24
2 9

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

EDITOR LOGICO PARA AYUDA DE DEDUCCIONES FORMALES EN LOGICA

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 :

SIMON PEREZ GONZALEZ

DIRECTOR DE TESIS: DOCTORA MA. CRISTINA LOYO VARELA

CIUDAD UNIVERSITARIA, MEXICO, D. F. OCTUBRE 1990

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

INDICE

INDICE

INTRODUCCION

Una de las ocupaciones de la Lógica Matemática es la formalización axiomática de las teorías matemáticas y una de sus tareas consiste en el estudio de las demostraciones y deducciones formales.

Una deducción formal implica la construcción de una secuencia finita de fórmulas que se derivan de la aplicación de axiomas y reglas de inferencia a una serie de hipótesis.

La longitud de la secuencia de fórmulas que componen la deducción es muy variable, en algunas ocasiones esta secuencia se compone de tres ó cinco fórmulas y en otras, se puede llegar a obtener una secuencia de más de setenta u ochenta fórmulas. Escribir cada fórmula de la secuencia bajo la aplicación de un axioma o regla de inferencia puede volverse muy laborioso.

Para llevar a cabo la aplicación de un axioma, es necesario sustituir fórmulas por las distintas metavariables del esquema axiomático seleccionado. Cuando se utilizan fórmulas muy extensas, esta sustitición se vuelve tediosa y resulta muy fácil equivocarse.

Algunas veces la aplicación de una regla requiere, además de llevar a cabo las sustituciones necesarias, que se haga un chequeo de ciertas condiciones de aplicación y ésto implica un recorrido detallado por cada uno de los términos que aparecen en cada una de las fórmulas. Todas estas tareas se van complicando conforme se complica la deducción y son frecuentemente fuente de errores que impiden la realización de una deducción correcta.

Este trabajo de tesis, presenta la definición y realización de un EDITOR DE AYUDA PARA DEDUCCIONES FORMALES EN LOGICA DE PRIMER ORDEN. El editor nos proporciona un ambiente para realizar deducciones y nos brinda además una serie de herramientas para facilitar el trabajo. Tales herramientas pueden ser, por ejemplo,

la aplicación automática de axiomas y reglas de inferencia indicando simplemente el axioma o la regla de inferencia que se desea aplicar y las fórmulas que sustituirán a cada metavariable dentro del esquema axiomático.

El editor mismo aplicará dicho axioma o regla de inferencia, checando todas las condiciones necesarias para su aplicación y además verificará que las fórmulas estén correctamente escritas.

Por otro lado, el editor nos brinda los comandos típicos de edición y cuenta con facilidades para el manejo de archivos.

Para su pesentación, el trabajo está estructurado de la siguiente manera. En el primer capítulo, se introducen conceptos básicos de la lógica de primer orden, la noción de deducción formal y algunos teoremas fundamentales.

En el segundo capítulo, se muestra el funcionamiento del editor y sus comandos. Este capítulo puede ser utilizado como una guía para el usuario del editor.

En el último capítulo se describen la estructura general del programa del editor,así como la estructura de su ambiente de trabajo y los predicados más relevantes que se utilizan para hacer verificación de condiciones; y los predicados que realizan la aplicación de los axiomas.

Para terminar, se presentan brevemente las conclusiones del trabajo y el anexo, que contendrá el código de algunos predicados que apoyan la deducción.

CAPITULO I

CONCEPTOS DE LOGICA

1.1 INTRODUCCION

En este capitulo veremos una introducción simple a la lógica de primer orden y las deducciones formales que nos permitan entender y justificar el uso del **Editor Lógico** *que presentaremos más adelante.*

1.2 EL LENGUAJE

La lógica de primer orden es un modelo matemático del pensamiento deductivo; trabaja con fórmulas que involucran variables, constantes, letras (funcionales y predicativas), conectivos y cuantificadores. Comenzaremos entonces por describir formalmente el vocabulario de nuestro modelo matemático de pensamiento deductivo.

VOCABULARIO

- *Las* **constantes** *se identifican normalmente con las primeras letras del alfabeto, y si se desea, seguidas de un número:*

  a, b, c, a1, b1, c1, a2, b2, c2, ...

- *Las* **variables** *utilizan las últimas letras del alfabeto, y si se desea, seguidas de un número:*

  x, y, z, x1, y1, z1, x2, y2, z2, ...

- *Los* **símbolos lógicos** *están dados por:*

  Los conectivos lógicos:

$\leftrightarrow$	*equivalencia*
$\rightarrow$	*implicación*
$\wedge$	*conjunción*
$\vee$	*disyunción*
$\neg$	*negación*

  y los Cuantificadores:

$\forall$	*Universal*
$\exists$	*Existencial*

- *Los* **símbolos funcionales** *se denotan con las letras:*

  f, g, h, f1, g1, h1, f2, g2, h2, ...

- *Los* **símbolos predicativos** *utilizan:*

  p, q, r, p1, q1, r1, p2, q2, q2, ...

Finalmente,

- *Los* **símbolos de puntuación**

  "," , "(" , ")".

Las constantes y variables denotarán objetos, y los símbolos funcionales y predicativos denotarán nombres de funciones y de relaciones, respectivamente.

Construiremos el lenguaje a partir de estos símbolos, y lo dividiremos en tres categorías: primero definiremos los términos del lenguaje, luego definiremos sus predicados y por último definiremos las fórmulas bien formadas o expresiones formales del sistema.

DEFINICION (Términos)

Los términos son expresiones que denotan objetos, y se construyen de acuerdo a las siguientes reglas:

1) *Las constantes son términos*

2) *Las variables son términos*

3) *Si* $t_1, t_2, \ldots, t_n$ *con* $n \geq 1$ *son términos y* f *es un símbolo funcional de aridad* n*, entonces*

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

es un término.

4) *Todos los términos están dados por las definiciones 1), 2), y 3).*

EJEMPLO 1

Supongamos que el simbolo funcional **f** *es de aridad dos (binario) y el simbolo funcional* **g** *es de aridad tres, entonces para*

g(x,f(a,x),a)

a *es un término, ya que* **a** *es una constante;*

x *es un término, ya que* **x** *es una variable;*

f(a,x) *es un término, ya que* **a** *y* **x** *son términos y* **f** *es un simbolo funcional binario;*

g(x,f(a,x),a) *también es un término, ya que* **x**, **f(a,x)** *y* **a** *son términos y* **g** *es un simbolo funcional de aridad tres.*

DEFINICION (Predicados o Fórmulas Atómicas)

Los **predicados** *representan relaciones entre los objetos, y se construyen de acuerdo a las siguientes reglas:*

- *Si* **t1, t2, ...,tn** *son términos, con* **n** $\geq$ **1** *y* **p** *un simbolo predicativo de aridad* **n**, *entonces*

  p(t1,t2, ...,tn)

  es un predicado o fórmula atómica.

Por ejemplo, si **p** *es un simbolo predicativo de aridad tres, entonces*

p(a,x,f(a,x))

es un predicado, ya que **a**, **x** *y* **f(a,x)** *son términos y* **p** *es un simbolo predicativo.*

DEFINICION (Fórmula Bien Formada)

Una fórmula bien formada se contruye de acuerdo a las siguientes reglas:

1) *Un predicado es una fórmula bien formada.*

2) *Si* $\mathcal{F}$ *es una fórmula bien formada, entonces su* **negación**

$$(\neg \mathcal{F})$$

también lo es.

3) *Si* $\mathcal{F}$ *y* $\mathcal{G}$ *son fórmulas bien formadas, entonces su* **conjunción,**

$$(\mathcal{F} \wedge \mathcal{G}),$$

y su **disyunción,**

$$(\mathcal{F} \vee \mathcal{G}),$$

también lo son.

4) *Si* $\mathcal{F}$ *y* $\mathcal{G}$ *son fórmulas bien formadas, entonces la* **implicación**,

$$(\mathcal{F} \Rightarrow \mathcal{G}),$$

y la **equivalencia**,

$$(\mathcal{F} \Leftrightarrow \mathcal{G}),$$

también lo son.

5) *Si* **x** *es una variable y* $\mathcal{F}$ *es una fórmula bien formada, entonces,*

$$(\forall \mathbf{x}\ \mathcal{F}),$$

y

$$(\exists \mathbf{x}\ \mathcal{F})$$

también son fórmulas bien formadas.

6) *Las únicas fórmulas bien formadas están dadas por las definiciones 1) a 5).*

En 5) los prefijos $\forall$ *"para toda" y* $\exists$ *"existe alguna", se les conoce con el nombre de* **cuantificador universal** *y* **cuantificador existencial** *respectivamente; y se dice que en ambos casos* $\mathcal{F}$ *está* **bajo el alcance** *del cuantificador correspondiente.*

EJEMPLO 2

Supongamos que los símbolos funcionales **f**, **g**, *y el predicativo* **q** *son binarios, y que el símbolo predicativo* **p** *es de aridad 3, entonces:*

1) **p(a,x,f(a,x))**

es una fórmula bien formada y es atómica.

