y)&~r(y,x), G4,[x|Us],[y|Es],Ls1,f)

la cual unificará Ls1 con la lista []. Al skolemizar obtendremos que G2 = p(f1(x) & ~r(f1(x),x) al sustituir la variable y por f1(x). Ahora solamente queda por resolver la segunda submeta de M13 :

skolem2(k(x), G3,[x|Us],[],Ls2,f)

la cual unificará con la última regla S8 de la definición de *skolem2* poniendo fin a la recursión. Entonces la fórmula resultante F3 será :

F3 : all(x, p(f1(x))&~r(f1(x),x)) # k(x)) (III.4)

III.2.4 La Skolemización con el Método 3 (*skolem3*).

El proceso, en este caso, estará dividido en dos partes :

a) En la primera simplemente eliminaremos los cuantificadores vacuos.
b) En la segunda parte llevaremos a cabo la skolemización.

El predicado principal será nuevamente pre_skolem(F2,F3) en donde F3 es el resultado de skolemizar la fórmula F2. En este caso *pre_skolem* llamará al predicado *vacuos* para eliminar los cuantificadores vacuos de la fórmula, llamará a *lee_funcion* para leer el símbolo funcional con el que generaremos las funciones de skolem y a *skolem3* para llevar a cabo la skolemización :

```
pre_skolem(F2,F3) :- vacuos(F2,Fa),
                     lee_funcion(S),
                     skolem3(Fa,F3,S).
```

Las definiciones de los predicados *vacuos* y *lee_funcion* pueden ser revisadas en el anexo 1; aquí describiremos solamente el predicado *skolem3*.

El predicado skolem3(Fa,F3,S) define F3 como el resultado de skolemizar la fórmula Fa. S es el símbolo definido para generar las funciones de skolem. Recordemos que Fa es una variante de F2 en la cual hemos eliminado los cuantificadores vacuos. Si el valor de S en *skolem3* es, por ejemplo, la constante f los nombres de las funciones de skolem que sustituirán a las variables acotadas

existencialmente en Fa serán f_1, f_2, etc. y los argumentos de éstas serán las variables libres que se encuentren bajo el alcance del cuantificador existencial que vamos a skolemizar. Por ejemplo, si Fa es ∃x(∃y P(x,y,w)) y si S es el símbolo f, entonces, F3 será P(f1(w),f2(w),w).

La idea intuitiva de *skolem3* es la siguiente: recorremos la fórmula que vamos a skolemizar de izquierda a derecha y en el momento en el que nos encontremos un cuantificador existencial, obtenemos las variables libres que se encuentren bajo el alcance de éste. Los argumentos de la función de skolem serán estas variables libres y la función sustituirá todas las ocurrencias de la variable cuantificada existencialmente dentro del alcance del cuantificador. Eliminamos el cuantificador existencial y seguimos recorriendo la fórmula repitiendo el proceso antes descrito cada vez que encontremos un cuantificador existencial.

La fórmula Fa puede tener las mismas formas que para *skolem2*, pero en esta ocasión no nos interesa separar los casos de fórmula atómica y negación de atómica por lo que la definición de *skolem3* es la siguiente :

```
S'1: skolem3(all(X,P),all(X,P1),S) :-
                                !,
                                skolem3(P,P1,S).
S'2: skolem3(exists(X,P),P2,S) :-
                                !,
                                obten_vars(P, Ls, [X], S),
                                gensim(S,Nom),
                                Sk =.. [Nom|Ls],
                                subst(X,Sk,P,P1),
                                skolem3(P1,P2,S).
S'3: skolem3(P & Q,P1 & Q1,S) :-
                                !,
                                skolem3(P,P1,S),
                                skolem3(Q,Q1,S).
S'4: skolem3(P # Q,P1 # Q1,S) :-
                                !,
                                skolem3(P,P1,S),
                                skolem3(Q,Q1,S).
S'5: skolem3(P,P,S).
```

Como para *skolem2* comenzaremos por explicar el caso relevante, cuando la fórmula Fa es del tipo exists(x,P). En este caso el predicado obten_vars(P,Ls,As,S) genera en Ls y As respectivamente las listas de variables libres y acotadas de P. El nombre Nom de la función de skolem se obtiene con el predicado gensim(S,Nom) a partir del símbolo funcional S; el predicado univ (=..) genera la función de skolem Sk a partir de Ls y Nom. Subst(x,Sk,P,P1) sustituye cada ocurrencia de la variable x por Sk en P obteniendo la fórmula skolemizada P1. Por último lo que hacemos es llamar recursivamente a *skolem3* con la fórmula P1 ya que ésta puede contener nuevamente en su interior cuantificadores existenciales que hay que eliminar.

Veamos ahora los casos restantes. Si la fórmula Fa está cuantificada universalmente sobre la variable x, lo único que hay que hacer es llamar a *skolem3* para skolemizar la fórmula P que se encuentra bajo el alcance del cuantificador all. Si la fórmula Fa es una conjunción de la forma P & Q, *skolem3* es llamado para skolemizar P obteniendo P1 y para skolemizar Q obteniendo Q1. El tratamiento de una disyunción es totalmente análogo al caso de una conjunción. El caso trivial es cuando P es una fórmula atómica o la negación de una atómica, en cuyo caso no hay nada que skolemizar.

Para terminar con el funcionamiento de *skolem3*, sigamos el mismo ejemplo que en la sección anterior aplicando el predicado *skolem3* a la fórmula III.3 de la sección III.2.2 que es la fórmula resultante de introducir las negaciones en la fórmula III.2. La primera llamada a *skolem3* suponiendo que f es el símbolo funcional con el que se generarán las funciones de skolem, está dada por la siguiente submeta :

M'11: skolem3(all(x, exists(y,p(y)&~r(y,x)) # k(x)), F3,f)

este predicado generará la siguiente submeta :

M'12: skolem3(exists(y, p(y)&~r(y,x)) # k(x), G1,f)

que al unificar con S'4 generará a su vez dos nuevas submetas :

M'13: skolem3(exists(y,p(y)&~r(y,x)), G2,f), skolem3(k(x), G3,f)

La primera submeta de M'13 unificará con la segunda regla S'2 de la definición de *skolem3*, dando lugar a la submeta :

M'14: obten_vars(p(y) & ~r(y,x),Ls,[y],f), skolem3(k(x),G3,f)

obten_vars regresará una Ls=[x], por lo que al skolemizar obtendremos una G2 igual a :

$$p(f_1(x)) \ \& \ \sim r(f_1(x),x)$$

al sustituir la variable y por $f_1(x)$. Ahora queda por resolver la segunda submeta de M'13:

skolem3(k(x),G3,f)

la cual unificará con la última definición S'5 de *skolem3* dando lugar a la misma fórmula resultante F3 que con el método 2 (III.4) :

F3: all(x, p(f_1(x)) & ~r(f_1(x),x)) # k(x))

Ahora analicemos conjuntamente los dos métodos de skolemización. En ambos métodos el recorrido de la fórmula para generar las funciones de skolem se lleva a cabo de izquierda a

derecha sin embargo existen diferencias en la forma de la realización de *skolem2* y *skolem3*. La diferencia fundamental está en el momento de la sustitución de las funciones de skolem :

- Para *skolem2*, la llamada recursiva se hace antes de sustituir la variable del cuantificador existencial "actual", llevándose a cabo la sustitución real de las variables de derecha a izquierda.
- Para *skolem3*, primero sustituimos la variable del cuantificador existencial actual y después llamammos recursivamente a *skolem3* para eliminar los demás cuantificadoras existenciales si es que existen, por lo que las sustituciones se llevan a cabo realmente de izquierda a derecha.

El momento en que las sustituciones se llevan a cabo en ambas realizaciones no afecta a las funciones de skolem porque éstas fueron determinadas siempre de izquierda a derecha. Sin embargo la realización de *skolem3* se vuelve más ineficiente :

Una vez que se encuentra el primer existencial, para determinar las variables que van a intervenir en la función de skolem es necesario completar el recorrido de la fórmula y después regresar hacia la izquierda nuevamente para llevar a cabo la sustitución. El proceso se repite para todos los existenciales que se encuentren a la derecha por lo que la fórmula es recorrida tantas veces como existenciales se encuentren en su interior. Se han hecho pruebas con fórmulas que confirman que la implantación de *skolem2* es más eficiente que la de *skolem3*. Sin embargo, esto no significa que el método 2 sea en general más interesante ya que no siempre genera mejores funciones de skolem que el método 3.

III.2.5 Paso a Forma Clausular (*fcl*).

Finalmete quedaría por revisar el paso de forma normal de skolem a forma clausular. Esto se realiza en tres pasos :

a) Eliminar los cuantificadores universales.
b) Pasar la fórmula a forma normal conjuntiva.
c) Generar la lista de cláusulas que conforman la conjunción.

La implantación de estos tres pasos está dada por la definición del predicado fcl(F3,Cls) en donde F3 es una fórmula ya skolemizada y Cls es la lista de cláusulas resultante de los tres pasos anteriores. El predicado se define como sigue :

```
fcl(F3,Cls) :- univ_fuera(F3,Fa1),
               fnc(Fa1,Fa2),
               gen_cls(Fa2,Cls,[]).
```

Cada cláusula de la lista Cls será representada como una pareja de listas cl(L,Ln) donde L es la lista de literales no negadas de la cláusula y Ln es la lista de literales negadas sin sus

negaciones. El predicado univ_fuera(F3,Fa1), como su nombre lo indica, es el que se encarga de eliminar los cuantificadores universales de la fórmula F3 generando una nueva fórmula Fa1. Su implantación es similar a la del predicado *eli_imp* que se encuentra en la sección III.2.1, por lo que puede ser revisado en el anexo 1. Aquí solamente discutiremos el funcionamiento de los predicados *fnc* y *gen_cls*.

El predicado fnc(Fa1,Fa2) permite pasar la fórmula Fa1 a su forma normal conjuntiva Fa2 haciendo uso de la siguiente equivalencia lógica :

$$P \# (Q \& R) \equiv (P \# Q) \& (P \# R)$$

que queda expresada con el subpredicado fnc1(F,F1), en donde si F es una fórmula del tipo P # (Q & R), F1 será su equivalente : (P # Q) & (P # R).

Recordemos que la fórmula Fa1 que será transformada a forma normal conjuntiva sólo contiene ya conectivos & y #, por lo que sus posibles formas son : P # Q, P & Q y P. Entonces, la definición de *fnc* se expresa por casos de la siguiente manera:

```
F1: fnc(P & Q, P1 & Q1) :-
                    !,
                    fnc(P,P1),
                    fnc(Q,Q1).
F2: fnc(P # Q, R) :-
                    !,
                    fnc(P,P1),
                    fnc(Q,Q1),
                    fnc1(P1 # Q1,R).
F3: fnc(P, P).
```

Veamos ahora como trabaja el predicado *fnc*. Si F es una conjunción de la forma P & Q, utilizaremos la primera regla F1 de la definición. El resultado de pasar F a su forma normal conjuntiva será P1 & Q1, en donde P1 y Q1 serán los resultados de pasar P y Q a forma normal conjuntiva. Si F es una fórmula con la forma P # Q, utilizaremos la segunda regla F2 de la definición llamando recursivamente a *fnc* para pasar P y Q a su forma normal conjuntiva obteniendo como resultado P1 y Q1 respectivamete. Después tomamos la disyunción P1 # Q1 y le aplicamos la equivalencia lógica definida con *fnc1* para obtener una fórmula en forma normal conjuntiva. Por último, tenemos que la forma normal conjuntiva de una fórmula atómica es la misma fórmula atómica.

Ahora analicemos el procedimiento *fnc1*. Este procedimiento consta de tres reglas, ya que debemos de considerar que la fórmula a la que le vamos a aplicar la equivalencia, puede tener cualquiera de las tres formas siguientes :

```
(P & Q) # R
P # (Q & R)
    P
```

y se define como sigue :

```
FN1: fnc1( (P & Q) # R, P1 & Q1) :-
                                  !,
                                  fnc(P # R,P1),
                                  fnc(Q # R,Q1).
FN2: fnc1( P # (Q & R), P1 & Q1) :-
                                  !,
                                  fnc(P # Q,P1),
                                  fnc(P # R,Q1).
FN3: fnc1(P, P).
```

Las dos primeras reglas corresponden directamente a la equivalencia lógica que distribuye la disyunción, por la izquierda y por la derecha, y la tercera regla determina el fin de la recursión para fórmulas atómicas.

Veamos que sucede al transformar a forma normal conjuntiva la fórmula

$$Fa1:\ (p(f_1(x)) \ \&\ \sim r(f_1(x),x)) \ \#\ k(x) \qquad \text{(III.5)}$$

que es el resultado de haber eliminado los cuantificadores universales de la fórmula F3 ya skolemizada (III.4). La aplicación del predicado *fnc* a Fa1 generará la submeta :

```
M15:    fnc( (p(f1(x)) & ~r(f1(x),x))  #  k(x), Fa2)
```

que al unificar con la segunda regla F2 de la definición de *fnc* generará tres submetas :

```
M16:    fnc( (p(f1(x)) & ~r(f1(x),x)), G1),
        fnc(k(x), G2),
        fnc1(G1 # G2, Fa2).
```

las dos primeras metas regresarán respectivamente :

$$G1 = (p(f_1(x)) \ \&\ \sim r(f_1(x),x)) \quad y \quad G2 = k(x)$$

por lo que el predicado fnc1(G1 # G2, Fa2) unificará con la primera regla FN1 de la definición de *fnc1* y generará a su vez dos submetas :

```
M17:    fnc(p(f1(x)) # k(x), G3),
        fnc(~r(f1(x),x) # k(x), G4)
```

que regresan los valores :

$$G3 = p(f_1(x)) \ \#\ k(x) \quad y \quad G4 = \sim r(f_1(x),x) \ \#\ k(x)$$

por lo que finalmente Fa2 tendrá la forma :

$$Fa2:\ (p(f_1(x)) \ \#\ k(x)) \ \&\ (\sim r(f_1(x),x) \ \#\ k(x)) \qquad \text{(III.6)}$$

Por último, describiremos como obtenemos con *gen_cls* la lista de cláusulas Cls, a partir de la forma normal conjuntiva Fa2. Recordemos que la fórmula con la cual trabajaremos es una forma normal conjuntiva D1 &.....& Dn en donde cada Di = A1 # A2 #...# Al # ~B1 # ~B2 #...# ~Bm con i=1..n.