2) $q(g(b,x),y)$

es una fórmula bien formada y también es atómica.

3) $(\exists y\ q(g(b,x),y))$

es una fórmula bien formada y $q(g(b,x),y)$ está bajo el alcance del cuantificador existencial $\exists y$.

4) $(p(a,x,f(a,x)) \wedge (\forall y\ q(g(b,x),y)))$

es una fórmula bien formada, en donde $q(g(b,x),y)$ está bajo el alcance del cuantificador $\forall y$.

5) $((\forall x\ p(a,x,f(a,x))) \vee (\exists y\ q(g(b,x),y)))$

es una fórmula bien formada, en donde $p(a,x,f(a,x))$ está bajo el alcance del cuantificador $\forall x$ y $q(g(b,x),y)$ está bajo el alcance del cuantificador $\exists y$.

En muchos casos omitiremos paréntesis exteriores, cuando no sean necesarios para entender la fórmula, por ejemplo, si se tiene la fórmula

$$(\forall x\ p(a,x,f(a,x)))$$

entonces la escribiremos de la siguiente manera:

$$\forall x\ p(a,x,f(a,x)).$$

DEFINICION (Expresión Formal)

Una **expresión formal** *es un término o una fórmula bien formada.*

EJEMPLO 3

x,
p(x)
∀x p(a,x,f(a,x))

son expresiones formales.

De aquí en adelante, cuando hablemos de fórmulas nos referiremos a fórmulas bien formadas de nuestro sistema.

PRIORIDAD Y ASOCIACION DE OPERADORES

Los operadores juegan un papel fundamental, no sólo en la lógica matemática sino en todas las ramas de las matemáticas.

La prioridad de los operadores, está definida en el orden en que aparecen en la siguiente lista, por ejemplo, la disyunción tiene mayor prioridad que la conjunción.

1) *Negación y cuantificadores (existencial y universal) tienen la misma prioridad.*

2) *Disyunción.*

4) *Conjunción.*

5) *Implicación.*

6) *Equivalencia.*

Por otro parte, en lo que respecta a la asociación, todo se hace por la derecha.

EJEMPLO 4

Considerando lo anterior, cuando se escriba una fórmula de la siguiente manera:

$$\forall x p(x) \Rightarrow q(x)$$

se entenderá como sigue:

$$(\forall x p(x)) \Rightarrow q(x)$$

y la fórmula

$$\neg p(x) \Rightarrow q(x) \vee r(x)$$

debe entenderse así:

$$(\neg p(x)) \Rightarrow (q(x) \vee r(x)).$$

Finalmente, la fórmula

$$p(x) \Rightarrow q(x) \Rightarrow p(x)$$

debe entenderse como sigue:

$$p(x) \Rightarrow (q(x) \Rightarrow p(x)).$$

VARIABLES LIBRES Y ACOTADAS

Antes de definir lo que es una variable libre o acotada, daremos una definición útil para tal propósito.

DEFINICION (Ocurrencia)

Se dice que una variable **ocurre** *dentro de una fórmula, si dicha variable forma parte de la fórmula mencionada; puede haber más de una ocurrencia de una variable en una fórmula.*

EJEMPLO 5

Supongamos que el símbolo predicativo **p** *es binario y el símbolo predicativo* **q**, *es de aridad tres, entonces en la fórmula:*

$$\forall x(p(x,y) \Rightarrow q(x)) \vee \exists z p(z,y)$$

la variable **x** *tiene tres* **ocurrencias**, *una en* $\forall x$, *otra en* $p(x,y)$ *y la tercera en* $q(x)$. *Las tres ocurren en la subfórmula* $\forall x(p(x,y) \Rightarrow q(x))$.

La variable **y** *tiene dos* **ocurrencias**, *la primera en* $\forall x(p(x,y)$ *y la segunda en* $\exists z\ p(z,y)$.

La variable **z** *también tiene dos* **ocurrencias** *en la subfórmula* $\exists z p(z,y)$, *una en* $\exists z$ *y la otra en* $p(z,y)$.

DEFINICION (Variables Libres y Acotadas)

Se dice que una variable x *que ocurre dentro de una fórmula, es una* **variable acotada**, *si es la variable de algún cuantificador* $\forall x$ *o* $\exists x$, *o si la variable ocurre en una fórmula que está bajo el alcance de algún cuantificador con la misma variable. De otra manera se dice que es una* **variable libre**.

EJEMPLO 6

Supongamos que el símbolo predicativo **p** *es binario y el símbolo predicativo* **q**, *es de aridad uno (unario). Entonces en la fórmula :*

$$\forall x(p(x,y) \rightarrow q(x)) \vee \exists z p(z,y)$$

la variable x *tiene tres* **ocurrencias**, *y las tres ocurren en forma acotada, ya que la primera es la del cuantificador* $\forall x$, *y las otras dos son de la subfórmula:*

$$(p(x,y) \rightarrow q(x))$$

que se encuentra bajo el alcance del cuantificador $\forall x$.

La variable y *tiene dos* **ocurrencias**, *y las dos son libres, ya que* y *no ocurre en ningún cuantificador.*

Las dos ocurrencias de la variable z *son variables acotadas, la primera porque ocurre en* $\exists z$ *y la segunda en* $p(z,y)$ *que está bajo el alcance de* $\exists z$.

DEFINICION (Fórmula Cerrada)

Una fórmula $\mathcal{F}$ es cerrada, si todas las variables que ocurren en ella son variables acotadas.

EJEMPLO 7

La fórmula

$$\forall x p(x,y)$$

no es cerrada, porque la variable y ocurre libre en ella.

Pero la fórmula

$$\forall x \exists y p(x,y)$$

si es cerrada, ya que todas las variables que ocurren en ella estan acotadas.

La fórmula

$$p(c,f(c))$$

es cerrada pues, como no tiene variables, cumple la definición vacuamente.

DEFINICION (Término Libre)

Se dice que un término t es **libre para una variable** *x, en una fórmula $\mathcal{F}$, si ninguna ocurrencia libre de x en $\mathcal{F}$, estd bajo el alcance de algún cuantificador que contenga variables del término.*

La razón de pedir que un término sea libre para una variable, se justifica cuando se desea sustituir todas las variables libres de una fórmula bien formada por un término, y se quiere que todas las variables del término sigan siendo libres después de la sustitución.

EJEMPLO 8

1) Consideremos el siguiente término

$$f(a,x)$$

y la fórmula

$$\forall x(p(x,y) \Rightarrow q(x)) \vee \exists zp(z,y)$$

en este caso el término $f(a,x)$*, no es libre para la primera ocurrencia de la variable* y*, ya que aunque ésta variable es libre, si sustituimos el término* $f(a,x)$*, por la variable* y*, quedaría así:*

$$\forall x(p(x,f(a,x)) \Rightarrow q(x)) \vee \exists zp(z,y)$$

y la ocurrencia de la variable x *en* $f(a,x)$ *estaría bajo el alcance del cuantificador* $\forall x$ *y por lo tanto quedaría acotada.*

En cambio el término $f(a,x)$*, sí es libre para la segunda ocurrencia de la variable* y*, puesto que ninguna variable del término* $f(a,x)$ *(en este caso sólo tiene una:* x*) es la variable del cuantificador* $\exists z$*.*

Al sustituir la segunda ocurrencia de la variable y*, por el término* $f(a,x)$*, nuestra fórmula quedaría así:*

$$\forall x(p(x,y) \Rightarrow q(x)) \vee \exists zp(z,f(a,x))$$

Otros ejemplos de término libre para una variable, son los siguientes:

2) Todo término que no contenga variables, es libre para cualquier ocurrencia libre de una variable en cualquier fórmula.

3) Un término **t**, *es libre para cualquier ocurrencia libre de una variable en una fórmula* $\mathcal{F}$, *si ninguna de las variables de* **t**, *está acotada en* $\mathcal{F}$ *o bien si en* $\mathcal{F}$ *no hay cuantificadores con alguna variable de* **t**.

4) La variable **x** *es libre para la misma variable* **x**, *en cualquier fórmula.*

5) Cualquier término **t** *es libre para la variable* **x** *en una fórmula* $\mathcal{F}$, *si ésta no contiene ocurrencias libres de* **x**.

1.3 POSTULADOS PARA EL SISTEMA FORMAL

En esta sección introduciremos un sistema formal para la lógica de primer orden. Para ello presentamos una lista de postulados o axiomas y **reglas deductivas** *o* **de inferencia,** *las cuales dan al sistema formal la estructura de una* **teoría deductiva.** *Los axiomas son fórmulas que se eligen entre las* **formulas bien formadas.** *También hablaremos de un* **esquema axiomático,** *para referirnos a una expresión metamatemática a partir de la cual podemos obtener axiomas particulares.*

EJEMPLO 9

Sea $\mathcal{B} \Rightarrow (\mathcal{A} \vee \mathcal{B})$ *un esquema axiomático en donde* $\mathcal{A}$ *y* $\mathcal{B}$ *se denominan* **metavariables** *que toman valores dentro del conjunto de fórmulas bien formadas del sistema.*

Si la fórmula $p(x,y)$ *toma el lugar de* $\mathcal{A}$, *y la fórmula* $\neg p(x,y)$ *el lugar de* $\mathcal{B}$, *entonces obtendremos el siguiente axioma o* **instancia** *del esquema axiomático*

$$\neg p(x,y) \Rightarrow (p(x,y) \vee \neg p(x,y)).$$

De esta forma, el **esquema axiomático** *es, digamos, un dispositivo metamatemático para generar una infinidad de axiomas, que tienen un formato común.*

Los otros postulados de los que hablaremos se denominan **reglas de inferencia**. *Por ejemplo,*

$$\frac{\mathcal{A},\ \mathcal{A} \Rightarrow \mathcal{B}}{\mathcal{B},}$$

es un esquema que contiene tres expresiones metamatemáticas: $\mathcal{A}$, $\mathcal{A} \Rightarrow \mathcal{B}$, $\mathcal{B}$, *en donde* $\mathcal{A}$ *y* $\mathcal{B}$ *son nuevamente metavariables que pueden tomar como valores cualesquiera fórmulas del sistema.*

La semántica de la regla dice que la fórmula representada por la expresión que se encuentra debajo de la línea, puede ser inferida a partir del par de fórmulas representadas por las expresiones que se encuentran arriba de la línea. Es decir, que de $\mathcal{A}$ y $\mathcal{A} \Rightarrow \mathcal{B}$, podemos inferir $\mathcal{B}$.

EJEMPLO 10

Si para la regla enunciada, la fórmula $\neg p(x,y)$ *toma el lugar de* $\mathcal{A}$, *y la fórmula* $p(x,y) \vee \neg p(x,y)$ *toma el lugar de* $\mathcal{B}$, *entonces*

$$\frac{\neg p(x,y) \quad , \quad \neg p(x,y) \Rightarrow (\, p(x,y) \vee \neg p(x,y) \,)}{p(x,y) \vee \neg p(x,y)}$$

esta regla de inferencia es conocida como el **Modus Ponens.**

Como frecuentemente se hablará de que la variable x ocurre, ya sea libre o acotada en una fórmula $\mathcal{A}$, *utilizaremos como notación* $\mathcal{A}(x)$ *para indicar que la variable x puede ocurrir en la fórmula* $\mathcal{A}$ *y* $\mathcal{A}(t)$ *para representar la fórmula que se obtiene al sustituir las ocurrencias libres de x, en* $\mathcal{A}(x)$, *por t.*

Mostraremos ahora la lista completa de postulados de nuestro sistema formal.

Ax_0. $\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{A})$

Ax_1. $(\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow ((\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{C})) \Rightarrow (\mathcal{A} \Rightarrow \mathcal{C}))$

R_2. $\dfrac{\mathcal{A},\ \mathcal{A} \Rightarrow \mathcal{B}}{\mathcal{B}}$ **Modus Ponens**

Ax_3. $\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{A} \wedge \mathcal{B})$

Ax_4. $(\mathcal{A} \wedge \mathcal{B}) \Rightarrow \mathcal{A}$

Ax_5. $(\mathcal{A} \wedge \mathcal{B}) \Rightarrow \mathcal{B}$

Ax_6. $\mathcal{A} \Rightarrow (\mathcal{A} \vee \mathcal{B})$

Ax_7. $\mathcal{B} \Rightarrow (\mathcal{A} \vee \mathcal{B})$

Ax_8. $(\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow ((\mathcal{B} \Rightarrow \mathcal{C}) \Rightarrow (\mathcal{A} \vee \mathcal{B}) \Rightarrow \mathcal{C})$

Ax_9. $(\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow ((\mathcal{A} \Rightarrow \neg\mathcal{B}) \Rightarrow \neg\mathcal{A})$

Ax_10. $\neg(\neg\mathcal{A}) \Rightarrow \mathcal{A}$

R_11. $\dfrac{\mathcal{C} \Rightarrow \mathcal{A}(x)}{\mathcal{C} \Rightarrow \forall x \mathcal{A}(x)}$

Ax_12. $\forall x \mathcal{A}(x) \Rightarrow \mathcal{A}(t)$

Ax_13. $\mathcal{A}(t) \Rightarrow \exists x \mathcal{A}(x)$

R_14. $\dfrac{\mathcal{A}(x) \Rightarrow \mathcal{C}}{\exists x \mathcal{A}(x) \Rightarrow \mathcal{C}}$

Para los postulados del 1 al 10, $\mathcal{A}$, $\mathcal{B}$ y $\mathcal{C}$ son fórmulas bien formadas. Para los postulados del 11 al 14, x es una variable, $\mathcal{A}(x)$ es una fórmula en la que puede ocurrir x, $\mathcal{C}$ es una fórmula bien formada que no contiene libre a x y t es un término libre para x en $\mathcal{A}(x)$.

Para terminar esta sección veremos algunas definiciones que serán útiles más adelante.

DEFINICION (Axioma)

La clase de los axiomas está definida como sigue. Una fórmula bien formada $\mathcal{F}$, es un **axioma** *si tiene la forma de alguno de los esquemas axiomáticos* **Ax_n** *postulados en la lista antes presentada.*

EJEMPLO 11

1) *La siguiente fórmula*

$$p(x,y) \rightarrow (q(x) \rightarrow p(x,y))$$

es un axioma, ya que tiene la forma

$$\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{A})$$

correspondiente al primer esquema axiomático Ax_1.

2) *La fórmula*

$$\forall x p(x,y) \rightarrow p(a,y)$$

es un axioma, ya que tiene la forma

$$\forall x A(x) \rightarrow A(t)$$

correspondiente al esquema axiomático Ax_10 en la lista de postulados y además cumple con la condición de que el término **a** *es libre para la variable* **x** *en la fórmula* **p(x,y)**.

3) *La fórmula*

$$(q(x) \rightarrow q(x)) \rightarrow ((q(x) \rightarrow (q(x) \rightarrow q(x))) \rightarrow (q(x) \rightarrow q(x)))$$

también es un axioma ya que tiene la forma

$$(\mathcal{A} \rightarrow \mathcal{B}) \rightarrow ((\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{C})) \rightarrow (\mathcal{A} \rightarrow \mathcal{C}))$$

correspondiente al esquema axiomático Ax_1.

Nótese que las fórmulas que sustituirán a las **metavariables** $\mathcal{A}$, $\mathcal{B}$, *y* $\mathcal{C}$, *no tienen por que ser distintas.*

4) *Por último la fórmula*

$$p(x,y) \rightarrow (q(x,y) \rightarrow r(x,y))$$

no es *un axioma, ya que no tiene la forma de ningún miembro de la lista de postulados.*

Al resultado que obtenemos de sustituir o darle un valor a cada una de las metavariables de un esquema axiomático, se le conoce como una **instancia del esquema axiomático**. *Cuando el resultado no incluye metavariables, se le llama* **axioma**.

DEFINICION (Consecuencia Inmediata)

La relación de consecuencia inmediata está definida como sigue: Una fórmula $\mathcal{F}$, es una **consecuencia inmediata**, *de una o dos fórmulas $\mathcal{G}_1$ y/o $\mathcal{G}_2$, si $\mathcal{F}$ tiene la forma que se muestra abajo de la linea para los casos R_2, R_11 ó R_14 de la lista de postulados y $\mathcal{G}_1$ y/o $\mathcal{G}_2$ tienen la forma de las fórmulas que se muestra arriba de la linea, para cada uno de los casos antes mencionados, respectivamente.*

A los postulados R_2, R_11 y R_14 los llamaremos **reglas de inferencia**. *Las expresiones o fórmulas que sustituirán a las metavariables arriba de la linea llevarán el nombre de* **premisas** *(la primera y segunda, respectivamente) y la fórmula mostrada abajo de la linea será la* **conclusión** *de la* **aplicación** *de la regla de inferencia.*

La conclusión es una consecuencia inmediata de las premisas bajo la regla de inferencia.

1.4 DEDUCCION FORMAL

Hablaremos ahora de la noción de deducción formal, la cual nos ocupa dentro del editor que describiremos en los capítulos siguientes.

DEFINICION (Deducción Formal)

Una deducción formal de **F** *bajo las hipótesis* $H_1, H_2, \ldots, H_m$ *es una secuencia finita de fórmulas* $F_1, F_2, \ldots, F_n$ *con* $n > 0$ *tales que:*

1) F_n *es* **F**
2) *Cada* F_k *con* $k \leq n$ *es :*

- *alguna de las hipótesis* $H_1, H_2, \ldots, H_m$
- *un axioma o*
- *una consecuencia inmediata de la aplicación de una regla a fórmulas* F_i *y* F_j *anteriores a* F_n *en la secuencia.*

A la última fórmula **Fn** *de la secuencia se le llama la* **conclusión de la deducción.**

Como consecuencia de ésta definición, a continuación enunciaremos otras dos muy importantes, las de Fórmula demostrable y Fórmula deducible.