El predicado gen_cls(Fa2,Cls,Laux) se interpreta de la siguiente manera: Fa2 es una fórmula en forma normal conjuntiva y Cls es la lista de cláusulas obtenida a partir de Fa2 y tiene la forma [cl(L,Ln)|Resto] en donde L es a su vez la lista de literales no negadas de la cláusula y Ln la lista de literales negadas sin sus negaciones. En un principio la lista auxiliar Laux es vacía.

Las cláusulas generadas durante este proceso tendrán las siguientes características : 1) una misma literal no aparecerá dos veces en la misma cláusula y 2) si una cláusula contiene una fórmula atómica y la negación de ésta, la cláusula no se agregará a la lista que estamos generando ya que si C es una cláusula de la forma

C = p # q # ~r # ~p

que contiene a p y a ~p (contiene una fórmula válida p V ~p), el valor de verdad de la cláusula C será siempre verdadero. Como esta cláusula C siempre es verdadera, el valor de verdad de la forma normal conjuntiva a la que pertenece, no depende de ella porque tenemos que :

D1 &....& Dn & C ≡ D1 &....& Dn

El procedimiento gen_cls esta compuesto por tres reglas :

```
G1: gen_cls(P & Q, C1, C2) :-
                           !,
                           gen_cls(P,C1,C3),
                           gen_cls(Q,C3,C2).
G2: gen_cls(P, [cl(L,Ln)|Cs], Cs) :-
                           clasif_lit(P,L,[],Ln,[]),
                           !.
G3: gen_cls(P, C, C).
```

Nótese que la lista, Cls = [cl(L,Ln), cl(L,Ln),....,Cs], que vamos construyendo termina con una variable Cs que se regresa como tercer argumento de G2. Ello permite que las otras reglas de *gen_cls* agreguen nuevas cláusulas al final de la lista Cls.

La primera regla G1 de *gen_cls* se encarga de separar la cláusula de más a la izquierda de la conjunción de cláusulas y se llama dos veces recursivamente : la primera vez con la primera cláusula P y la segunda vez con el resto de la forma normal conjuntiva Q.

La segunda regla G2 de *gen_cls* va construyendo la lista de cláusulas con ayuda del predicado clasif_lit(P,L,Laux1,Ln,Laux2) quien separa las disyunciones de la cláusula P y construye las listas L y Ln recursivamente. Laux1 y Laux2 son listas auxiliares

para la construcción. El predicado *clasif_lit* también elimina las cláusulas siempre verdaderas y las literales redundantes.

La tercera regla G3 de *gen_cls* sirve simplemente para que una cláusula en la que ocurren una fórmula atómica y su negación unifique con ella y el proceso pueda continuar sin agregarla a la lista.

Veamos ahora que sucede con la forma normal conjuntiva Fa2 de la fórmula que hemos venido analizando (III.6). La llamada inicial será :

M18: gen_cls((p(f1(x))#k(x)) & (~r(f1(x),x)#k(x)), Cls, [])

la cual generará

M19: gen_cls(p(f1(x)) # k(x), Cls, G1),
gen_cls(~r(f1(x),x) # k(x), G1, [])

La primera llamada unificará con la segunda regla G2 de *gen_cls* dando lugar a una lista Cls de la forma:

[cl([p(f1(x)), k(x)], []), G1]

La segunda llamada unificará también con la segunda regla G2 con lo que tendremos que G1 es igual a :

[cl([k(x)], [r(f1(x),x)]), []].

y finalmente obtendremos la lista

Cls: [cl([p(f1(x)),k(x)], []), cl([k(x)], [r(f1(x),x)])].

con lo cual terminamos de analizar el predicado *traduce*.

Por último, ejemplificaremos todos juntos, los cuatro pasos del predicado *traduce* con la fórmula que hemos venido analizando. La llamada inicial será :

traduce(all(x, all(y, p(y)→ r(y,x)) → k(x)), Cls)

el conjunto de transformaciones que se llevan a cabo durante el proceso son:

F1: all(x,~all(y,~p(y) # r(y,x)) # k(x)) — al salir de eli_imp.

F2: all(x,exists(y,p(y) & ~r(y,x)) # k(x)) — al salir de neg_dentro.

F3: all(x,(p(f1(x)) & ~r(f1(x),x)) # k(x)) — al salir de pre_skolem.

Cls: [cl([p(f1(x)),k(x)],[]),cl([k(x)],[r(f1(x),x)])]. — al salir de fcl.

CAPITULO IV.

Estructura Global de los Programas.

En el capítulo anterior describimos el funcionamiento del predicado *traduce*, que puede considerarse como el predicado más importante del programa, ya que con él, definimos los distintos mecanismos para la traducción de una fórmula de primer orden a su forma clausular. Sin embargo, este predicado está inmerso dentro de una estructura más general que será el objeto de análisis de este capítulo.

El programa puede, además de traducir una fórmula de primer orden a su forma clausular, generar a partir de ésta última un conjunto de cláusulas PROLOG o bien parar el proceso de traducción después de la skolemización e imprimir la forma normal de skolem de la fórmula dada. Un menú de opciones permite escoger el tipo de acción que desee hacerse. El predicado principal de la estructura que analizaremos será entonces, *menu*. Este despliega simplemente en la pantalla un menú de 3 opciones : 1) *Clausulas*, con la cual transformamos una fórmula a forma clausular 2) *Fns*, con la cual transformamos una fórmula a forma normal de skolem 3) *Ayuda*, con la cual obtenemos información sobre las dos opciones anteriores. En la primera parte del capítulo, describiremos los predicados centrales de las primeras dos opciones. El funcionamiento de la opción *Ayuda* al igual que el funcionamiento detallado del menú será descrito en el próximo capítulo.

Dentro de la estructura general, existe una parte que define los conectivos lógicos de manera que éstos puedan ser aceptados por PROLOG; en una segunda parte del capítulo hablaremos de esta definición.

IV.1 Predicados Centrales del Programa.

El predicado *traduce* se encuentra inmerso dentro del programa en uno de caracter más general, el predicado *ejecuta*, quien prepara primero la fórmula y llama a *traduce* para efectuar la transformación a forma clausular e imprime después la forma clausular obtenida. El predicado *ejecuta* pude ser visto como el predicado central de la opción del programa que transforma a forma clausular. Paralelamente a *ejecuta*, existe el predicado *ejecuta_fns*, que es el predicado central de la opción que transforma a forma normal de skolem y que será descrito más adelante.

IV.1.1 El predicado ejecuta.

A continuación daremos la definición del predicado *ejecuta*, explicando el funcionamiento de cada una de sus partes.

```
ejecuta :- datos(Fa,Var),
           renombra(Fa,F,[],Var),
           traduce(F,Cls),
           imprime(Cls),
           opcion.
```

Los pasos que lleva a cabo este predicado son los siguientes:

1) Con datos(Fa,Var), se pregunta al usuario cual es la fórmula Fa que desea traducir y el símbolo Var para renombrar las variables acotadas de Fa.

2) Con renombra(Fa,F,L,Var), las variables acotadas de Fa son renombradas, utilizando el símbolo Var y una lista auxiliar L de variables de Fa, generada por el mismo *renombra*. La fórmula resultante es F.

3) Con traduce(F,Cls), se transforma F a su forma clausular dejando en la variable Cls la lista de listas que conforman las cláusulas de F.

4) Con imprime(Cls), se despliegan en forma de implicación cada una de las listas que conforman la lista Cls, mostrando así la forma clausular obtenida a partir de la fórmula original Fa.

5) Con *opcion*, se despliegan en la pantalla una serie de instrucciones que le indicarán al usuario que hacer para transformar la forma clausular obtenida a cláusulas PROLOG.

Los predicados *datos* y *opcion* pueden revisarse en el anexo 3 y no los analizaremos aquí por ser muy simples . Sólo analizaremos en detalle los predicados *renombra* e *imprime*.

IV.1.1.1 *El Renombramiento de Variables Acotadas.*

Se vió en el capitulo I, que para traducir una fórmula a su forma clausular necesitamos que la fórmula sea rectificada, i.e. que la fórmula :i) no contenga dos cuantificadores que acoten ocurrencias de la misma variable, ii) tampoco contenga ocurrencias libres y acotadas de una misma variable y iii) no contenga cuantificadores vacuos.

Los cuantificadores vacuos se eliminan dentro del predicado *traduce*, mientras que las condiciones i) e ii) se llevan a cabo dentro de *renombra*. El cambio de nombre de las variables acotadas no afecta el significado de la fórmula ya que ∀x F(x) o ∃x F(x) son respectivamente equivalentes a ∀y F(y) o ∃y F(y), por ser x sólo un testimonio de todo valor posible o de un valor específico de la variable.

El predicado renombra(Fa,F,L,Var) recibe la fórmula leída en el argumento Fa y regresa en F la fórmula renombrada. Para ello recorre la fórmula generando en la lista L una secuencia de parejas de la forma [y,xi], en donde cada y es una variable cuantificada de Fa, y cada xi es la variable que sustituirá en F todas las ocurrencias de y que se encuentren bajo el alcance del cuantificador analizado.

La idea intuitiva de este predicado es ir recorriendo Fa de izquierda a derecha para detectar cada cuantificador dentro de la fórmula y la lista de variables acotadas por él. De esta forma, se genera una lista de parejas L para cada subfórmula cuantificada de F, en donde el elemento izquierdo de la pareja es una variable acotada y el derecho su nuevo nombre. Cuando se llega a una fórmula atómica se sustituyen en ella las variables que aparecen como argumentos izquierdos de las listas, por las que aparecen como argumentos derechos. Por ejemplo, supongamos que cuando se encuentra la fórmula atómica

p(x,f(w,x),z)

la lista L de parejas es [[x,x1], [z,x2], [w,x3]], en donde x, z y w son las variables cuantificadas de la fórmula cuya subfórmula es p, y x1, x2, x3 sus nuevos nombres respectivamente . Entonces la variable x será sustituida en p(x,f(y,x),z) por x1, la variable z por x2 y la variable w por x3, obteniendo una nueva variante:

p(x1,f(x3,x1),x2).

La definición del predicado *renombra* aparece en el programa como sigue:

```
R1: renombra(all(X,P),all(X1,P1),L,Var) :-
                                !,
                                gensim(Var,X1),
                                renombra(P,P1,[[X,X1]|L],Var).
R2: renombra(exists(X,P),exists(X1,P1),L,Var) :-
                                !,
                                gensim(Var,X1),
                                renombra(P,P1,[[X,X1]|L],Var).
R3: renombra(P (↔ ⇒ & /) Q, P1 (↔ ⇒ & /) Q1, L, Var) :-

                                !,
                                renombra(P,P1,L,Var),
                                renombra(Q,Q1,L,Var).
R4: renombra(~P,~P1,L,Var) :-
                                !,
                                renombra(P,P1,L,Var).
R5: renombra(P,P1,L,Var) :-
                                substituye(P,P1,L).
```

Ahora analizaremos como funciona *renombra*, para la fórmula all(y,P) que corresponde a la primera regla R1 de la definición :

a) Con el predicado gensim(Var,Sim) se crea en *Sim* el símbolo para sustituir la variable cuantificada *y*. Este se construye concatenando al valor de Var, que es el nombre de la variable dada por el usuario para renombrar, con un entero positivo que varía en cada ejecución de *gensim*. Por ejemplo, si la variable

Var dada por el usuario es z, entonces gensim(z,Sim) dará a Sim el valor z1 en la primera llamada a *gensim*.

b) El predicado *renombra* aparece de nuevo para analizar ahora la fórmula P dentro de all(X,P), ya que ésta puede contener a su vez subfórmulas cuantificadas cuyas variables habrá que renombrar. Pero en esta nueva llamada recursiva a *renombra*, la pareja [X, X1] ha sido agregada ya a la lista L del primer *renombra*.

Por ejemplo, supongamos que la fórmula F dada es all(y,P) y que z es el valor de la variable Var, entonces el predicado renombra(all(y,P), all(X1,P1), L, z) generaría: a) la llamada a gensim(z, X1) que, si es la primera vez que se ejecuta, unificaría X1 con z1 y b) generaría la segunda llamada : renombra(P, P1, [[y,z1]|L], z), en donde la pareja (y,z1) ha sido agregada a la lista L de la llamada inicial. Nótese que cuando *renombra* es llamado por primera vez desde *ejecuta*, el valor de L es la lista vacía y ésta se va incrementando con cada llamada recursiva de *renombra*. El caso en el que la fórmula tenga la forma exists(y,P) es análogo al que se acaba de describir, por lo que pasaremos a los siguientes.

Si la fórmula es de la forma : P ↔ Q, P → Q, P & Q o P # Q, utilizaremos la definición R3 del predicado *renombra*. En todos estos casos no hay un cuantificador externo y los conectivos son binarios, por lo que simplemente haremos dos llamadas recursivas al predicado *renombra*: renombra(P,P1,L,Var) y renombra(Q,Q1,L,Var) para analizar las fórmulas P y Q, quienes a su vez pueden contener subfórmulas cuantificadas cuyas variables será necesario renombrar.

Si la fórmula F tiene la forma ~P, utilizamos la definición R4 que solamente llama recursivamente a renombra(P,P1,L,Var) para analizar a P.

Si la fórmula F es una fórmula atómica P, se utiliza la última definición R5. Esta llama al predicado substituye(P,P1,L) para hacer efectiva la sustitución de variables acotadas de la fórmula atómica P por sus nuevos nombres, generando así la fórmula P1. Recordemos que la lista L contiene en este momento todas las parejas de la forma [X,X1] que indican las sustituciones a efectuar en P.

Al término de la aplicación del predicado *renombra* a la fórmula Fa, se ha obtenido una nueva fórmula F que ya no contiene ocurrencias libres y acotadas de la misma variable, ni tampoco contiene cuantificadores distintos que acoten variables iguales; por lo que la fórmula está lista para ser traducida.

IV.1.1.2 *La impresión de las Cláusulas.*

Para poder describir el procedimiento que imprime las cláusulas será necesario discutir primero la notación que

utilizaremos para ello. Esta notación se eligió en estrecha relación con el lenguaje PROLOG.