DEFINICION (Fórmula Deducible)

Se dice que una **fórmula F es deducible** *a partir de un conjunto de fórmulas* Γ *(llamado conjunto de hipótesis), si existe una deducción formal, de* **F** *a partir de* Γ.

DEFINICION (Fórmula Demostrable)

Cuando en una deducción el conjunto Γ *de hipótesis es vacío, entonces la fórmula que obtenemos como conclusión de la deducción es demostrable y se le llama* **Teorema**. *En este caso, la deducción es una* **demostración** *de* **F**.

Para indicar que una fórmula **A**, *es deducible o demostrable a partir de un conjuto de hipótesis* Γ, *utilizaremos la siguiente notación:*

$$\Gamma \vdash A$$

si se trata de un teorema **T**, *entonces:*

$$\vdash T$$

A continuación veremos algunos ejemplos de deducciones formales.

EJEMPLO 12

1.- *Supongamos, que nuestro conjunto* Γ *de hipótesis contiene las siguientes dos fórmulas* $\{ B, A \Rightarrow (B \Rightarrow C) \}$ *y que a partir de este conjunto de hipótesis queremos deducir la fórmula:* $A \Rightarrow C$. *Construimos entonces la siguiente* **deducción formal**:

1.- **B**	*hipótesis*
2.- **A ⇒ (B⇒C)**	*hipótesis*
3.- **B ⇒ (A⇒B)**	*axioma Ax_o*
4.- **A⇒B**	*regla R_2,apl.1,3*
5.- **(A⇒B) ⇒ ((A ⇒ (B⇒C)) ⇒ (A⇒C))**	*axioma Ax_1.*
6.- **(A ⇒ (B⇒C)) ⇒ (A⇒C)**	*regla R_2,apl.4,5*
7.- **A⇒C**	*regla R_2,apl.2,6*

Las fórmulas 1 y 2, son elementos de Γ, es decir hipótesis. La 3 es el axioma Ax_o, donde la metavariable $\mathcal{A}$ fué sustituida por la fórmula **B**, *y la metavariable $\mathcal{B}$ por la fórmula* **A**. *La 4 es el resultado de la aplicación de la regla de inferencia modus ponens a las fórmulas 1 y 3 de la secuencia. La 5, es el axioma Ax_1, en donde las sustituciones fueron las siguientes: $\mathcal{A}$ por* **A**, *$\mathcal{B}$ por* **B** *y $\mathcal{C}$ por* **C**. *Para obtener la fórmula 6, se aplicó la regla de inferencia modus ponens, a 4 y 5. Para llegar a la conclusión, se volvió a aplicar modus ponens, a las fórmulas 2 y 6 de la secuencia.*

Con esto nos podemos dar cuenta de que la fórmula **A⇒C**, *es deducible a partir de las hipótesis antes descritas, y el proceso de la demostración es una deducción formal.*

2.- *Sea* **x** *una variable,* **A(x)** *una fórmula bien formada y* **y** *una variable tal que:*

i) **y** *es libre para* **x** *en la fórmula* **A(x)**,

ii) **y** *no ocurre libre en* **A(x)**

iii) **A(y)** *es, por definición, el resultado de sustituir las ocurrencias libres de* **x** *en la fórmula* **A(x)**, *por el término* **y**.

Por (i), las ocurrencias de y *en* **A(y)**, *que fueron introducidas por la sustitución, son ocurrencias libres.*

Por (ii) no hay otras ocurrencias libres de y *en* **A(y)**, *más que las introducidas mediante la sustitución.*

Por lo tanto,

iv) **x** *es libre para* y *en* **A(y)**

v) **x** *no ocurre libre en* **A(y)**

vi) **A(x)** *es el resultado de sustituir las ocurrencias libres de* y, *en la fórmula* **A(y)**, *por la variable* **x**.

Bajo las consideraciones anteriores, si **C** *es una fórmula que no contiene libre a* y, *entonces la siguiente es una deducción de la fórmula* **C⇒∀xA(x)**, *a partir de la hipótesis* **C⇒A(y)**.

1.- **C ⇒ A(y)**	*hipótesis*
2.- **∀yA(y) ⇒ A(x)**	*axioma Ax_12*
3.- **∀yA(y) ⇒ ∀xA(x)**	*regla R_11,apl.2*
4.- **C ⇒ ∀yA(y)**	*regla R_11,apl.1*
5.- **[∀yA(y)⇒∀xA(x)] ⇒ [C ⇒ (∀yA(y)⇒∀xA(x))]**	*axioma Ax_0*
6.- **C ⇒ (∀yA(y)⇒∀xA(x))**	*regla R_2,apl.3,5*
7.- **(C⇒∀yA(y)) ⇒ [(C ⇒ (∀yA(y)⇒∀xA(x))) ⇒ (C⇒∀xA(x))]**	*axioma Ax_1*
8.- **[C ⇒ (∀yA(y)⇒∀xA(x))] ⇒ (C⇒∀xA(x))**	*regla R_2,apl.4,7*
9.- **C⇒∀xA(x)**	*regla R_2,apl.6,8*

la fórmula 1 es elemento del conjunto Γ de hipótesis. La 2 es una **instancia** *del axioma Ax_12, recordando que el término* y *es libre para la fórmula* **A(x)**. *La 3 es consecuencia inmediata de la fórmula 2 al aplicarle la regla R_11, ya que la fórmula* **∀yA(y)** *no contiene*

libre a **x** *(por (v)). La fórmula 4 es consecuencia inmediata de la 1, al aplicarle la regla R_11, puesto que la fórmula* **C** *no contiene libre a la variable* **y** *(por hipótesis). La fórmula 5 es simplemente una* **instancia** *del esquema axiomático Ax_6. La 6 es la consecuencia inmediata al aplicar la regla R_2 (modus ponens), a las fórmulas 3 y 5 respectivamente. La 7 es simplemente una* **instancia** *del axioma Ax_1. La fórmula 8 es consecuencia inmediata de aplicar la regla 2 (modus ponens), a las fórmulas 4 y 7 respectivamente. La última fórmula,* **la conclusión de la deducción**, *es consecuencia inmediata al aplicar la regla 2, a las fórmulas 4 y 7.*

Ahora mencionaremos un teorema muy importante referente a las deducciones, el **teorema de la deducción**. *Para ello daremos dos versiones, primero una sin considerar variables y otra con variables.*

No realizaremos la demostración del teorema, pero se puede encontrar en [5],[10], sólo daremos algunos ejemplos de aplicación en cada una de sus versiones para entenderlo.

Teorema de la deducción *(sin variables).*

Si tenemos que Γ, **A** $\vdash$ **B**, *entonces* $\Gamma \vdash$ **A**$\rightarrow$**B**. *Es decir, que si existe una deducción formal de la fórmula* **B** *a partir de un conjuto de hipótesis* Γ, *y de la fórmula* **A**, *entonces podemos deducir la fórmula* **A**$\rightarrow$**B**, *a partir del conjunto de hipótesis* Γ.

Es más, se puede contruir la deducción correspondiente de **A**$\rightarrow$**B** *a partir de* Γ.

EJEMPLO 13

En el ejemplo 12, inciso 1 de fórmula deducible que vimos anteriormente, se realizó la siguiente demostración:

$$\Gamma \vdash A \Rightarrow C$$

en donde $\Gamma = \{ B, A \Rightarrow (B \Rightarrow C) \}$. *Esta demostración la podemos realizar más corta si aplicamos el teorema de la deducción, para lo cual primero encontraremos una deducción formal de* **C**, *a partir de* **B**, **A** *y* $A \Rightarrow (B \Rightarrow C)$:

$$B, A, A \Rightarrow (B \Rightarrow C) \vdash C$$

Es decir, aumentamos **A** *como hipótesis y dejamos* **C** *en lugar de* $A \Rightarrow C$ *para demostrar.*

Veamos entonces como quedaría la deducción formal:

1.- **B**	*hipótesis*
2.- $A \Rightarrow (B \Rightarrow C)$	*hipótesis*
3.- **A**	*hipótesis*
4.- $B \Rightarrow C$	*regla R_2,apl.3,2*
5.- **C**	*regla R_2,apl.1,4*

Con esta deducción formal, más corta y fácil, hemos demostrado que la fórmula **C**, *es deducible a partir de* **A** *y de las hipótesis originales* Γ *o sea que:*

$$\Gamma, A \vdash C$$

si aplicamos el **teorema de la deducción**, *tendremos entonces lo siguiente:*

$$\Gamma \vdash A \Rightarrow C$$

que es lo que deseábamos demostrar.

Antes de presentar el teorema de la deducción con variables daremos tres definiciones.

DEFINICION (Dependencia)

Dada una deducción $A_1, A_2, \ldots A_k$ *a partir de un conjunto de hipótesis* $\Gamma = \{ H_1, H_2, \ldots, H_l \}$, *se dice que una fórmula* A_i *de la deducción* **depende** *de una hipótesis* $H_j \in \Gamma$ *si:*

1) A_i *es precisamente* H_j *o*

2) A_{i1} **depende** *de* H_j *y* A_i *es una consecuencia inmediata de* A_{i1} *ó de* A_{i1} *junto con otra* A_{i2}.

3) A_i **depende de** H_j *únicamente si se cumple 1) ó 2).*

EJEMPLO 14

En la siguiente deducción formal:

1.- **B**	*hipótesis*
2.- **A ⇒ (B⇒C)**	*hipótesis*
3.- **A**	*hipótesis*
4.- **B⇒C**	*regla R_2,apl.3,2*
5.- **C**	*regla R_2,apl.1,4*

la fórmula 3 **depende** *de la hipótesis* **A**, *puesto que se trata de la misma fórmula, y la fórmula 5* **depende** *de la hipótesis* **B**, *ya que 5 es consecuencia inmediata de las fórmulas 1 y 4, al aplicar la regla R_2, y la fórmula 1 depende de* **B** *ya que es precisamente* **B**.

DEFINICION (Variable Variada)

Se dice que una variable **y**, *es* **variada** *en una deducción para una hipótesis* $\mathbf{H}_i \in \Gamma$ *dada, si:*

a) **y** *ocurre libre en* $\mathbf{H}_i$ *y además*

b) si la deducción contiene una aplicación de las reglas R_11 ó R_14, con respecto a **y**, *a una fórmula que depende de* $\mathbf{H}_i$.

De otra manera se dice que **y**, **se mantiene constante** *en la deducción para la hipótesis* $\mathbf{H}_i$.

EJEMPLO 15

En el ejemplo 12.2 de deducción formal que vimos anteriormente, se realizó la siguiente demostración:

1.- $C \Rightarrow A(y)$	*hipótesis*
2.- $\forall y A(y) \Rightarrow A(x)$	*axioma* Ax_12
3.- $\forall y A(y) \Rightarrow \forall x A(x)$	*regla* R_11,*apl.*2
4.- $C \Rightarrow \forall y A(y)$	*regla* R_11,*apl.*1
5.- $[\forall y A(y) \Rightarrow \forall x A(x)] \Rightarrow [C \Rightarrow (\forall y A(y) \Rightarrow \forall x A(x))]$	*axioma* Ax_0
6.- $C \Rightarrow (\forall y A(y) \Rightarrow \forall x A(x))$	*regla* R_2,*apl.*3,5
7.- $(C \Rightarrow \forall y A(y)) \Rightarrow [(C \Rightarrow (\forall y A(y) \Rightarrow \forall x A(x))) \Rightarrow (C \Rightarrow \forall x A(x))]$	*axioma* Ax_1
8.- $[C \Rightarrow (\forall y A(y) \Rightarrow \forall x A(x))] \Rightarrow (C \Rightarrow \forall x A(x))$	*regla* R_2,*apl.*4,7
9.- $C \Rightarrow \forall x A(x)$	*regla* R_2,*apl.*6,8

En este caso, **y** *es* **variada** *para la hipótesis* **C ⇒ A(y)**, *ya que la variable* **y** *aparece libre en* **A(y)** *y además en el cuarto paso, fórmula 4, se aplicó la regla* R_11 *cuya variable utilizada fué precisamente* **y**, *en donde la premisa de la regla fué la fórmula 1, que* **depende** *de la hipótesis en cuestión.*

Por otro lado, la variable **x**, *no es* **variada** *en la deducción para la hipótesis* **C ⇒ A(y)**, *ya que la premisa que se utilizó al aplicar la regla* R_11, *con respecto a la variable* **x** *en el paso 3, fué la fórmula 2,* **∀yA(y) ⇒ A(x)**, *la cual no* **depende** *de la hipótesis. Por lo tanto, se dice que la variable* **x** **se mantiene constante** *en la deducción para la hipótesis* **C ⇒ A(y)**.

Teorema de la deducción *(con variables).*

Si existe una deducción **Γ, A ⊢ B**, *donde las variables libres de* **A**, *se* **mantienen contantes para** *la hipótesis* **A**, **entonces**

$$\Gamma \vdash A \Rightarrow B.$$

EJEMPLO 16

Tomemos nuevamente el ejemplo 12, inciso 2 de fórmula probable y de deducción formal que acabamos de revisar, en él se realizó la siguiente demostración:

$$\Gamma \vdash C \Rightarrow \forall x A(x)$$

donde $\Gamma = \{ C \Rightarrow A(y) \}$. *Esta demostración la podemos realizar más corta, si aplicamos el teorema de la deducción encontrando primero una deducción formal de* $\forall x A(x)$ *a partir de* Γ *y la fórmula* C:

$$\Gamma , C \vdash \forall x A(x)$$

Es decir, agregando la fórmula C *al conjunto* Γ *de hipótesis que originalmente se tenía.*

Veamos entonces como quedaría la deducción formal:

1.- $C \Rightarrow A(y)$	*hipótesis*
2.- C	*hipótesis*
3.- $C \Rightarrow \forall y A(y)$	*regla R_11,apl.1*
4.- $\forall y A(y)$	*regla R_2,apl.2,3*
5.- $\forall y A(y) \Rightarrow A(x)$	*axioma Ax_12*
6.- $\forall y A(y) \Rightarrow \forall x A(x)$	*regla R_11,apl.5*
7.- $\forall x A(x)$	*regla R_2,apl.4,6.*

Con esta deducción formal, hemos demostrado que la fórmula $\forall x A(x)$, *es deducible a partir de la* Γ *original y de* C, *en otras palabras*

$$\Gamma , C \vdash \forall x A(x)$$

y aplicando el teorema de la deducción, tendremos lo siguiente:

$$\Gamma \vdash C \rightarrow \forall x A(x),$$

que es lo que queríamos demostrar.

El teorema se puede aplicar, puesto que la variable **x**, *que es la variable implicada en la regla* $R_{\forall I}$ *para la obtención* $\forall x A(x)$, *se mantiene constante en la fórmula* **C**.

1.5 INTRODUCCION Y ELIMINACION DE SIMBOLOS LOGICOS

Con los siguientes teoremas, introducimos una serie de reglas deductivas o de inferencia.

	(Introducción)	(Eliminación)
(Implicación)	*Si* $\Gamma, A \vdash B$ *entonces* $\Gamma \vdash A \rightarrow B$	$A, A \rightarrow B \vdash B$. (Modus Ponens)
(Conjunción)	$A, B \vdash A \wedge B$	$A \wedge B \vdash A$ $A \wedge B \vdash B$
(Disyunción)	$A \vdash A \vee B$	Si $\Gamma, A \vdash C$ y $\Gamma, B \vdash C$ *entonces* $\Gamma, A \vee B \vdash C$ (Prueba por casos)
(Negación)	Si $\Gamma, A \vdash B$ y $\Gamma, \neg A \vdash B$ *entonces* $\Gamma \vdash \neg A$ (Reducción al absurdo)	$\neg \neg A \vdash A$ descargue de la doble implicación
(Generalidad)	$A(x) \stackrel{x}{\vdash} \forall x A(x)$	$\forall x A(x) \vdash A(t)$
(Existencia)	$A(t) \vdash \exists x A(x)$	Si $\Gamma(x), A(x) \vdash C$, *entonces* $\Gamma(x), \exists x A(x) \stackrel{x}{\vdash} C$,

La variable **x**, *que escribimos en la parte superior del simbolo* $\vdash$ *de dos de las reglas, indica la aplicación de la regla* $\forall$ *o* $\exists$, *con respecto a* **x**, *en la construcción del resultado de la deducción.*

Para los teoremas anteriores, **A**, **B**, **C** *y* **A(x)** *son fórmulas bien formadas,* **x** *es una variable y* **t** *es un término que satisfacen las siguientes condiciones:* **C** *no contiene libre a* **x**, **t** *es un término libre para* **x** *en* **A(x)** *y* Γ *es una lista de fórmulas.*

Para el cálculo proposicional, todas las reglas se cumplen puesto que no requieren de ninguna condición.

Para el cálculo de predicados, todas las reglas se cumplen, siempre y cuando en cada deducción las variables libres se mantengan **constantes** *para las hipótesis que serán descargas.*

Las demostraciones de estos teoremas tampoco se darán aquí puesto que no es el objetivo del trabajo pero puden encontrarse en [5] y [10].

CAPITULO II

DESCRIPCION DEL SISTEMA.

2.1 INTRODUCCION

En este capitulo, describiremos el funcionamiento general del editor lógico **Edilog**, *este capitulo será una guía para el usuario, ya que hablaremos con detalle acerca de cada una de las partes que componen el editor, tales como sus ventanas de trabajo, el menú (y sus comandos), de las ayudas que nos brindan, de los apoyos para llevar a cabo una deducción, etcétera.*

2.2 CONSIDERACIONES GENERALES DEL SISTEMA

Edilog *es un editor de fórmulas del cálculo de primer orden, que permite manejar conectivos lógicos como:*

¬ , ∨ , ∧ , => , <=>

y cuantificadores como:

V y **E**

El conjunto de axiomas manejado por **Edilog** *será el mismo que describimos en el capitulo anterior.*

El editor ofrece ciertas facilidades de ayuda a la generación de deducciones formales, por ejemplo, permitiendo la instanciación de axiomas de acuerdo a las sustituciones especificadas para las metavariables o la aplicación de reglas de ineferencia.