PROLOG puede verse como un subconjunto del lenguaje de la lógica de primer orden, ya que solamente trabaja con algunas de las formas clausulares conocidas bajo el nombre de cláusulas de Horn. Estas se definen como cláusulas que tienen, a lo más, una literal no negada. Es decir algo de la forma :

$$A \vee \sim B1 \vee \ldots \vee \sim Bm \ .$$

Haciendo uso de equivalencias lógicas como :

$$\begin{aligned} A \rightarrow B \quad &\equiv \quad \sim A \vee B \\ \sim(A \ \& \ B) \quad &\equiv \quad \sim A \vee \sim B \end{aligned}$$

se puede demostrar que una cláusula de Horn, como la arriba definida, es lógicamente equivalente a :

$$A \vee \sim(B1 \ \& \ \ldots \ \& \ Bm)$$

que a su vez es lógicamente equivalente a :

$$A \leftarrow B1 \ \& \ \ldots \ \& \ Bm \qquad (IV.1)$$

Esta será la notación usada para la impresión de formas clausulares en el programa. La diferencia con PROLOG es que sustituye el símbolo de conjunción tradicional '&' por la ',' y el símbolo '<- ' por ':-' quedando la fórmula anterior representada en PROLOG de la siguiente manera :

$$A :- \ B1 \ , \ldots \ , Bm \qquad (IV.2)$$

Cuando se traduzca de forma clausular a cláusulas PROLOG esta será la notación usada.

Ahora bien, si consideramos que las cláusulas de Horn pueden tener, a lo más, una letra no negada A, la fórmula anterior podría tener sólo tres formas distintas :

1) Hechos :	A <-	(ninguna letra negada)
2) Reglas :	A <- B1,...,Bm	(una letra no negada y alguna no negada)
3) Metas :	<- B1,...,Bm	(ninguna letra no negada)

Los hechos son cláusulas sin antecedente, o sea la afirmación de un predicado. Las reglas tienen como consecuente un predicado no negado y como antecedente una conjunción de predicados no negados. Por último, la meta se interpreta como la negación de una conjunción de predicados no negados.

No olvidemos sin embargo, que las cláusulas de una forma clausular resultante de la traducción pueden no ser cláusulas de Horn, por lo que su forma puede ser más general que IV.1, digamos:

$$A1 \vee \ldots \vee Ak \leftarrow B1, \ldots, Bm$$

Para ser consistentes con los conectivos definidos en el programa habria que sustituir el simbolo 'V' por el '#' y el simbolo ',' por '&'. Con lo que la notación general para la impresión de la forma clausular se veria así :

$$A1 \# \ldots \# Ak \leftarrow B1 \& \ldots \& Bm \qquad (IV.3)$$

Es importante mostrar ahora que la fórmula IV.3 es realmente equivalente a una cláusula cualquiera, no necesariamente de Horn. Por ejemplo, si tenemos la cláusula :

$$p_1 \# \neg q_3 \# \neg q_1 \# p_2 \# p_3 \# \neg q_2$$

debido a la conmutatividad de la disyunción, ésta se puede escribir como :

$$p_1 \# p_2 \# p_3 \# \neg q_1 \# \neg q_2 \# \neg q_3.$$

En general cualquier cláusula se puede "ordenar" de manera que tengamos primero las literales no negadas y después las negadas. Entonces, una cláusula como :

$$A1 \# A2 \# \ldots \# Ak \# \neg B1 \# \neg B2 \# \ldots \# \neg Bm$$

es lógicamente equivalente a :

$$A1 \# A2 \# \ldots \# Ak \# \neg(B1 \& B2 \& \ldots \& Bm)$$

ya que : $(\neg(A \& B) \equiv \neg A \# \neg B)$ y ésta a su vez utilizando $(\neg(A \# B) \equiv A \rightarrow B)$, es lógicamente equivalente a :

$$\boxed{A1 \# A2 \# \ldots \# Ak \leftarrow B1 \& B2 \& \ldots \& Bm} \qquad (IV.4)$$

Una vez definida la notación clausular que utilizaremos para la impresión, analizaremos ahora el predicado **imprime(Cls)**, en donde Cls es la lista de parejas de la forma cl(L,Ln), generada por el último paso del predicado *traduce* (capitulo II). La lista L representa la lista de literales no negadas de una cláusula y Ln la lista de literales negadas de la misma. Este predicado arma cláusula por cláusula de la lista Cls y se define de la siguiente manera :

```
imprime(Cls) :-
        print('>La formula en forma clausular es  la  siguiente: '),
        nl,
        resultado(Cls),
        archiva(Cls).
```

El predicado imprime([cl(L,Ln)|Resto]) llama al predicado intrinseco de PROLOG print(M) para escribir en la pantalla el mensaje M; a continuación llama a resultado([cl(L,Ln)|Resto]) para separar una a una las cláusulas cl(L,Ln) de la lista Cls e imprimirlas. El predicado resultado([cl(L,Ln)|Resto]) llama, para cada elemento cl(L,Ln) de la lista Cls, al predicado impclaus(L,Ln) quien escribe, con disyuncion(L), la disyunción de literales no negadas que formará el consecuente de la implicación y con, conjuncion(Ln), la conjunción de literales negadas sin sus negaciones que formará el antecedente de la cláusula. Supongamos que el predicado *imprime* es llamado para imprimir la lista Cls de dos elementos correspondiente a la forma clausular de la fórmula que analizamos durante todo el capitulo III (pag.64) :

```
imprime( [ cl([p(f1(x)),k(x)],[]), cl([k(x)],[r(f1(x),x)]) ] )
```

El primer elemento de la lista Cls tiene una pareja de listas de la forma L = [p(f1(x)),k(x)] y Ln = []. Entonces, impclaus([p(f1(x)),k(x)], []) llama a disyuncion([p(f1(x)), k(x)]) y a conjuncion([]) para escribir la primera cláusula IV.5. El segundo elemento de Cls tiene una lista L = [k(x)] y una lista Ln = [r(f1(x),x)], por lo que el predicado *impclaus* llamará a disyuncion([k(x)]) y conjuncion([r(f1(x),x)]) para escribir la segunda cláusula IV.6.

```
p(f1(x)) # k(x) .                          (IV.5)
k(x)  <-  r(f1(x),x).                      (IV.6)
```

Por falta de espacio en el stack del intérprete, el conjunto de cláusulas PROLOG correspondiente a una forma clausular, no puede ser escrito directamente en la pantalla; con el propósito de que el usuario pueda imprimirlo posteriormente, si así lo desea, el predicado archiva(Cls) escribe en el archivo *clausula* la siguiente meta :

```
imprime_p(Cls)?
```

que al ser activada, escribirá la lista de cláusulas Cls en forma de cláusulas de PROLOG. La definición del predicado *archiva* puede ser revisada en el anexo 3. A continuación describiremos la manera como se lleva a cabo la transformación de cláusulas a cláusulas PROLOG.

IV.1.1.3 *Paso a Clausulas PROLOG.*

El predicado *ejecuta*, llama por último al predicado *opcion*; éste desplegará en pantalla una serie de instrucciones que indicarán al usuario como generar el conjunto de las distintas

cláusulas PROLOG, a partir de la forma clausular obtenida. La definición del predicado *opcion* es muy simple y puede ser revisada en el anexo 3. Lo que el usuario debe hacer para generar el conjunto de cláusulas PROLOG es cargar a memoria el archivo *clausula* que contiene la meta imprime_p(Cls)?, quien al ser activada se encargará de imprimirlas. En el próximo capítulo explicaremos como se carga a memoria este archivo; ahora simplemente describiremos el predicado *imprime_p*, pero antes veremos como una cláusula cualquiera que contiene varias literales no negadas en el consecuente, puede ser escrita como una cláusula que solamente contiene una literal no negada en el consecuente.

Hemos visto que el predicado *ejecuta*, imprime las cláusulas en forma de implicación. Estas cláusulas pueden tener 3 formas :

<-- B1,...,Bn	El consecuente de la implicación no tiene elementos.
A <-- B1,...,Bn	El consecuente de la implicación tiene solamente un elemento.
A1 #..# Ak <-- B1,...,Bn	El consecuente de la implicación tiene dos o más elementos.

Si se quisiera hacer una analogía de estas cláusulas con las cláusulas de PROLOG, podríamos decir lo siguiente : la primera forma corresponde a metas de PROLOG, la segunda forma corresponde a reglas de PROLOG, mientras que la tercera forma no tiene un correspondiente en PROLOG. Evidentemente, no toda fórmula de primer orden tiene una cláusula de Horn correspondiente. Sin embargo, aún cuando la cláusula obtenida sea del tercer tipo (no de Horn), es posible, utilizando la negación extralógica de PROLOG, generar una cláusula que cumpla con las restricciones de Horn y cuyo significado siga siendo equivalente a la cláusula en cuestión. Lo que se hace es seleccionar una de las literales no negadas del consecuente y el resto de las literales no negadas se pasan como negadas a la conjunción que forma el antecedente.

Analicemos el siguiente razonamiento. Supongamos que se tiene la cláusula :

$$P \vee Q \vee \sim R \qquad \text{(IV.7)}$$

Por un lado,

$P \vee Q \vee \sim R$	$\equiv$	$\sim\sim P \vee Q \vee \sim R$	ya que $A \equiv \sim\sim A$
$\sim\sim P \vee Q \vee \sim R$	$\equiv$	$Q \vee \sim R \vee \sim\sim P$	por la conmutatividad de la disyunción.
$Q \vee \sim R \vee \sim\sim P$	$\equiv$	$Q \vee \sim(R \,\&\, \sim P)$	ya que $\sim A \vee \sim B \equiv \sim(A \,\&\, B)$
$Q \vee \sim(R \,\&\, \sim P)$	$\equiv$	$Q \leftarrow R \,\&\, \sim P$	ya que $A \vee \sim B \equiv A \leftarrow B$

cláusulas PROLOG, a partir de la forma clausular obtenida. La definición del predicado *opcion* es muy simple y puede ser revisada en el anexo 3. Lo que el usuario debe hacer para generar el conjunto de cláusulas PROLOG es cargar a memoria el archivo *clausula* que contiene la meta imprime_p(Cls)?, quien al ser activada se encargará de imprimirlas. En el próximo capítulo explicaremos como se carga a memoria este archivo; ahora simplemente describiremos el predicado *imprime_p*, pero antes veremos como una cláusula cualquiera que contiene varias literales no negadas en el consecuente, puede ser escrita como una cláusula que solamente contiene una literal no negada en el consecuente.

Hemos visto que el predicado *ejecuta*, imprime las cláusulas en forma de implicación. Estas cláusulas pueden tener 3 formas :

<-- B1,...,Bn	El consecuente de la implicación no tiene elementos.
A <-- B1,...,Bn	El consecuente de la implicación tiene solamente un elemento.
A1 #..# Ak <-- B1,...,Bn	El consecuente de la implicación tiene dos o más elementos.

Si se quisiera hacer una analogía de estas cláusulas con las cláusulas de PROLOG, podríamos decir lo siguiente : la primera forma corresponde a metas de PROLOG, la segunda forma corresponde a reglas de PROLOG, mientras que la tercera forma no tiene un correspondiente en PROLOG. Evidentemente, no toda fórmula de primer orden tiene una cláusula de Horn correspondiente. Sin embargo, aún cuando la cláusula obtenida sea del tercer tipo (no de Horn), es posible, utilizando la negación extralógica de PROLOG, generar una cláusula que cumpla con las restricciones de Horn y cuyo significado siga siendo equivalente a la cláusula en cuestión. Lo que se hace es seleccionar una de las literales no negadas del consecuente y el resto de las literales no negadas se pasan como negadas a la conjunción que forma el antecedente.

Analicemos el siguiente razonamiento. Supongamos que se tiene la cláusula :

$$P \vee Q \vee \sim R \qquad \text{(IV.7)}$$

Por un lado,

$P \vee Q \vee \sim R$	$\equiv$	$\sim\sim P \vee Q \vee \sim R$	ya que $A \equiv \sim\sim A$
$\sim\sim P \vee Q \vee \sim R$	$\equiv$	$Q \vee \sim R \vee \sim\sim P$	por la conmutatividad de la disyunción.
$Q \vee \sim R \vee \sim\sim P$	$\equiv$	$Q \vee \sim(R \,\&\, \sim P)$	ya que $\sim A \vee \sim B \equiv \sim(A \,\&\, B)$
$Q \vee \sim(R \,\&\, \sim P)$	$\equiv$	$Q \leftarrow R \,\&\, \sim P$	ya que $A \vee \sim B \equiv A \leftarrow B$

por lo cual P V Q V ~R es lógicamente equivalente a :

$$Q \leftarrow R \,\&\, \sim P \qquad (IV.8)$$

Por otro lado,

P V Q V ~R	≡	P V ~~Q V ~R	ya que A ≡ ~~A
P V ~~Q V ~R	≡	P V ~R V ~~Q	por la conmutatividad de la disyunción.
P V ~R V ~~Q	≡	P V ~(R & ~Q)	ya que ~A V ~B ≡ ~(A & B)
P V ~(R & ~Q)	≡	P <- R & ~Q	ya que A V ~B ≡ A <- B

por lo que P V Q V ~R es también lógicamente equivalente a :

$$P \leftarrow R \,\&\, \sim Q \qquad (IV.9)$$

En la lógica de primer orden las fórmulas IV.7 IV.8 y IV.9 son equivalentes, pero las cláusulas IV.8 y IV.9 no pertenecen al conjunto de cláusulas de Horn. Sin embargo, utilizando el predicado extralógico *not* de PROLOG, estas cláusulas podrían escribirse como :

$$Q \leftarrow R \,\&\, \text{not}(P) \qquad (IV.10.1)$$
$$P \leftarrow R \,\&\, \text{not}(Q) \qquad (IV.10.2)$$

por lo que alguna de las cláusulas de (IV.10) podría tomarse para generar un programa PROLOG para la fórmula (IV.7).