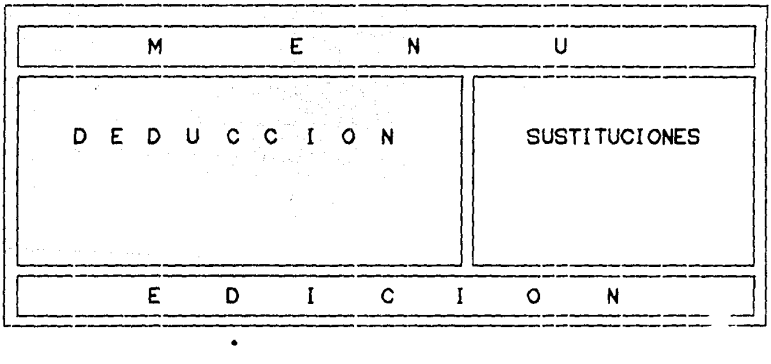
Edilog *consta de una serie de ventanas de edición, por ejemplo, para ver y accesar la deducción que se esta llevando a cabo, para editar fórmulas especificas; para describir términos y/o variables. Además, en* **Edilog** *se pueden realizar las tareas normales de un editor como: marcar, copiar, mover y borrar fórmulas.*

Por otro lado, el editor tiene interfases para accesar el disco y el sistema operativo, que permiten manipular deducciones en varias facetas, tales como cargar, grabar en disco o renombrar archivos que contengan deducciones; imprimir deducciones; cambiar de directorio; cambiar de formato al leer un archivo. Ademds tiene una opción para salir al sistema operativo y regresar a **Edilog**.

El editor ofrece también comandos de apoyo al usuario, que permiten orientar de alguna manera la deducción. Por ejemplo, dada una fórmula, se le puede pedir mostrar algunas equivalencias lógicas de dicha fórmula, nos puede sugerir también el tipo de fórmulas que podemos deducir a partir de una fórmula seleccionada o nos puede indicar que la fórmula seleccionada puede ser concluida a partir de la aplicación de ciertos axiomas.

2.3 ESTRUCTURA GENERAL DEL SISTEMA.

Al llamar a ejecución al editor, aparece una pantalla dividida en cuatro ventanas: **Menú, Deducción, Edición y Sustituciones,** *en donde cada ventana tiene su función particular, vedse figura II.1.*



FiguraIII.1. Ventana del editor

Describiremos globalmente cada una de estas cuatro ventanas.

Menú. *Esta ventana se encuentra en la parte superior de la pantalla, y contiene los comandos del menú principal del sistema. Estos permiten accesar el contexto externo del editor, consultar axiomas, apoyar una deducción, orientarse en el manejo del sistema, etcétera.*

Deducción. *Esta ventana es la que contiene realmente la deducción con la que se esta trabajando y en ella pueden agregarse o eliminarse fórmulas.*

Edición. *Esta ventana permite editar una fórmula para ser agregada a la lista de la deducción, ya sea como hipótesis o como resultado de la aplicación de un axioma o una regla de inferencia.*

Sustituciones. *Esta ventana se compone en realidad de varias ventanas pequeñas, que albergan fórmulas correspondientes a las metavariables que intervienen en un axioma. Esta serie de ventanitas se utilizan para apoyar la instanciación de un axioma o de una regla de inferencia, ya que para esto es necesario explicitar las fórmulas que sustituirán a cada una de las metavariables del axioma.*

2.4 DESCRIPCION DEL MENU.

Desde cualquier lugar dentro de **Edilog** *podemos accesar el menú. Para lograr esto, basta presionar la tecla* **Esc** *y estaremos automáticamente en la ventana del menú. En ella tendremos las siguientes opciones:*

Archivos,
Axiomas,
Apoyos y
Ayuda.

Estas pueden seleccionarse moviéndose con las flechas, hacia la izquierda o derecha y apretando **return**, *o bien, basta con teclear las letras que están realzadas para entrar en el submenú correspondiente de cada una de estas opciones. Por ejemplo, si se presiona la letra* **x** *entonces estaremos entrando al submenú de* **axiomas**, *que contiene las opciones de seleccionar y aplicar axiomas.*

La figura II.2 muestra gráficamente la ventana del menú.

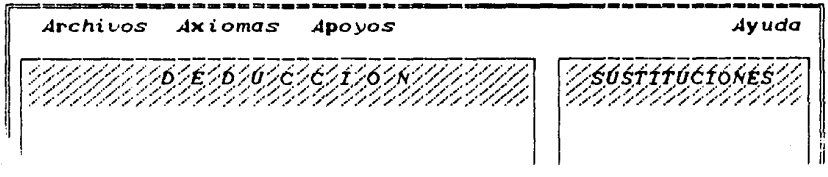


Figura II.2. Los comandos del menú

A continuación describiremos el funcionamiento de cada uno de los comandos y sus respectivos submenús.

ARCHIVOS

Para ver el submenú de **Archivos**, *nos movemos con las flechas* (← o →) *hasta iluminar la palabra* **Archivos** *y oprimimos* **Return** *o simplemente presionamos la letra* **r**.

Una vez dentro del submenú de **Archivos** *aparecerá una ventanita con las opciones correspondientes, que puede verse en la figura II.3.*

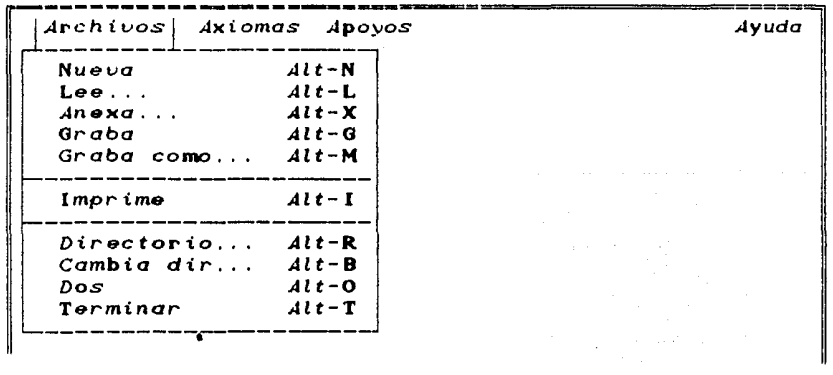


Figura II.3. Submenú de archivos

La opción **Archivos** *nos brinda dentro de su submenú nueve opciones para leer y grabar en disco:*

NUEVA: nueva deducción.
LEE: lee deducción.
ANEXA: agrega otra deducción
GRABA: graba deducción.
GRABA_COMO: graba una deducción con otro nombre.
IMPRIME: imprime una deducción.
DIRECTORIO: muestra el directorio de las deducciones.
CAMBIA FORMATO: cambia el formato del directorio.
DOS: hace un salto a DOS, y
TERMINA: finaliza el programa.

Para poder elegir alguna de esas opciones tenemos que movernos con las flechas (↑ o ↓) hasta iluminar la opción deseada y oprimir **Return** *o simplemente presionar la letra que esté realzada en cada palabra.*

Siempre es posible accesar cualquier opción del submenú, aún cuando el cursor no se encuentre dentro del menú de archivos. Es decir, si nos encontramos dentro de alguna ventana de trabajo, podemos realizar cualesquiera de las opciones de archivos, si presionamos simultáneamente la tecla **Alt** *y la letra realzada del comando. Por* ***ejemplo****, para accesar al directorio, basta apretar simultáneamente* **Alt** *y* **D.**

Describiremos ahora que hacen cada uno de los comandos mencionadas.

Nueva (Alt-N). *Al escoger ésta opción estaremos indicándole a* **Edilog** *que deseamos empezar una nueva deducción. Esto implica que se borrará la deducción actual en la ventana de deducción del editor, el sistema preguntará antes si la graba en disco para protegerla.*

Terminado este paso, desaparecerá el submenú de archivos y nos encontraremos en la ventana de deducción, en donde aparecerá el letrero de **Nuevo-archivo** *en la parte superior del marco de la ventana y el usuario estará en posición de crear una nueva deducción. (ver figura II.4)*

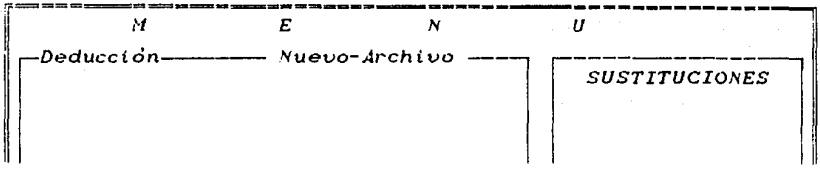


Figura II.4. Creación de una nueva deducción

Lee *(Alt-L): Esta opción nos sirve para leer del disco algún archivo que contenga una deducción ya editada y colocarla en la ventana de deducción, borrando la deducción actual. Nuevamente, antes de borrar la deducción actual, pregunta si la gnaba en disco.*

Después de ejecutado el comando, se borrará el submenú de archivos y el cursor aparecerá en la ventana de deducción, en donde: el nombre del archivo leído, por ejemplo "prueba.ded", aparecerá en la parte superior del marco de **Deducción.** *Vease figura II.5.*

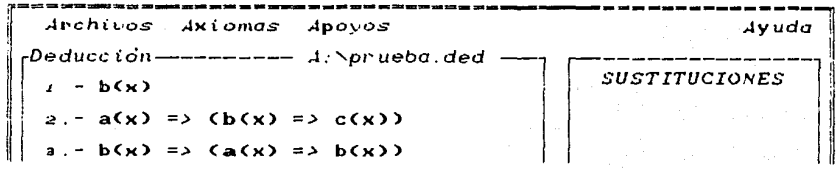


Figura II.5. Lectura de un archivo con una deducción

Edilog *verificará, antes de traer el archivo, que éste contenga realmente una deducción realizada por él mismo. En caso contrario, lo indicará al usuario y regresará a la ventana de trabajo en donde originalmente se encontraba el cursor.*

Anexa (Alt-X). Esta opción nos brinda la ventaja de agregar a la deducción actual otra deducción que se encuentre en disco, haciendo un "Merge" a partir de la posición del cursor en la ventana de deducción.

Graba (Alt-G). Con éste comando podemos grabar en disco un archivo que contenga la deducción actualmente editada. La deducción se grabará en el sub-directorio en el que se esté trabajando en ese momento bajo el nombre que se encuentre en la parte superior del marco de **Deducción.** *Si en el momento de grabar la deducción, aún no tiene nombre, entonces podemos darle uno utilizando la opción* **Graba_como** *(Alt-M).*

Graba_como (Alt-M) : Esta opción nos brinda la posibilidad de grabar en disco la deducción actual, con un nuevo nombre que nosotros queramos darle.

Imprime (Alt-I) : *Como su nombre lo indica, con este comando podemos mandar a imprimir una deducción que se encuentre en disco. Toda fórmula que está dentro de una deducción, es una hipótesis, o bien es la instancia de un axioma o el resultado de la aplicación de una regla de inferencia a fórmulas anteriores. Al motivo por el cual se encuentra cada fórmula en la deducción, se le llama* **justificación**. **Edilog** *preguntará si la impresión será con la justificación de cada fórmula o no.*

Si se responde **si** *a la opción que nos brinda* **Imprime**, *esto ocasionará que después de cada fórmula aparezca su justificación en la impresión.*

A continuación veremos un ejemplo de **impresión**. *Vamos a suponer que se tiene la deducción de la figura II.6, dentro del editor.*

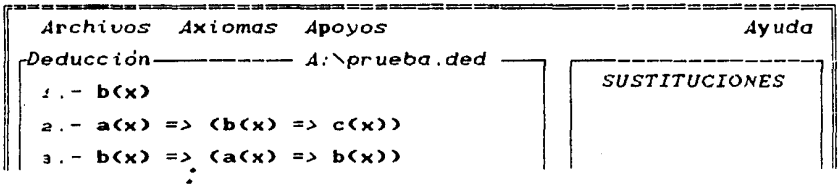


Figura II.6. *Deducción:* **prueba.ded**

La primera y segunda fórmulas son hipótesis y la tercera es una instancia del esquema axiomático Ax_o, con $\mathcal{A}$ / **b(x)** *y* $\mathcal{B}$ / **a(x)**.

Si se desea que **Edilog**, *imprima la justificación de cada fórmula, entonces la impresión se verá así:*

Archivo: prueba.pas

1.- b(x).
hipótesis.

2.- a(x) ⇒ (b(x) ⇒ c(x)).
hipótesis.

3.- b(x) ⇒ (a(x) ⇒ b(x)).
Axioma Ax_0 con
ϕ / b(x).
ℬ / a(x).

Cuando la fórmula dentro de la deducción, es el resultado de sustituir las metavariables de un esquema axiomático, como es el caso de la fórmula 3, **Edilog** *nos indica el esquema axiomático instanciado y cuales fueron las sustituciones de cada una de las metavariables del esquema axiomático aplicado.*

*Directorio (Alt-R): Esta opción nos permite ver el directorio de archivos, cuyo formato por omisión es "***.ded**", claro que si deseamos cambiarlo podemos hacerlo.*

Cambia_dir (Alt-B): Al elegir esta opción, podemos cambiar tanto el **path** *como el* **drive**, *del directorio que contiene o contendrá los archivos de deducciones.*

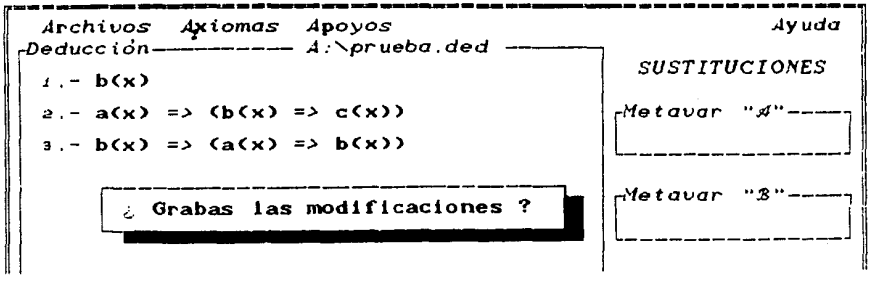
Dos (Alt-O) : Con esta opción salimos al sistema operativo DOS, para regresar al programa, tenemos que escribir la palabra EXIT desde el sistema operativo y oprimir return; de ésta forma se regresa a **Edilog** dentro de la ventana en donde estábamos antes de salir al DOS.

Termina (Alt-T) : Esta opción, como su nombre lo indica termina la ejecución del programa **Edilog** y regresa al sistema operativo.

Si el archivo que contiene la deducción actual no ha sido grabado desde su última modificación y nosotros deseamos terminar el programa, entonces **Edilog** preguntará al usuario si desea grabar las modificaciones realizadas; una vez que se respondió a la pregunta, se ejecuta la acción correspondiente y el programa termina. Ahora que si no se han hecho modificaciones, desde la última vez que se grabó, entonces aparecerá una ventanita en el centro de la pantalla, preguntando si realmente se desea terminar.

En la figura II.7, veremos un ejemplo gráfico, de cada uno de los dos casos, de ésta opción.

a.- Si el archivo que contiene la deducción no ha sido grabado



b.- *Si el archivo que contiene la deducción no ha sido modificado desde su última grabación.*

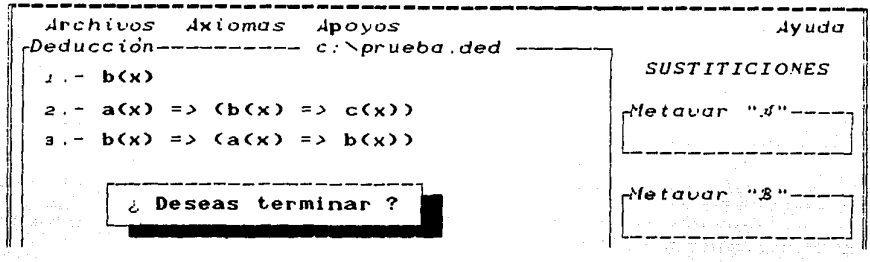


Figura II.7. Opciones de terminación

AXIOMAS

El segundo comando del menú, **Axiomas**, *nos ofrece dos opciones referentes al conjunto de axiomas y reglas de inferencia que maneja el editor. Las dos opciones son Selecciona y Aplica.*

Para poder utilizar alguna de estas dos opciones, primero tenemos que entrar al menú principal (presionando **Esc**), *una vez en el menú principal, tenemos que entrar al submenú de* **Axiomas**; *para ello nos movemos con las flechas "→" y "←", hasta iluminar la palabra* **Axiomas** *y oprimimos* **Return** *o simplemente presionamos la letra que esté realzada en la palabra, en este caso la letra* **x**.

Una vez dentro del submenú de **Axiomas**, *aparecerá una ventanita con las opciones antes mencionadas, como puede verse en la figura II.8.*

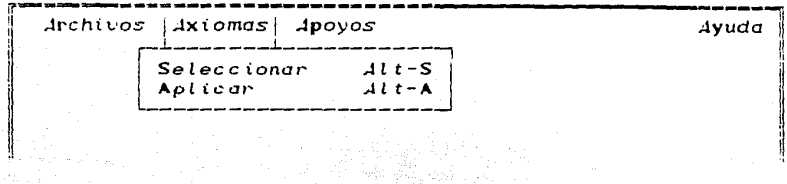


Figura II.8. Submenu de axiomas.