Nótese, sin embargo, que como el *not* de PROLOG no es lógicamente equivalente a la negación lógica, las cláusulas (IV.10) no son exactamente equivalentes a (IV.7). Por otro lado, hay que notar que el conjunto de inferencias posibles a partir de la cláusula IV.10.1 no es el mismo que el de la cláusula IV.10.2 ya que el consecuente de IV.10.1, y por lo tanto el predicado unificable con una meta, es Q, mientras que en IV.10.2 el consecuente es P. El programa generará ambas cláusulas a partir de la cláusula IV.7 con el propósito de que el usuario pueda escoger la que mejor le convenga.

En general, si una cláusula resultante de la traducción tiene $A_1,\ldots,A_k$ como literales no negadas, es decir tiene la forma :

$$A1 \# A2 \# \ldots \# Ak \leftarrow B1 \,\&\ldots\&\, Bn \;. \qquad (IV.11)$$

nuestro programa generará un conjunto con k cláusulas PROLOG, en donde cada cláusula tendrá una sola de las Ai, como consecuente :

```
A1  <-- B1,...,Bn,not(A2),...., not(Ak)

A2  <-- B1,...,Bn,not(A1),not(A3),...,not(Ak)

.
.
.

Ak  <-- B1,...Bn,not(A1),...., not(Ak-1)
```

Si se quisiera escribir un programa en PROLOG que incluyera una cláusula correspondiente a la fórmula IV.11, se tendría que escoger una de las k cláusulas anteriores y es claro por lo que mencionamos anteriormente que el potencial de deducción de este programa no será el mismo que el de la cláusula IV.11.

Ahora pasaremos a describir el predicado *imprime_p*. Este transforma cada cláusula de la lista [cl(L,Ln) | Resto] generada por *traduce*, en cláusulas que solamente contienen una literal no negada en el consecuente, i.e., en las distintas cláusulas PROLOG correspondientes. El conjunto de las distintas cláusulas PROLOG podrá ser guardado en el disco bajo un nombre, dado como dato por el usuario. La definición de *imprime_p* es la siguiente :

```
imprime_p(Cls) :-
   resultado2(Cls),
   prin('Quieres guardar las clausulas PROLOG en un archivo '),
   print('(s/n) ? '),
   ratom(Res),
   guarda(Cls,Res),
   prin('Para entrar al menu principal, escribe la meta '),
   print('siguiente : menu? '),
   nl.
```

El predicado **imprime_p([cl(L,Ln) | Resto])** llama a **resultado2([cl(L,Ln) | Resto])** para separar una a una las cláusulas cl(L,Ln) de la lista Cls e imprimirlas. Recordemos que L es la lista de predicados no negados de la cláusula y Ln es la lista de los predicados que aparecen negados en la cláusula pero sin sus negaciones. El predicado *resultado2* se encarga de llamar a otros predicados auxiliares para que estos impriman las cláusulas; de estos predicados el que hace el trabajo principal es *impclaus2* que imprime para cada elemento a de L, la cláusula cuyo consecuente es a y cuyo antecedente esta formado por Ln y por la negación extralógica de todos los elementos de L, exceptuando a a. La definición de *impclaus2*, al igual que la de los demás predicados auxiliares, puede ser revisada en el anexo 3.

Después de imprimir el conjunto de cláusulas PROLOG, el predicado *imprime_p*, escribe en la pantalla el siguiente mensaje : Quieres guardar las clausulas PROLOG en un archivo (s/n) ? ; al que el usuario contestará con una s o una n dependiendo de lo que quiera hacer. La respuesta del usuario será leída con ratom(Res) y finalmente, dependiendo del valor de Res, el predicado

guarda(Cls,Res) guardará el conjunto en un archivo cuyo nombre fue dado como dato.

Los últimos predicados de la definición de *imprime_p*, prin(M) y print(M), son predicados intrínsecos de prolog-86 que se encargan de escribir el mensaje M en pantalla. La diferencia que existe entre ellos es que prin(M) escribe el mensaje M y se queda en la misma linea de impresión, mientras que print(M) escribe el mensaje y se cambia a la siguiente linea.

Para ilustrar el funcionamiento de *imprime_p*, veremos a continuación un ejemplo. Supongamos que se desea obtener el conjunto de cláusulas PROLOG correspondiente a la cláusula

$$p_1 \ \# \ p_2 \ \text{<-- } q_1, q_2, q_3.$$

que está representada por la lista :

$$cl([p_1, p_2], [q_1, q_2, q_3])$$

La llamada inicial a *imprime_p* que se encuentra escrita en el archivo *clausula* es :

$$\text{imprime\_p}([\ cl(\ [p_1, p_2], \ [q_1, q_2, q_3] \ ) \ ] \ )?$$

En este caso la lista de literales no negadas L, es $[p_1, p_2]$ y la lista de literales negadas sin sus negaciones es $[q_1, q_2, q_3]$. *Impclaus2* se encargará de imprimir para cada literal p_i de L, la cláusula cuyo consecuente sea p_i y cuyo antecedente sea Ln más la negación extralógica de cada p_j con $j \neq i$; i.e., *impclaus2* imprimirá el siguiente conjunto de cláusulas :

$$p_1 \ \text{:-} \ q_1, q_2, q_3, \text{not}(p_2).$$
$$p_2 \ \text{:-} \ q_1, q_2, q_3, \text{not}(p_1).$$

y el siguiente mensaje :

Quieres guardar el conjunto en un archivo (s/n) ?

el usuario contestará s o n dependiendo de lo que desee hacer.

A continuación analizaremos la opción del programa que transforma una fórmula a forma normal de skolem, cuya implantación esta dada por el predicado *ejecuta_fns*.

IV.1.2 Paso a Forma Normal de Skolem.

Hay ocasiones en las que puede ser útil tener la forma normal de skolem de una fórmula, en lugar de su forma clausular. En estos casos podemos utilizar la segunda opción del menú que transforma una fórmula a forma normal de skolem. El predicado central de esta opción es *ejecuta_fns* que se define de la siguiente manera :

```
ejecuta_fns  :- datos(Fa,Var),
                renombra(Fa,F,[],Var),
                fns(F,Fns),
                imp_fns(Fns).
                opcion_fns.
```

Al igual que el predicado *ejecuta*, *ejecuta_fns* utiliza los predicados datos(Fa,Var) y renombra(Fa,F,[],Var) para leer respectivamente los datos (la fórmula Fa en cuestión y el símbolo Var) y renombrar las variables con el símbolo leído en Var. Sin embargo, en esta ocasión en lugar del predicado *traduce*, se utiliza directamente el predicado fns(F1,Fns), quien lleva a cabo la traducción a forma normal de skolem. Finalmente, *ejecuta_fns* utiliza sus propios predicados *imprime_fns* y *opcion_fns* para la impresión de esta forma y para desplegar una serie de instrucciones que indicarán como traducir otra fórmula, ya sea a forma clausular o a forma normal de skolem. A continuación daremos las definiciones correpondientes a *fns* y a *imprime_fns*, la definición de *opcion_fns* es muy simple y puede ser revisada en el anexo 3.

```
fns(F1,Fns)  :-
             eli_imp(F1,F2),
             neg_dentro(F2,F3),
             pre_skolem(F3,F4),
             paso_a_fnc(F4,Fns).

imp_fns(F)  :-
            print('La formula en forma normal de skolem es : '),
            write(F).
```

El predicado *fns*, al igual que *traduce*, elimina las implicaciones de la fórmula con el predicado *eli_imp*, introduce las negaciones con el predicado *neg_dentro* y skolemiza con el predicado *pre_skolem*. Sin embargo, *fns* ya no pasa la fórmula a forma clausular, solamente la transforma a forma normal conjuntiva utilizando el predicado *paso_a_fnc*. Este predicado elimina los cuantificadores universales de la fórmula F4 y la transforma a forma normal conjuntiva. El resultado final queda en Fns y está listo para ser impreso, por lo que el predicado imp_fns(Fns) simplemente escribe con el predicado intrínseco de PROLOG write, la forma normal de skolem.

Por último, describiremos como definimos los conectivos de las fórmulas en el intérprete prolog-86.

IV.2 Tratamiento de los conectivos en las fórmulas.

Para que las fórmulas de primer orden pudieran ser reconocidas como secuencias de símbolos por el intérprete prolog-86 y para ahorrarnos el trabajo de checar la sintaxis de las mismas, fué necesario declarar los conectivos como operadores

dentro de PROLOG, utilizando para ello un predicado autocontenido: op. Al definir de esta manera los conectivos, además de quedar establecida su precedencia, su asociatividad y el hecho de ser infijos (como el #, &, → y ↔) o prefijos (como el ~), la sintaxis de las fórmulas que se traducen se checa automáticamente. El predicado op(P,E,N) tiene tres argumentos : Precedencia, Especificador y Nombre, los cuales describiremos en seguida:

a) Precedencia :
Se refiere a la clase de precedencia del operador con respecto a los otros y esta dada por un entero cuyo rango puede variar de 1 a 1200. Los números más chicos indican mayor precedencia que los números mayores. Solamente los paréntesis pueden alterar la definición de la precedencia. Por ejemplo, si suponemos que el '#' tiene mayor precedencia que el '→', la máquina leería la fórmula

p → q # r

como p implica que o bien q o bien r. Si queremos alterar esto, hacemos uso de los paréntesis y cambiamos la fórmula anterior por la siguiente:

(p → q) # r

que sería leída como o bien p implica q o bien r.

b) Especificador :
Para todo conectivo es necesario especificar dos características:

1. Si el conectivo es infijo o prefijo.
2. La asociatividad entre conectivos de la misma precedencia.

Ya que si en una fórmula ocurren dos conectivos con la misma precedencia y sin paréntesis que indiquen la operación que debe llevarse a cabo primero, se necesita una convención para decidir cual ejecutamos en primer lugar. El especificador es un símbolo que da la posición de los operadores y la asociatividad de los mismos. Si f es un conectivo y x1,x2 los argumentos:

. fx1 indica que el conectivo es prefijo.
. x1fx2 indica que el conectivo es infijo.

Los especificadores para operadores infijos serán :

xfx xfy yfx yfy

en donde f representa al operador y los símbolos x, y los argumentos. Se escogerá entre x y y para definir a los argumentos de acuerdo a la siguiente regla: Suponiendo que no existen paréntesis que nos indiquen

la asociatividad de la fórmula :

. una y indica que el argumento puede contener operadores de la misma precedencia o de menor precedencia que el operador principal f.

. una x indica que cualquier operador en el argumento debe de tener precedencia estrictamente menor que el operador principal.

Por ejemplo, si declaramos que el operador de la disyunción, #, tiene especificador xfy y tenemos la fórmula, p # q # r, existen para ella dos posibles interpretaciones: (p # q) # r y p # (q # r). La primera interpretación no sería válida ya que el primer argumento (p # q) contiene un operador de la misma precedencia que el # y esto contradice la presencia de x antes de f en xfy. Por lo que si el especificador es xfy, la interpretación hecha por prolog-86 de la fórmula p # q # r será p # (q # r). Así, un operador que tiene como especificador xfy, será asociativo por la derecha y uno que tiene como especificador yfx será asociativo por la izquierda. Los especificadores para los operadores prefijos serán :

fx fy

en donde la x y la y tienen el mismo significado que para los operadores infijos. Por ejemplo, supóngase que el operador de la negación ~ , está especificado con fx y consideramos la fórmula ~~p, esta fórmula no sería válida ya que la x indica que la fórmula afectada por la negación debe contener conectivos de precedencia estrictamente menor. Ahora, si suponemos que el operador tiene como especificador fy la fórmula ~~p si sería válida.

c) Nombre :
El nombre del operador será simplemente el símbolo que usaremos para denotarlo.

La precedencia para los conectivos lógicos será la misma que adoptamos en el capítulo I. Por lo que la fórmula :

$$\sim q(x) \ \# \ r(x) \Rightarrow p(x)$$

en realidad se interpreta como :

$$((\sim q(x)) \ \# \ r(x)) \Rightarrow p(x).$$

Además se hará la siguiente convención : los operadores #, &, $\Rightarrow$, $\Leftrightarrow$ serán asociativos por la derecha; i.e., cuando se tengan ocurrencias del mismo conectivo o de conectivos con la misma precedencia, se asociará de derecha a izquierda. Por ejemplo la fórmula :

$$p \rightarrow q \leftrightarrow s$$

será interpretada como:

$$(p \rightarrow (q \leftrightarrow s))$$

Basándose en la descripción anterior, la declaración de los conectivos se hizo mediante los siguientes predicados :

```
op(30,fy,~)

op(100,xfy,#)

op(100,xfy,&)

op(150,xfy,=>)

op(150,xfy,<=>)
```

Estas declaraciones se colocaron en un archivo externo de nombre conec.pro y tendrán que ser cargadas a memoria antes de utilizar el programa. Esta explicación forma parte del siguiente capítulo en el cual presentaremos un manual de operación del programa.

CAPITULO V.

Manual de Utilización de los Programas.

Este capítulo puede ser visto como el manual de operación del programa. En realidad, existen dos versiones del programa objeto de esta tesis : traduc2.pro y traduc3.pro; la primera versión skolemiza las fórmulas con el método 2 y la segunda las skolemiza con el método 3 descritos en el capítulo II . En la primera parte del capítulo explicaremos como están conformados los archivos traduc2.pro y traduc3.pro, describiremos como se carga el programa a memoria y el funcionamiento de cada una de las opciones del menú. En una segunda parte ilustraremos el funcionamiento del programa con algunos ejemplos.

V.1 Funcionamiento del programa.

El programa en cualquiera de sus dos versiones, está constituido por tres archivos : 1) el archivo que contiene las definiciones de los conectivos como operadores (conec.pro), 2) el archivo que contiene las definiciones de los predicados *ejecuta* y *ejecuta_fns* (t2.pro y t3.pro para las vesiones traduc2.pro y traduc3.pro respectivamente) y por último el archivo que contiene las definiciones de los menús (menus.pro). A continuación describiremos los pasos necesarios para entrar al intérprete prolog-86 y para cargar cualesquiera de las dos versiones del programa; para ilustrar el funcionamiento del mismo, utilizaremos el archivo traduc2.pro. La versión traduc3.pro funciona de manera idéntica.

Desde el sistema operativo de la máquina (MS.DOS), teniendo en el controlador A: el diskette con el intérprete prolog-86 y en el controlador B: el archivo traduc2.pro, se debe llamar al intérprete de prolog-86 incrementando el stack a 6000 posiciones :

```
A > prolog -s6000
```

Cuando aparezca el indicador de prolog-86, ":", teclear : load'b:traduc2.pro'! , esta instrucción carga el archivo traduc2.pro a memoria y éste a su vez carga los tres archivos que mencionamos anteriormente : conec.pro, t2.pro y menus.pro. Cuando se termina de cargar el archivo traduc2.pro, la llamada al programa genera la siguiente pantalla :

```
                TRADUCTOR AUTOMATICO DE FORMULAS
                   DE LOGICA DE PRIMER ORDEN
                        (Menu principal)

      Clausulas          (Transforma a forma clausular)
      Fns                (Transforma a forma normal de skolem)
      Ayuda              (Describe los procesos de traduccion)
      Sistema            (Regresa al sistema operativo)

>Teclea c, f, a o bien s :
>
```

si tecleamos la letra a aparecerá el siguiente menú en pantalla :

```
                    MENU DE INFORMACION.

Clausulas   (Informacion para traducir a forma clausular)
Fns         (Informacion para traducir a forma normal de skolem)
Menu        (Regresa al menu principal)

>Teclea c, f  o  m :
>
```

A continuación explicaremos como llevar a cabo la traducción a forma clausular, a cláusulas PROLOG y a forma normal de skolem.

V.1.1 Traducción a Forma Clausular y a Cláusulas PROLOG.

Antes de explicar como llevar a cabo esta traducción, es importante aclarar que para transformar una fórmula a cláusulas PROLOG, es necesario haberla transformado primero a forma clausular; ya que el predicado que imprime las cláusulas PROLOG, utiliza la lista de cláusulas generada por la traducción a forma clausular.

Para traducir a forma clausular, debemos escoger la primera opción del menú principal y teclear la letra c. Al hacer esto, el programa nos preguntará por: la fórmula que queremos traducir, el nombre de la variable para renombrar las variables acotadas de la fórmula y por un símbolo funcional para generar las funciones de skolem. Finalmente, imprimirá en pantalla la forma clausular resultante y el indicador del intérprete ":" aparecerá de nuevo. Si en este momento queremos tener las distintas cláusulas PROLOG correspondientes, escribimos la siguiente meta :

```
:consult(clausula)!
```

El predicado consult(clausula), carga a memoria el archivo *clausula*. Si éste contiene metas, ellas son activadas en el orden en el que aparecen. Por lo tanto al cargarse a memoria el archivo *clausula*, la meta imprime_p(Cls)? que fue escrita en el archivo por el predicado *imprime* (Cap. IV, sección IV.1.1.2, pag.72), es activada y la impresión en forma de cláusulas PROLOG se realiza en la pantalla. A continuación aparecerá el siguiente mensaje :

```
>Quieres guardar las clausulas PROLOG en un archivo (s/n)?
```

si respondemos s, el programa nos preguntará por el nombre bajo el cual queremos guardar las cláusulas PROLOG, este nombre no debe contener extensión.

Después de esto, para poder traducir otra fórmula, ya sea a forma clausular o a forma normal de skolem o bien terminar el programa, al aparecer el indicador de prolog ":", debemos teclear la meta siguiente :

: menu?

con ella aparecerá en pantalla nuevamente el menú principal y de esta manera podremos escoger la siguiente opción deseada.

V.1.2 Traducción a Forma Normal de Skolem.

Para traducir una fórmula a forma normal de skolem, debemos escoger la segunda opción del menú principal y teclear la letra f. Al hacer esto, el programa nos preguntará por: la fórmula que queremos traducir, el nombre de la variable para renombrar las variables acotadas de la fórmula y por un símbolo funcional para generar las funciones de skolem. Finalmente, imprimirá en pantalla la forma normal de skolem resultante y el indicador ":" de prolog-86. Después de esto, para poder continuar con el programa, debemos teclear la meta siguiente :

: menu?

con ella aparecerá en pantalla de nuevo el menú principal y de esta manera podremos continuar utilizando el programa.

Esta misma información ilustrada con ejemplos, se podrá encontrar al teclear las opciones del menú de información del programa, facilitando así al usuario, el acceso a la información sobre el funcionamiento del mismo.

V.2 Ejemplos.

Mostraremos el funcionamiento del programa con 5 ejemplos. Los primeros tres ejemplos son muy sencillos. El primero transforma una fórmula a forma clausular; el segundo y el tercero transforman fórmulas a forma normal de skolem. Los últimos dos ejemplos son un poco más complejos. El primero traduce una fórmula a forma clausular y a cláusulas PROLOG y el segundo una fórmula a forma clausular. Solamente en el primer ejemplo y en el cuarto describiremos paso a paso lo que el programa y el usuario van escribiendo.

Ejemplo 1.

Supongamos que después de la aparición del menú principal, tecleamos la letra c, i.e. :

```
            TRADUCTOR AUTOMATICO DE FORMULAS
               DE LOGICA DE PRIMER ORDEN
                    (Menu principal)

     Clausulas          (Transforma a forma clausular)
     Fns                (Transforma a forma normal de skolem)
     Ayuda              (Describe los procesos de traduccion)
     Sistema            (Regresa al sistema operativo)

>Teclea c, f, a  o bien s :
> c
```

Al hacer esto en la pantalla aparecerá el siguiente mensaje :

```
>Que formula quieres traducir (Al final teclea un ".") #
>
```

después del signo " > ", escribimos la siguiente fórmula :

```
>all(x,animal(x) => exists(y, mama(y,x))).
```

que corresponde a la fórmula

$$\forall x(animal(x) \rightarrow \exists y\ mama(y,x))$$

ahora el programa escribirá :

```
>Dame una variable.
>
```

después del signo " > " escribimos la siguiente variable :

```
>x
```

el programa responderá lo siguiente :

```
>Dame un simbolo funcional.
>
```

después del signo " > ", escribimos el siguiente simbolo funcional :

```
>f
```

La respuesta será :

```
>La formula en forma clausular es la siguiente :

mamaCf1(x1),x1) <- animalCx1).

>Quieres generar el conjunto de clausulas PROLOG (s/n) #
>
```

Después del signo " > ", escribimos s o n dependiendo de lo que deseemos hacer. En este caso responderemos con n. En el ejemplo 4 veremos un ejemplo de traducción de cláusulas a cláusulas PROLOG.

Ejemplo 2.

Si tecleamos la letra f, al aparecer el menú principal para traducir una fórmula a forma normal de skolem, contestamos a la pregunta de que fórmula se desea traducir con la misma fórmula del ejemplo anterior :

```
>allCx,animalCx) => existsCy, mamaCy,x))).
```

y damos x como variable y f como símbolo funcional, el programa responderá :

```
>La formula en forma normal de skolem es la siguiente :

~animalCx1) # mamaCf1(x1),x1)
```

Ejemplo 3.

Si con la opción f damos la siguiente fórmula :

```
>allCx,existsCy,masCx,y,cero))).
```

y además damos x y f como símbolos obtendremos :

```
>La formula en forma normal de skolem es la siguiente :

mas(x1,f1(x1),cero).
```

Ejemplo 4.

Supongamos que después de la aparición del menú principal, tecleamos la letra c, i.e. :

```
                TRADUCTOR AUTOMATICO DE FORMULAS
                    DE LOGICA DE PRIMER ORDEN
                        (Menu principal)

     Clausulas          (Transforma a forma clausular)
     Fns                (Transforma a forma normal de skolem)
     Ayuda              (Describe los procesos de traduccion)
     Sistema            (Regresa al sistema operativo)

>Teclea c, f, s  o bien s :
> c
```

Al hacer esto en la pantalla aparecerá el siguiente mensaje :

```
>Que formula quieres traducir (Al final teclea un ".") ?
>
```

después del signo " > ", escribimos la siguiente fórmula :

```
>all(x,exists(w,all(u,exists(z,p(w,z))&(m(x,u,w)#exists(y,q(u,y.
))))))).
```

que corresponde a la fórmula

$$\forall x \exists w \forall u (\exists z\ p(w,z)\ \&\ (m(x,u,w)\ \vee\ \exists y\ q(u,y))) \qquad (V.1)$$

ahora el programa escribirá :

```
>Dame una variable.
>
```

después del signo " > " escribimos la siguiente variable :

```
>x
```

el programa responderá lo siguiente :

```
>Dame un simbolo funcional.
>
```

después del signo " > ", escribimos el siguiente simbolo funcional :

```
>f
```

Finalmente, después de aproximadamente 15 segundos, obtendremos el resultado y mensaje siguientes :

```
>La formula en forma clausular es la siguiente :

 p(f3(x1),f1(x1)).
 m(x1,x3,f3(x1)) # q(x3,f2(x3,x1)).

 >Quieres generar el conjunto de clausulas PROLOG  (s/n) ?
 >
```

...(V.2)

Después del signo " > ", escribimos s o n dependiendo de lo que deseemos hacer :

Caso a

Por falta de espacio en el stack, las cláusulas PROLOG correspondientes no pueden ser generadas directamente; para poder generarlas, el usuario debe cargar a memoria el archivo *clausula*. Si a la pregunta anterior contestamos con s, en la pantalla aparecerá el siguiente mensaje :

```
>s
>Para generar las clausulas PROLOG, despues del indicador de
prolog " : ", escribe la siguiente meta :
:consult(clausula) /
```

Entonces para generar las distintas cláusulas PROLOG, escribimos la meta :

:consult(clausula)!

y el programa escribirá en pantalla el conjunto de cláusulas PROLOG correspondiente y nos preguntará si queremos guardarlo en un archivo :

```
p(f3(x1),f1(x1)).

m(x1,x3,f3(x1)) :-
          not(q(x3,f2(x3,x1))).

o

q(x3,f2(x3,x1)) :-
          not(m(x1,x3,f3(x1))).

>Quieres guardar las clausulas PROLOG en un archivo (s/n) ?
>
```

después del signo " > ", escribimos : s o n.

Caso a1.

Si escribimos s, el programa preguntará lo siguiente :

```
>s
>Bajo que nombre (maximo 8 letras) guardo el archivo ?
>
```

a lo cual, responderemos con un nombre que no tenga extensión, por ejemplo :

```
>prueba
```

Caso a2.

Si escribimos n, la ejecución de la traducción termina y aparece el siguiente mensaje :

```
>n
>Para entrar al menu principal teclea : menu?
```

Entonces, al aparecer el indicador de prolog-88 " > ", escribimos la meta : menu? . Con esto aparecerá nuevamente el menú principal en pantalla y podremos elegir otra opción.

Caso b.

Si a la pregunta

```
>Quieres generar el conjunto de clausulas PROLOG (s/n) ?
```

contestamos con n, se repetirá el caso a2.

En el capítulo II, al analizar los métodos 2 y 3, dijimos que había fórmulas para las cuales el método 3 daba mejores resultados que el método 2 y viceversa. En particular, en la sección II.3 de ese capítulo, vimos que para la fórmula de este primer ejemplo V.1, el método 3 da mejores resultados. Al correr este mismo ejemplo con la versión traduc3.pro, el programa que skolemiza con el método 3, obtenemos la siguiente respuesta :

```
>La formula en forma clausular es la siguiente :

p(f1(x1),f2(x1)).
m(x1,x3,f1(x1)) # q(x3,f3(x3)).
```

(V.3)

que es óptima con respecto a la que obtuvimos con la versión traduc2.pro en el cuadro V.2, ya que la función de skolem fs que sustituye a y , en este caso, sólo depende de una variable x3, mientras que en V.2 la función fz que sustituye a y depende de dos variables. Es importante notar que los nombres de las funciones de skolem difieren de V.2 a V.3 ya que en V.2 se utilizó el método 2 para crear las funciones y por esto éstas se generaron de adentro hacia afuera: primero se skolemizó el ∃z, después el ∃y y por último ∃w. En V.3, dado que se skolemizó con el método 3, las funciones de skolem se generaron de afuera hacia adentro : primero se skolemizó ∃w, después ∃z y por último ∃y.

En el capítulo II expusimos una mejora para el método 2 (pag. 42) con la cual para esta fórmula se obtendría el mismo resultado que con el método 3.

Ejemplo 5.

Si tecleamos la letra c, al aparecer el menú principal, damos como variable y y como símbolo funcional *g* y contestamos a la pregunta de que fórmula se desea traducir con la siguiente fórmula :

```
>(all(x,p(x,x))&all(x,all(y,all(z,(p(x,y)&p(y,z))=>p(x,z))))&
all(x,all(y,p(x,y)#p(y,x))))=>exists(y,all(x,p(y,x))).
```

que corresponde a la fórmula

(∀x p(x,x) & ∀x∀y∀z((p(x,y) & p(y,z)) → p(x,z)) & ∀x∀y(p(x,y) V p(y,x))) → ∃y∀x p(y,x)

obtendremos el siguiente resultado y el siguiente mensaje :

```
>La formula en forma clausular es la siguiente :

p(g3,g2) # p(g7,y8) <- p(g1,g1) & p(g5,g6).
p(g3,g2) # p(g7,y8) <- p(g1,g1) & p(g6,g5).
p(g2,g4) # p(g7,y8) <- p(g1,g1) & p(g5,g6).
p(g2,g4) # p(g7,y8) <- p(g1,g1) & p(g6,g5).
p(g7,y8) <- p(g1,g1) & p(g3,g4) & p(g5,g6).
p(g7,y8) <- p(g1,g1) & p(g3,g4) & p(g6,g5).
```

Este ejemplo requiere un poco más de tiempo para responder ya que esta fórmula es de las más complejas que hemos traducido con el programa.

CONCLUSIONES.

Para concluir, empezaremos por hacer un recuento de lo que a lo largo de esta tesis fueron los puntos más importantes : primero presentamos el algoritmo clásico de traducción de una fórmula de primer orden a forma clausular (método 1); después, analizamos dos variantes de este algoritmo (métodos 2 y 3) y comparamos los tres algoritmos presentados. También programamos las dos variantes del algoritmo clásico en PROLOG, para así poder llevar a cabo la traducción automáticamente. Por último diseñamos un predicado que toma la forma clausular obtenida y la transforma en una serie de cláusulas PROLOG usando el concepto de negación extralógica de PROLOG.

A continuación presentamos un cuadro que contiene los algoritmos de traducción de los tres métodos analizados :

METODO 1	METODO 2	METODO 3
1. Eliminar → y ↔ y reducir el alcance de la negación.	1. Eliminar → y ↔ y reducir el alcance de la negación.	1. Eliminar → y ↔ y reducir el alcance de la negación.
2. Prenexar.	2. Reducir el alcance de los cuantificadores.	2. Skolemizar.
3. Paso a FNC.	3. Skolemizar.	3. Prenexar y eliminar c. universales.
4. Skolemizar.	4. Prenexar y eliminar c. universales.	4. Paso a FNC.
5. Eliminar c. universales.	5. Paso a FNC.	5. Paso a Forma Clausular.
6. Paso a Forma Clausular.	6. Paso a Forma Clausular.	

...Ci

El método 1 no fue programado, en primer lugar, porque los métodos 2 y 3 dan mejores resultados que él y además porque, como dijimos

en el capitulo II, Clocksin y Mellish presentan en [3] un programa que se asemeja bastante a la automatización de éste. Sin embargo creemos que es interesante esquematizar la secuencia principal que la implantación de cada uno de los tres métodos tendría para poder comparar la implantación de cada método con los métodos correspondientes del cuadro anterior y además para poder comparar las secuencias de los tres porgramas. A continuación presentamos un cuadro con las secuencias de los tres programas :

IMPLANTACION DEL METODO 1	IMPLANTACION DEL METODO 2	IMPLANTACION DEL METODO 3
1) eli_imp	1) eli_imp	1) eli_imp
2) neg_dentro	2) neg_dentro	2) neg_dentro
3) skolem1	3) cuantif_dentro skolem2	3) skolem3
4) univ_fuera	4) univ_fuera	4) univ_fuera
5) fnc	5) fnc	5) fnc
6) gen_cls	6) gen_cls	6) gen_cls

...C2

Para poder comparar el método 1 con su implantación supondremos que las fórmulas que transformaremos con este método, son cerradas, ya que Clocksin y Mellish así lo hacen. Además, supondremos que al prenexar, los cuantificadores salen al principio de la fórmula en el orden en el que se encuentran. Para los tres métodos supondremos que las variables acotadas de las fórmulas ya fueron renombradas, para así poder mover libremente los cuantificadores de lugar.

Al analizar los pasos del cuadro correspondiente a la implantación del método 1, vemos que ésta difiere del mismo en dos aspectos. Por un lado, en el método 1 la fórmula es prenexada después de que las negaciones han sido introducidas y antes de skolemizar los cuantificadores existenciales, cosa que su implantación no lleva a cabo. Nótese sin embargo, que al skolemizar la fórmula con el método 1, los argumentos de las funciones de skolem son las variables cuantificadas universalmente anteriores al cuantificador existencial en cuestión, y en la implantación, aunque la fórmula es skolemizada sin mover los cuantificadores, el predicado *skolem1* va listando las variables acotadas universalmente anteriores al cuantificador existencial haciendo que la función de skolem dependa de estas variables al

igual que si los cuantificadores hubieran sido movidos físicamente. Por esto el efecto de la implantación es el mismo que en el método 1 aunque no se realice de igual manera.

Por otro lado, el segundo aspecto en el que difiere la implantación del método 1 con el método mismo radica en el momento en el que se lleva a cabo el paso a forma normal conjuntiva. En el método 1, este paso se puede llevar a cabo antes de la skolemización ya que la fórmula ha sido prenexada y por lo tanto la matriz esta libre de cuantificadores, mientras que en la implantación, los cuantificadores no se mueven físicamente, por lo que es necesario esperar a la skolemización y posteriormente a la eliminación de los cuantificadores universales para poder efectuar la normalización. Sin embargo, el cambio de orden del paso a forma normal conjuntiva no afecta la fórmula obtenida.

Los pasos de los cuadros correspondientes a las implantaciones de los métodos 2 y 3 corresponden exactamente con los pasos de los métodos 2 y 3 del cuadro C1. En estos dos casos la implantación de los métodos corresponde exactamente con los algoritmos de los métodos mismos.

Analicemos ahora, la diferencia principal entre la implantación de los 3 métodos. En realidad, lo que varia en las tres implantaciones es la forma de skolemizar. La implantación del método 1 no lleva a cabo un análisis de cuales variables forman, en realidad, parte de los argumentos de las funciones de skolem y cuales son superfluas. Mientras que con la implantación del método 2, antes de skolemizar, el alcance de los cuantificadores es reducido al mínimo, introduciendo en la fórmula todos los cuantificadores posibles y reduciendo así, el número de variables acotadas universalmente que afectan las funciones de skolem. En la implantación del método 3, el proceso de skolemización se lleva a cabo sin mover los cuantificadores hacia dentro. Pero en el momento de sustituir cada variable por su función de skolem, se hace una depuración de los argumentos de las funciones quitando aquellos que son superfluos ya que sólo se toman en cuenta las variables que son libres dentro del alcance del cuantificador existencial.

Aparentemente el efecto de la skolemización del método 2 y del método 3, el primero moviendo físicamente los cuantificadores y el segundo haciendo sólo un análisis sobre las variables libres, es el mismo. Sin embargo, si analizamos con más cuidado el proceso nos damos cuenta de que en algunos casos, la introducción de los cuantificadores no puede completarse debido a la forma particular en que están organizados los cuantificadores de la fórmula (ejemplo II.4, pag.36,Cap.II). Esto implica que a veces, con el método 2, algunas variables superfluas no pueden ser eliminadas con la introducción de los cuantificadores. En cambio, el método 3 no presenta este problema ya que la decisión de cuales variables intervienen en las funciones de skolem se hace a través de un análisis y no de la reducción real del alcance de los

cuantificadores. El problema que acabamos de mencionar para el método 2, puede ser subsanado con la primera modificación que se discute en el capítulo II. Creemos que con ella, el método 2 llega a ser igual o mejor que el método 3.

En cuanto a la comparación de los métodos 2 y 3, aunque por lo dicho en el párrafo anterior parecería que el método 3 es mejor, esto no puede ser afirmado, pues a veces, con el método 3 algunas variables superfluas sí son consideradas en el análisis de variables libres, como en el ejemplo II.10 de la página 41; en cambio, el método 2 no presenta este problema, ya que la variable superflua en este caso se elimina como argumento, al introducir los cuantificadores. No se ve la posibilidad de que este problema del método 3 pueda ser subsanado. En realidad, no podemos concluir que uno de los métodos 2 ó 3 sea mejor en términos absolutos, pues esto depende de la forma original de la fórmula dada.

Por otro lado, hay cierto tipo de fórmulas para las cuales ni el método 2, ni el método 3, ni la primera modificación del método 2 dan resultados óptimos, por lo que en el capítulo II proponemos también una segunda modificación al método 2 que trata este tipo de fórmulas optimamente.

Una conclusión que podemos obtener de este estudio es que los métodos 2 y 3 son mejores que el método 1 y su implantación corrobora este análisis, ya que ambos métodos generan funciones de skolem con menos argumentos que las del método 1.

Nuestra experiencia con la implantación de los dos métodos nos mostró que para fórmulas complejas, la implantación del método 2 es más eficiente en tiempo de ejecución que la del método 3. Sin embargo, en máquinas más rápidas, el tiempo de ejecución para ambas implantaciones es casi imperceptible. La forma clausular de fórmulas complejas se obtuvo en a lo más 20 segundos en una máquina tipo PC-XT y el tiempo promedio para fórmulas comunes fue de aproximadamente 6 segundos.

En el momento en el que hicimos la implantación de los algoritmos de traducción, contabamos solamente con PROLOG para micros, en particular, con el intérprete Prolog-86 y con el compilador Turbo-Prolog. De estas dos opciones escogimos el intérprete Prolog-86, ya que éste muestra las características escenciales de un lenguaje lógico y tiene definidos predicados intrínsecos de mucha utilidad : el *name*, el *univ*, el *functor*, el *assert*, el *op* , etc. Sin embargo, actualmente existen compiladores para micros como Arity/Prolog, con al menos las mismas características que Prolog-86, que por el hecho de ser compiladores harían que nuestros programas fueran más eficientes en tiempos de ejecución. Esta traducción a Arity/Prolog ya no se llevo a cabo.

Una vez que automatizamos el paso de una fórmula de primer orden a forma clausular, resulta relativamente sencillo automatizar el paso de forma clausular a cláusulas de PROLOG, por lo que decidimos hacer el programa que lleva a cabo esta transformación. Sin embargo, sabemos que no todas las cláusulas tienen la forma de una cláusula de Horn, por lo que en principio no toda cláusula puede ser llevada a una cláusula PROLOG. Ahora bien, haciendo uso de la negación como hecho extralógico del lenguaje PROLOG, siempre es posible encontrar una cláusula de PROLOG que pueda sustituir a una cláusula no de Horn, por lo que nuestro programa puede pasar de cualquier forma clausular a un conjunto de cláusulas PROLOG. Este hecho puede resultar importante porque una vez que tenemos un conjunto de cláusulas PROLOG, podemos utilizar un intérprete de PROLOG para llevar a cabo las deducciones de manera automática. Nótese sin embargo, que cada cláusula PROLOG generada, haciendo uso de la negación extralógica, ni es equivalente ni tiene el mismo potencial de deducción que la cláusula original. Este problema se discute en el capítulo IV.

Ahora nos gustaría mencionar la utilidad que tiene el haber programado los métodos de traducción de una fórmula de primer orden a forma clausular :

1) Trabajar con formas clausulares permite automatizar el proceso de deducción, es decir si tomamos dos fórmulas de primer orden F y G, y queremos probar que F es deducible de G, una manera de hacerlo es pasar ambas a su forma clausular y demostrar que cada cláusula de F es deducible del conjunto de cláusulas de G.

2) Proporcionar un sistema que genere automáticamente formas clausulares razonablemente buenas, puede resultar de mucha utilidad para fórmulas complejas en donde la optimización de las funciones de skolem no es trivial.

3) Contar con dos métodos automatizados de optimización de funciones de skolem permite hacer un análisis sobre la optimización de formas clausulares de una fórmula.

4) Servir como herramienta de apoyo para bases de conocimiento deductivas en PROLOG. En un proceso de tratamiento del lenguaje natural hay que traducir primero los enunciados a lógica de primer orden y después a programas PROLOG. Nuestros programas realizan la segunda fase del proceso.

De este trabajo se desprenden varios problemas que quedaron por resolver : 1) las dos mejoras al método 2 propuestas en el capítulo II no fueron programadas y presentan una posible continuación de este trabajo, 2) el reprogramar el sistema en un PROLOG compilable generaría seguramente un sistema más eficiente y tampoco se realizó, 3) la notación utilizada para las fórmulas es muy obscura y valdría la pena hacer un preprocesador que aceptara

la notación con los conectivos y cuantificadores comunes (∀, ∃, &, ∨, ~) y la transformara a los predicados aceptados por el programa y 4) nos parece importante documentar, con las demostraciones correspondientes, los lemas y teoremas que fundamentan los métodos de traducción 1, 2 y 3; aunque la mayoría de estos resultados existen en diferentes libros, sería interesante conjuntarlos en un solo trabajo.

ANEXOS.

ANEXO 1.
Predicados principales del programa.

ejecuta : lleva a cabo los siguientes pasos : pregunta por la fórmula Fa que será traducida y por la variable Var para renombrar las variables acotadas de la fórmula, renombra las variables acotadas de Fa y genera como resultado F, traduce F a forma clausular dejando la lista de cláusulas resultante en Cls y por último imprime las cláusulas de Cls.

```
ejecuta :-
        datos(Fa,Var),
        renombra(Fa,F,[],Var),
        traduce(F,Cls),
        imprime(Cls),
        opcion.
```

traduce(F,Cls) : traduce una fórmula F de lógica de primer orden a forma clausular dejando la lista de cláusulas resultante en Cls.

```
traduce(F,Cls) :-
                eli_imp(F,F1),
                neg_dentro(F1,F2),
                pre_skolem(F2,F3),
                fcl(F3,Cls).
```

eli_imp(F,F1) : F1 es la fórmula generada al eliminar de F todos los conectivos de la forma → y ↔.

```
eli_imp(P ↔ Q, (P1 & Q1) # (~P1 & ~Q1)) :-
                    !,
                    eli_imp(P,P1),
                    eli_imp(Q,Q1).
eli_imp(P → Q, ~P1 # Q1) :-
                    !,
                    eli_imp(P,P1),
                    eli_imp(Q,Q1).
eli_imp(all(X,P), all(X,P1)) :-
                    !,
                    eli_imp(P,P1).
eli_imp(exists(X,P), exists(X,P1)) :-
                    !,
                    eli_imp(P,P1).
eli_imp(P & Q, P1 & Q1) :-
                    !,
                    eli_imp(P,P1),
                    eli_imp(Q,Q1).
```

```
eli_imp(P # Q, P1 # Q1) :-
                    !,
                    eli_imp(P,P1),
                    eli_imp(Q,Q1).
eli_imp(~P, ~P1) :-
                    !,
                    eli_imp(P,P1).
eli_imp(P, P).
```

neg_dentro(F1,F2) : recorre la fórmula F1 de izquierda a derecha hasta que encuentra una negación, en ese momento llama a neg para introducir las negaciones dejando el resultado en F2.

```
neg_dentro(~P, P1) :-
                    !,
                    neg(P,P1).
neg_dentro(all(X,P), all(X,P1)) :-
                    !,
                    neg_dentro(P,P1).
neg_dentro(exists(X,P), exists(X,P1)) :-
                    !,
                    neg_dentro(P,P1).
neg_dentro(P & Q, P1 & Q1) :-
                    !,
                    neg_dentro(P,P1),
                    neg_dentro(Q,Q1).
neg_dentro(P # Q, P1 # Q1) :-
                    !,
                    neg_dentro(P,P1),
                    neg_dentro(Q,Q1).
neg_dentro(P, P).
```

neg(F1,F2) : F2 es el resultado de introducir en F1 las negaciones.

```
neg(~P, P1) :-
            !,
            neg_dentro(P,P1).
neg(all(X,P), exists(X,P1)) :-
            !,
            neg(P,P1).
neg(exists(X,P), all(X,P1)) :-
            !,
            neg(P,P1).
neg(P & Q, P1 # Q1) :-
            !,
            neg(P,P1),
            neg(Q,Q1).
```

```
neg(P # Q, P1 & Q1) :-
            !,
            neg(P,P1),
            neg(Q,Q1).
neg(P, ~P).
```

pre_skolem(F2,F3) : F3 es el resultado de skolemizar la fórmula F2; ya sea con skolem2 ó con skolem3.

Caso a: skolem2.

```
pre_skolem(F2,F3) :-
                    cuantif_dentro(F2,Fa),
                    lee_funcion(S),
                    skolem2(Fa,F3,[],[],Ls,S).


skolem2(all(X,P), all(X,P1),Us,Es,Ls,S) :-
                            !,
                            skolem2(P,P1,[X|Us],Es,Ls,S).
skolem2(exists(X,P), P2,Us,Es,Ls,S) :-
                            !,
                            skolem2(P,P1,Us,[X|Es],Ls,S),
                            concatena(Us,Ls,Args),
                            gen_sim(S,F),
                            Sk =.. [F|Args],
                            subst(X,Sk,P1,P2).
skolem2(P # Q, P1 # Q1,Us,Es,Ls,S) :-
                            !,
                            skolem2(P,P1,Us,Es,Ls1,S),
                            skolem2(Q,Q1,Us,Es,Ls2,S),
                            concatena(Ls1,L_Lib2,Laux),
                            depura_lista(Laux,Ls,[]).
skolem2(P & Q, P1 & Q1,Us,Es,Ls,S) :-
                            !,
                            skolem2(P,P1,Us,Es,Ls1,S),
                            skolem2(Q,Q1,Us,Es,Ls2,S),
                            concatena(Ls1,Ls2,Laux),
                            depura_lista(Laux,Ls,[]).
skolem2(~P, ~P1,Us,Es,Ls,S) :-
                            !,
                            skolem2(P,P1,Us,Es,Ls,S).
skolem2(P, P,Us,Es,Ls,S) :-
                            !,
                            gen_var(P,Lps),
                            concatena(Us,Es,As),
                            diferencia(As,Lps,Ls).
```

Caso b: skolem3.

```
pre_skolem(F2,F3) :-
                    vacuos(F2,Fa),
                    lee_funcion(S),
                    skolem3(Fa,F3,S).


skolem3(all(X,P), all(X,P1),S) :-
                              !,
                              skolem3(P,P1,S).
skolem3(exists(X,P), P2,S) :-
                              !,
                              obten_vars(P, Ls, [X], S),
                              gen_sim(S,Nom),
                              Sk =.. [Nom|Ls],
                              subst(X,Sk,P,P1),
                              skolem3(P1,P2,S).
skolem3(P & Q, P1 & Q1,S) :-
                              !,
                              skolem3(P,P1,S),
                              skolem3(Q,Q1,S).
skolem3(P # Q, P1 # Q1,S) :-
                              !,
                              skolem3(P,P1,S),
                              skolem3(Q,Q1,S).
skolem3(P, P,S).
```

fcl(F3,Cls) : Cls es la lista de cláusulas de la forma normal conjuntiva de F3, siendo F3 una fórmula en forma normal de skolem.

```
fcl(F3, Cls) :-
              univ_fuera(F3,Fa1),
              fnc(Fa1,Fa2),
              gen_cls(Fa2,Cls,[]).
```

ANEXO 2.
Predicados de utilería.

I. Predicados utilizados en fol.

univ_fuera(F3,Fa1) : Fa1 es el resultado de eliminar los cuantificadores universales de F3.

```
univ_fuera(all(X,P), P1) :-
                          !,
                          univ_fuera(P,P1).
univ_fuera(P & Q, P1 & Q1) :-
                          !,
                          univ_fuera(P,P1),
                          univ_fuera(Q,Q1).
univ_fuera(P # Q, P1 # Q1) :-
                          !,
                          univ_fuera(P,P1),
                          univ_fuera(Q,Q1).
univ_fuera(P, P).
```

fnc(Fa1,Fa2) : Fa2 es la forma normal conjuntiva de Fa1.

```
fnc(P & Q, P1 & Q1) :-
              !,
              fnc(P,P1),
              fnc(Q,Q1).
fnc(P # Q, R) :-
              !,
              fnc(P,P1),
              fnc(Q,Q1),
              fnc1(P1 # Q1,R).
fnc(P, P).
```

fnc1(F,F1) : Si F es una fórmula de la forma p # (q & r), F1 es la fórmula (p # q) & (p # r).

```
fnc1((P & Q) # R, P1 & Q1) :-
                           !,
                           fnc(P # R, P1),
                           fnc(Q # R, Q1).
fnc1(P # (Q & R), P1 & Q1) :-
                           !,
                           fnc(P # Q, P1),
                           fnc(P # R, Q1).
fnc1(P, P).
```

gen_cls(Fa2,Cls,L) : Si Fa2 es una forma normal conjuntiva, entonces Cls es la lista de cláusulas de la forma [cl(L,Ln)|Res] en donde L es la lista de literales no negadas de la clásula y Ln es la lista de literales negadas sin sus negaciones.

```
gen_cls(P & Q, C1, C2) :-
                        !,
                        gen_cls(P,C1,C3),
                        gen_cls(Q,C3,C2).
gen_cls(P, [cl(L,Ln) | Cs], Cs) :-
                        clasif_lit(P,L,[],Ln,[]),
                        !.
gen_cls(F,C,C).
```

II. Predicados usados en skolem2.

cuantif_dentro(P,P1) : P1 es el resultado de mover hacia dentro los cuantificadores de P eliminando los cuantificadores que resulten vacuos.

```
cuantif_dentro(all(X,all(Y,P)), P2) :-
                !,
                cuantif_dentro(all(Y,P),P1),
                iguales(all(Y,P),P1,Switch),
                inter_cuant(all(X,P1),P2,Switch).
cuantif_dentro(all(X,exists(Y,P)), P2) :-
                !,
                cuantif_dentro(exists(Y,P),P1),
                iguales(exists(Y,P),P1,Switch),
                checa_vacuo(all(X,P1),P2,Switch).
cuantif_dentro(exists(X,exists(Y,P)), P2) :-
                !,
                cuantif_dentro(exists(Y,P),P1),
                iguales(exists(Y,P),P1,Switch),
                inter_cuant(exists(X,P1),P2,Switch).
cuantif_dentro(exists(X,all(Y,P)), P2) :-
                !,
                cuantif_dentro(all(Y,P),P1),
                iguales(all(Y,P),P1,Switch),
                checa_vacuo(exists(X,P1),P2,Switch).
cuantif_dentro(all(X,P & Q), P1 & Q1) :-
                !,
                cuantif_dentro(all(X,P),P1),
                cuantif_dentro(all(X,Q),Q1).
cuantif_dentro(exists(X,P # Q), P1 # Q1) :-
                !,
                cuantif_dentro(exists(X,P),P1),
                cuantif_dentro(exists(X,Q),Q1).
```

```
cuantif_dentro(all(X,P # Q), P2) :-
                !,
                cuantif_dentro(P # Q,P1 # Q1),
                oc_formula(X,P1,Switch1),
                oc_formula(X,Q1,Switch2),
                formula_trans(all(X,P1 # Q1),P2,Switch1,Switch2).
cuantif_dentro(exists(X,P & Q), P2) :-
                !,
                cuantif_dentro(P & Q,P1 & Q1),
                oc_formula(X,P1,Switch1),
                oc_formula(X,Q1,Switch2),
                formula_trans(exists(X,P1 & Q1),P2,Switch1,Switch2).
cuantif_dentro(all(X,P), P1) :-
                !,
                oc_formula(X,P,Switch),
                elimina_vacuo(all(X,P),P1,Switch).
cuantif_dentro(exists(X,P), P1) :-
                !,
                oc_formula(X,P,Switch),
                elimina_vacuo(exists(X,P),P1,Switch).
cuantif_dentro(P # Q, P1 # Q1) :-
                !,
                cuantif_dentro(P,P1),
                cuantif_dentro(Q,Q1).
cuantif_dentro(P & Q, P1 & Q1) :-
                !,
                cuantif_dentro(P,P1),
                cuantif_dentro(Q,Q1).
cuantif_dentro(P, P).
```

III. Predicados utilizados en skolem3.

vacuos(P,P1) : checa si la fórmula P tiene cuantificadores vacuos y si es así los elimina dejando el resultado en P1.

```
vacuos(all(X,P), P2) :-
                      !,
                      oc_formula(X,P,Switch),
                      vacuos(P,P1),
                      formula_trans(all(X,P1),P2,Switch).
vacuos(exists(X,P), P2) :-
                      !,
                      oc_formula(X,P,Switch),
                      vacuos(P,P1),
                      formula_trans(exists(X,P1),P2,Switch).
vacuos(P & Q, P1 & Q1) :-
                      !,
                      vacuos(P,P1),
                      vacuos(Q,Q1).
```

```
vacuos(P # Q, P1 # Q1) :-
                        !,
                        vacuos(P,P1),
                        vacuos(Q,Q1).
vacuos(P, P).
```

IV. Predicados utilizados por skolem2 y skolem3.

> **lee_funcion(S)** : lee en S el simbolo funcional con el que se van a generar las funciones de skolem.

```
lee_funcion(S) :-
                write('>Dame un simbolo funcional. '),
                ratom(S),
                nl.
```

> **gen_var(P,Lps)** : si P es un predicado atómico, Lps es la lista de variables de los argumentos de P.

```
gen_var(P, Lps) :-
                 P =.. [Cab|Rs],
                 cons_lista(Rs,Las),
                 compacta(Las,Lais),
                 depura_lista(Lais,Lps,[]).
```

> **cons_lista(Ls,Lsi)** : si Ls es una lista de argumentos de un predicado atómico, Lsi es la lista de todas las variables que ocurren en sus argumentos.

```
cons_lista([],[]).
cons_lista([X|Res],[Y|Resi]) :-
                              functor(X,Nom,Arid),
                              Arid <> 0,
                              !,
                              X =.. [Cab|Rs],
                              cons_lista(Rs,Y),
                              cons_lista(Res,Resi).
cons_lista([X|Res],[Y|Resi]) :-
                              name(X, Ls),
                              obten_cab(Ls,Cab),
                              ascii(Cab,99),
                              !,
                              cons_lista(Res,Resi).
cons_lista([X|Res],[Y|Resi]) :-
                              cons_lista(Res,Resi).
```

obten_vars(P,Ls,As,S) : si P es una fórmula, Ls es la lista de variables libres de P y As es la lista de variables acotadas.S es el símbolo con el que se generan las funciones de skolem, como éstas pueden ser constantes es necesario checar que en Ls no ocurra una de ellas.

```
obten_vars(all(X,P),Ls,As,S) :-
                        !,
                        obten_vars(P,Ls,[X|As],S).
obten_vars(exists(X,P),Ls,As,S) :-
                        !,
                        obten_vars(P,Ls,[X|As],S).
obten_vars(P & Q,Ls,As,S) :-
                        !,
                        obten_vars(P,Ls1,As,S),
                        obten_vars(Q,Ls2,As,S),
                        concatena(Ls1,Ls2,Las),
                        depura_lista(Las,Ls,[]).
obten_vars(P # Q,Ls,As,S) :-
                        !,
                        obten_vars(P,Ls1,As,S),
                        obten_vars(Q,Ls2,As,S),
                        concatena(Ls1,Ls2,Las),
                        depura_lista(Las,Ls,[]).
obten_vars(~P,Ls,As,S) :-
                        !,
                        obten_vars(P,Ls,As,S).
obten_vars(P,Ls,As,S) :-
                        gen_lib(P,Ls,As,S).
```

compacta(Ls,Ls1) : si Ls es una lista de listas, Ls1 es la lista formada por todos los elementos de las listas de Ls.

```
compacta(Ls,Ls1) :-
                  compacta1(Ls,[],Ls1).


compacta1([X|Xs],S,Ys) :-
                        lista(X),
                        compacta1(X,[Xs|S],Ys).
compacta1([X|Xs],S,[X|Ys]) :-
                        atomic(X),
                        compacta1(Xs,S,Ys).
compacta1([],[X|S],Ys) :-
                        compacta1(X,S,Ys).
compacta1([],[],[]).
```

depura_lista(Ls,Ls1,Las) : Ls1 es el resultado de eliminar los elementos repetidos de la lista Ls utilizando Las como lista auxiliar.

```
depura_lista([X|Xs],Ls1,Las) :-
                          !,
                          depura_lista(X,La2s,Las),
                          depura_lista(Xs,Ls1,La2s).
depura_lista(X,Ls1,Las) :-
                          X <> [],
                          !,
                          agrega(X,Las,Ls1).
depura_lista([],Ls1,Ls1).
```

diferencia(As,Lps,Ls) : Ls es el resultado de "restar" la lista As de la lista Lps.

```
diferencia([X|Xs], Lps,Ls) :-
                          !,
                          diferencia(X,Lps,Las),
                          diferencia(Xs,Las,Ls).
diferencia(X,Lps,Lais) :-
                          X <> [],
                          !,
                          borra(X,Lps,Lais).
diferencia([],Ls,Ls).
```

V. Predicados globales del programa.

gensim(Raiz,At) : At es el símbolo generado con el nombre dado en Raiz más un entero positivo que se incrementa en cada llamada a gensim.

```
gensim(Raiz, Atomo) :-
              dame_num(Raiz, Num),
              name(Raiz, Nom1),
              concatena(Nom1, Num, Nombre),
              name(Atomo, Nombre).
```

dame_num(Raiz,Num) : en Num llevamos el contador del número de símbolos que hemos generado con Raiz.

```
dame_num(Raiz, Num) :-
                    retract(num_actual(Raiz,Num1)),
                    !,
                    Num2 is Num1 + 1,
                    asserta(num_actual(Raiz,Num2)),
                    crealista(Num2,Num).
dame_num(Raiz, Num) :-
                    asserta(num_actual(Raiz,1)),
                    crealista(1,Num).
```

subst(V,T,P,P1) : P1 es la fórmula que se obtiene a partir de P, sustituyendo todas las ocurrencias de la variable V por T.

```
subst(V, Sk, all(X,P), all(X,P1)) :-
                    !,
                    subst(V,Sk,P,P1).
subst(V, Sk, exists(X,P), exists(X,P1)) :-
                    !,
                    subst(V,Sk,P,P1).
subst(V, Sk, P & Q, P1 & Q1) :-
                    !,
                    subst(V,Sk,P,P1),
                    subst(V,Sk,Q,Q1).
subst(V, Sk, P # Q, P1 # Q1) :-
                    !,
                    subst(V,Sk,P,P1),
                    subst(V,Sk,Q,Q1).
subst(V, V1, P <-> Q, P1 <-> Q1) :-
                    !,
                    subst(V,V1,P,P1),
                    subst(V,V1,Q,Q1).
subst(V, V1, P -> Q, P1 -> Q1) :-
                    !,
                    subst(V,V1,P,P1),
                    subst(V,V1,Q,Q1).
subst(V, Sk, ~P, ~P1) :-
                    !,
                    subst(V,Sk,P,P1).
subst(V, Sk, P, P1) :-
                    !,
                    P =.. [Cab|Rs],
                    subst_var(V,Sk,Rs,Z),
                    P1 =.. [Cab|Z].
```

subst_var(V,T,L,Z) : Z es el resultado de sustituir V por T en la lista L.

```
subst_var(V, T, [], []).
subst_var(V, Sk, [V,..Li], [Sk,..N]) :-
                                        !,
                                        subst_var(V,Sk,Li,N).
subst_var(V, Sk, [Y,..Li], [Z,..N]) :-
                                        functor(Y,Nom,Arid),
                                        Arid <> 0,
                                        !,
                                        Y =.. [Cab|Rs],
                                        subst_var(V,Sk,Rs,Ls),
                                        Z =.. [Cab|Ls],
                                        subst_var(V,Sk,Li,N).
subst_var(V, Sk, [Y,..Li], [Z,..N]) :-
                                        subst_var(V,Sk,Li,N).
```

ANEXO 3.
Predicados para el preprocesamiento e impresión de fórmulas.

```
datos(Fa,Var)  : lee en Fa la fórmula a traducir y en Var el
símbolo para renombrar variables.
```

```
datos(F,Var) :-
  print('>Que formula quieres traducir (Al final teclea un .) ?'),
  read(F),
  nl,
  print('>Dame una variable.'),
  ratom(Var),
  nl.
```

```
renombra(Fa,F,L,Var) :renombra las variables acotadas de Fa con
Var1, Var2, etc., dejando el resultado en F.
```

```
renombra(all(X,P), all(X,P1), L, Var) :-
                        !,
                        gensim(Var, X1),
                        renombra(P, P1, [[X, X1], ..L], Var).
renombra(exists(X,P), exists(X1,P1), L, Var) :-
                        !,
                        gensim(Var, X1),
                        renombra(P, P1, [[X, X1], ..L], Var).
renombra(P <-> Q, P1 <-> Q1, L, Var) :-
                        !,
                        renombra(P, P1, L, Var),
                        renombra(Q, Q1, L, Var).
renombra(P -> Q, P1 -> Q1, L, Var) :-
                        !,
                        renombra(P, P1, L, Var),
                        renombra(Q, Q1, L, Var).
renombra(P & Q, P1 & Q1, L, Var) :-
                        !,
                        renombra(P, P1, L, Var),
                        renombra(Q, Q1, L, Var).
renombra(P # Q, P1 # Q1, L, Var) :-
                        !,
                        renombra(P, P1, L, Var),
                        renombra(Q, Q1, L, Var).
renombra(~P, ~P1, L, Var) :-
                        !,
                        renombra(P, P1, L, Var).
renombra(P, P1, L, Var) :-
                        substituye(P, P1, L).
```

substituye(P,P1,L) : sustituye en la fórmula atómica P las variables que aparecen como primeros argumentos de las listas de L por los segundos argumentos de las mismas generando P1.

```
substituye(P, P, []) :- !.
substituye(P, P2, [[Var,Var1],..Cola]) :-
                        subst(Var, Var1, P, P1),
                        substituye(P1, P2, Cola).
```

imprime(Cls) : imprime la cláusula representada por la lista Cls en forma de una implicación.

```
imprime(Cls):-
     print('>La formula en forma clausular es la siguiente: '),
     nl,
     resultado(Cls),
     archiva(Cls).
```

resultado(Cls) : separa las listas de la lista Cls para imprimir una por una.

```
resultado([]) :-
              !,
              nl,
              nl.
resultado([cl(L,Ln)|Cs]) :-
              impclaus(L, Ln),
              nl,
              resultado(Cs).
```

impclaus(L,Ln) : imprime la lista L como la disyunción del consecuente de la cláusula y la lista Ln como la conjunción del antecedente.

```
impclaus(L, []) :-
                !,
                disyuncion(L),
                write(.).
impclaus([], L) :-
                !,
                write(<-),
                tab(1),
                conjuncion(L),
                write(.).
```

```
impclaus(L1, L2) :-
                  disyuncion(L1),
                  tab(1),
                  write(<-),
                  tab(1),
                  conjuncion(L2),
                  write(.).
```

conjuncion(L) : imprime los elementos de L en una conjunción que forma el antecedente de cada cláusula.

```
conjuncion([L]) :-
                  !,
                  write(L).
conjuncion([L |Ls]) :-
                  write(L),
                  tab(1),
                  write(&),
                  tab(1),
                  conjuncion(Ls).
```

disyuncion(L) : imprime los elementos de L en una disyunción que forma el consecuente de cada cláusula.

```
disyuncion([L]) :-
                  !,
                  write(L).
disyuncion([L|Ls]) :-
                  write(L),
                  tab(1),
                  write(#),
                  tab(1),
                  disyuncion(Ls).
```

archiva(Cls) : escribe en el archivo clausula, la siguiente meta : imprime_p(Cls)? .

```
archiva(Cls) :-
              tell(clausula),
              prin('imprime_p('),
              write(Cls),
              print(')?'),
              close(clausula),
              nl,nl.
```

imprime_p(Cls) : imprime la lista de cláusulas, Cls, en forma de cláusulas de PROLOG, y pregunta si se quiere que el resultado quede guardado en un archivo.

```
imprime_p(Cls):-
  resultado2(Cls),
  prin('>Quieres guardar las clausulas PROLOG en un archivo '),
  print('(s/n) ?'),
  ratom(Res),
  guarda(Cls,Res),
  prin('>Para entrar al menu principal escribe la meta '),
  print('siguiente : menu? '),
  nl.
```

resultado2(Cls) : separa las listas de la lista Cls para imprimir una por una.

```
resultado2([]):-
              !,
              nl.
resultado2([cl(L,Ln)|Cs]) :-
              nl,
              length(L,N),
              impclaus2(L,L,Ln,1,N),
              nl,
              resultado2(Cs).
```

impclaus2(L,L,Ln,Cont,Long) : imprime para cada elemento 'l' de L, la regla cuya cabeza es 'l' y cuyo cuerpo esta formado por Ln y por la negación de todos los elementos de L exceptuando a 'l'.

```
impclaus2(L, Li, Ln, I, 0) :-
                            !,
                            write(:-),
                            conj(Ln),
                            write(.).
impclaus2(L, [Lc|Lr], Ln, I, 1) :-
                            !,
                            write(Lc),
                            vacia(Ln,[]),
                            conj(Ln),
                            write(.).
```

```
impclaus2(L, L1, Ln, I, N) :-
                              I > N.
impclaus2(L, [Lc|Lr], Ln, I, N) :-
                              I <= N,
                              write(Lc),
                              borra(Lc,L,Letneg),
                              vacia(Ln,Letneg),
                              conjuncion2(Ln,Letneg),
                              write(.),
                              nl,nl,
                              cond(I,N),
                              impclaus2(L,Lr,Ln,I+1,N).
```

vacia(L,L1) : escribe el simbolo ':-' si las listas L o L1 son no vacias.

```
vacia([],[]) :-
             !.
vacia(L,L1) :-
             tab(1),
             write(:-).
```

cond(I,N) : imprime la letra 'o' si I < N.

```
cond(I,N) :-
          I < N,
          !,
          write(o),
          nl,nl.
cond(I,N).
```

conjuncion2(Ln,Lneg) : genera el antecedente de cada cláusula, imprimiendo primero los elementos de Ln y después los de Lneg.

```
conjuncion2(Ln, Lneg) :-
                        conj(Ln),
                        coma(Ln,Lneg),
                        conj_neg(Lneg).
```

conj(L) : imprime los elementos de L (antecedente), separándolos por medio de comas.

```
conj([]) :-
         !.
```

```
conj([L]) :-
        !,
        nl,
        tab(9),
        write(L).
conj([Lc|Lr]) :-
        nl,
        tab(9),
        write(Lc),
        write(,),
        conj(Lr).
```

coma(L,L') : escribe una coma si L y L' son listas no vacias.

```
coma([Lnc|Lnr], [Lnegc|Lnegr]) :-
                                !,
                                write(,).
coma(L,L1).
```

conj_neg(L) : imprime los elementos de L precediéndolos por 'not' y separándolos por medio de comas.

```
conj_neg([]) :-
              !.
conj_neg([L]) :-
              !,
              nl,
              tab(9),
              write(not),
              prin('('),
              write(L),
              prin(')').
conj_neg([Lc|Lr]) :-
              nl,
              tab(9),
              write(not),
              prin('('),
              write(Lc),
              prin(')'),
              write(,),
              nl,
              conj_neg(Lr).
```

guarda(Cls,Res) : si el valor de Res es s , guarda en un archivo las cláusulas PROLOG generadas.

```
guarda(Cls,s):-
     !,
     print('>Bajo que nombre(max. 8 letras) guardo el archivo ?'),
     ratom(Nombre),
     tell(Nombre),
     resultado2(Cls),
     close(Nombre).
guarda(Cls,R).
```

opcion : pregunta al usuario si desea generar las distintas cláusulas PROLOG correspondientes a la fórmula original.

```
opcion :-
   prin('>Quieres generar el conjunto de clausulas PROLOG '),
   print('(s/n) ?'),
   ratom(C),
   change_case(Res,C),
   opcion(Res).
```

opcion(R) : si el valor de R es s, despliega en la pantalla una serie de instrucciones que indican como generar las cláusulas PROLOG correspondientes; si el valor de R es n despliega en la pantalla instrucciones que indican como desplegar el menú.

```
opcion(s) :-
          !,
          nl,nl,nl,
          prin('>Para generar las clausulas PROLOG, despues del '),
          print('prompt de prolog ":", escribe la '),
          print('meta siguiente :consult(clausula)! '),
          nl,nl.
opcion(n) :-
          !,
          nl,nl,nl,
          prin('>Para entrar al menu principal teclea la meta '),
          print('siguiente : menu? '),
          nl,nl.
opcion(O) :-
          print('>Teclea s  o bien  n '),
          ratom(C),
          chage_case(Res,C),
          opcion(Res).
```

opcion_fns : escribe en pantalla una serie de instrucciones que indican como desplegar el menú principal.

```
opcion_fns :-
   nl,nl,nl,nl,nl,
   prin('>Para entrar al menu principal escribe la siguiente '),
   print('meta : menu? '),
   nl,nl,nl,nl.
```

ANEXO 4.
Equivalencias lógicas utilizadas durante el proceso de traducción.

Si P, Q, R, F(x), G(x) y F(x,y) son fórmulas cualesquiera, D es una fórmula en la cual x no ocurre libre y F es una fórmula en la cual no ocurre x, entonces

Equivalencias correspondientes a los conectivos → y ↔.

$$P \rightarrow Q \equiv \neg P \vee Q$$
$$P \leftrightarrow Q \equiv (\neg P \wedge \neg Q) \vee (P \wedge Q)$$

Equivalencias de la negación.

$$\neg\neg P \equiv P$$
$$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$$
$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$$
$$\neg\exists x\, F(x) \equiv \forall x\, \neg F(x)$$
$$\neg\forall x\, F(x) \equiv \exists x\, \neg F(x)$$

Distribución de la disyunción.

$$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$$

Equivalencias de cuantificadores.

$$\forall x\,(F(x) \wedge G(x)) \equiv \forall x\, F(x) \wedge \forall x\, G(x)$$
$$\exists x\,(F(x) \vee G(x)) \equiv \exists x\, F(x) \vee \exists x\, G(x)$$
$$\forall x\,(F(x) \wedge D) \equiv \forall x\, F(x) \wedge D$$
$$\exists x\,(F(x) \vee D) \equiv \exists x\, F(x) \vee D$$
$$\forall x\,(F(x) \vee D) \equiv \forall x\, F(x) \vee D$$
$$\exists x\,(F(x) \wedge D) \equiv \forall x\, F(x) \wedge D$$
$$\forall x \forall y\, F(x,y) \equiv \forall y \forall x\, F(x,y)$$
$$\exists x \exists y\, F(x,y) \equiv \exists y \exists x\, F(x,y)$$
$$\forall x\, F \equiv F$$
$$\exists x\, F \equiv F$$

BIBLIOGRAFIA.

[1] ANDREWS, Peter B. An Introduction to Mathematical Logic and Type Theory : to Truth through Proof. Orlando,Florida, Academic Press,Inc., 1986.

[2] BUNDY, Alan. The Computer Modelling of Mathematical Reasoning. London, Academic Press,Inc., 1983.

[3] CLOCKSIN, W.F. and MELLISH, C.S. Programming in Prolog. Berlin, Springer-Verlag, 1984.

[4] HOGGER, Christopher J. Introduction to Logic Programming. London, Academic Press,Inc., 1984.

[5] LLOYD, J.W. Foundations of Logic Programming (Technical Report), 1984.

[6] MALITZ, J. Introduction to Mathematical Logic (Part III), New York, Springer-Verlag, 1984.

[7] MENDELSON, Elliot. Introduction to Mathematical Logic. New York, D. Van Nostrand Company, 1964.

[8] NILSSON, Nils J. Problem-Solving Methods in Artificial Intelligence. E.U.A., McGraw Hill, 1971.

[9] NILSSON, Nils J. Principles of Artificial Intelligence. Berlin, Springer-Verlag, 1982.

[10] PROLOG-86 Reference Manual, Software version 2.00, Document version 2.00.

[11] STERLING, L. and SHAPIRO, E. The Art of Prolog. MIT Press, 1986.

[12] VAN HEIJENOORT, J. El Desarrollo de la Teoría de la Cuantificación. Instituto de Investigaciones Filosóficas, UNAM, Cuaderno 32, 1976.