Como en el submenú de archivos, para poder elegir alguna de esas opciones tenemos que movernos con las flechas "↑" y "↓", hasta iluminar la opción deseada y oprimir **Return** *o simplemente presionar la letra* **S** *o* **A**. *También como en el caso anterior, los comandos selecciona y aplica pueden accesarse en forma directa desde cualquiera otra ventana de trabajo oprimiendo simultáneamente las teclas* **Alt** *y* **S** *o bien* **Alt** *y* **A**.

Selecciona *(Alt-S). Esta opción del submenú de* **Archivos**, *nos permite seleccionar un axioma o regla de inferencia, para obtener una instancia de él. Los axiomas son mostrados en una ventana al oprimir el comando* **selecciona**, *como puede verse en la figura II.9.*

Para elegir alguno de estos axiomas, necesitamos movernos con las flechas hasta el deseado y posteriormente oprimir **Return**. *Después de esto,* **Edilog** *nos colocará en la primera ventana de* **sustituciones** *para escribir allí la fórmula que reemplazará a la metavariable "𝒜". Presionando* **Return** *o* **Tab**, *nos moveremos a la segunda ventana de metavariables, para hacer lo mismo que en la primera, solo que ahora para la metavariable "ℬ" y así susecivamente.*

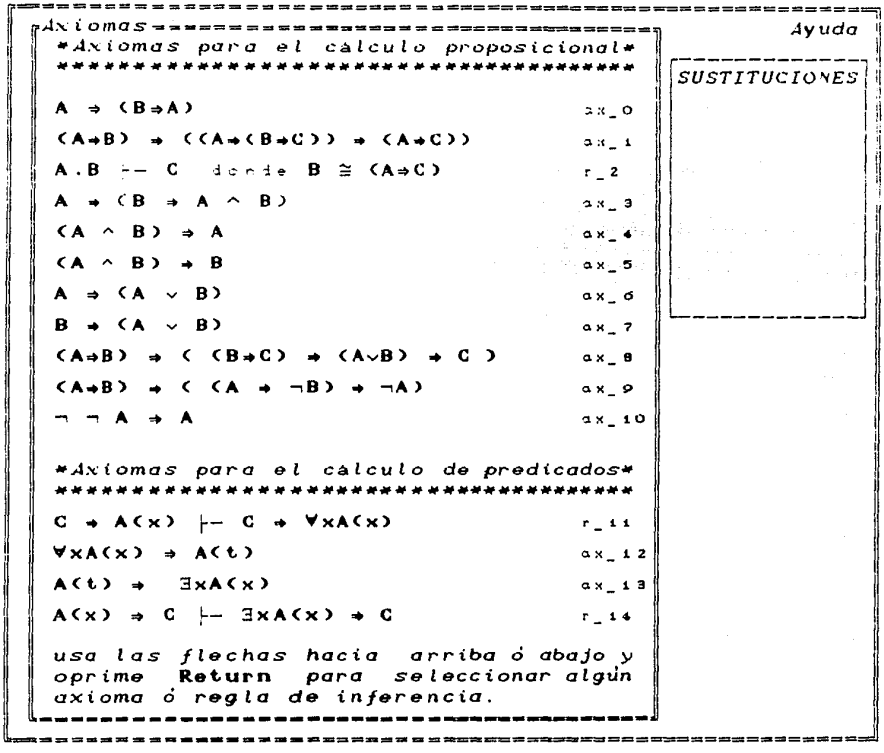


Figura II.9. Axiomas del editor con el comando selecciona

A continuación veremos un ejemplo para mostrar el funcionamiento de la ventana de sustitución. Para ello suponemos que se eligió la regla R_2 y que deseamos sustituir la metavariable "$\mathcal{A}$" por la fórmula 1 y la metavariable "$\mathcal{B} \cong \mathcal{A} \Rightarrow \mathcal{B}$" por la fórmula 3. De esta manera la pantalla de **Edilog** *quedaría como en la figura II.10.*

Edilog *deja el resultado de la aplicación en la ventana de* **Edición**, *como se ve gráficamente en la figura II.11.*

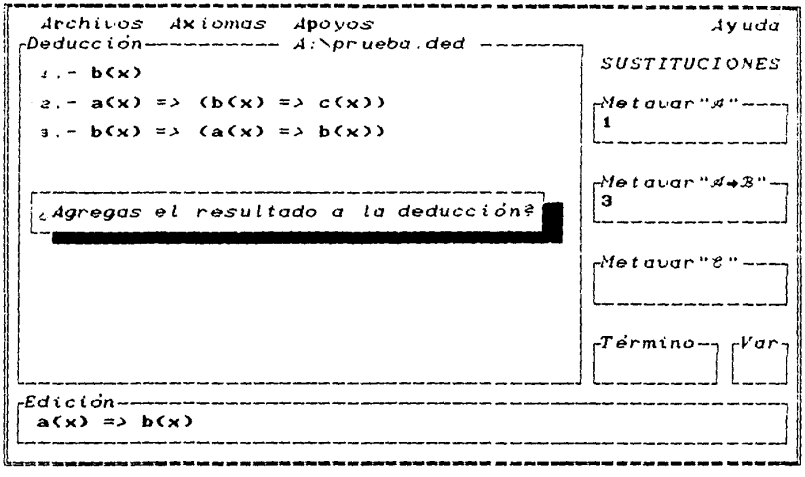


Figura II.11. Aplicación de la regla R_2, bajo la sustitución A / la fórmula 1 y A→B / la fórmula 3.

Una vez instanciado el axioma, **Edilog** *nos preguntará si agregamos el resultado de la aplicación a la lista de la deducción.*

:

APOYOS

El comando **Apoyo** *del menú nos ofrece cinco opciones para apoyar una deducción:* **Concluye, Deriva, Justifica. Equivalencias** *y* **Término libre.** *Dichas opciones podemos verlas en la ventana de la figura II.12.*

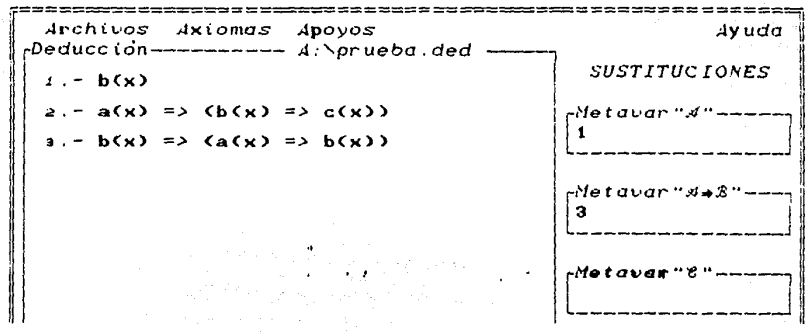


Figura II.10. Sustitución de metavariables en una regla.

Como el axioma instanciado es la regla R_2, y en éste no se utiliza la metavariable "C", entonces no será necesario escribir ninguna fórmula en la ventana correspondiente a dicha metavariable.

Una vez descritas las fórmulas que sustituirán a las metavariables en un axioma o en una regla de inferencia, tenemos que indicarle a **Edilog** *que ya podemos proceder a la aplicación del axioma elegido presionando* **Alt** *y* **A**.

Si deseamos interrumpir esta opción, presionamos la tecla **Esc** *en cualquier paso de la selección.*

Aplicar (Alt-A). *Al tomar esta opción nosotros ya tuvimos que haber elegido un axioma y las sustituciones para las metavariables. Así, simplemente le indicamos a* **Edilog** *que ya deseamos aplicar el axioma o regla de inferencia escogida presionando las teclas Alt y A simultáneamente o dentro del menú de axiomas elegimos la opción de* **Aplicar**.

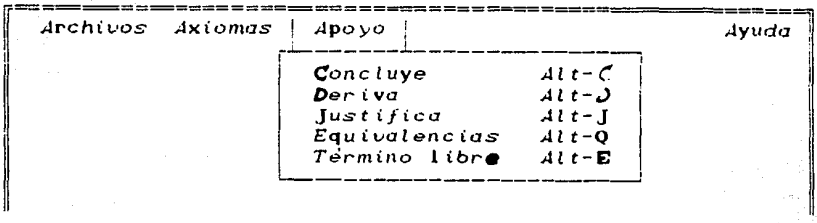


Figura II.12. Opciones del comando **Apoyo**.

Las opciones de **Concluye, Deriva. Equivalencias** *y* **Término libre,** *se aplican a la fórmula apuntada por el cursor en cualquiera de las ventanas de:* **Deducción, Edición** o **Sustituciones.**

A diferencia de las demás, la opción **Justifica** *sólo se aplica a la fórmula apuntada por el cursor dentro de la ventana de deducción o bien a la fórmula donde estuvo el cursor la última vez, en caso de que en ese momento no estemos dentro de la ventana de deducción.*

Como en los casos anteriores, siempre es posible accesar cualquier opción de éste submenú, aún cuando uno no se encuentre dentro de él, si presionamos simultáneamente la tecla **Alt** *y alguna de las letras:* **Y, V, J, Q, L.**

Concluye *(Alt-Y). Este comando nos va a sugerir qué axiomas permiten concluir una fórmula dada y, dependiendo del axioma sugerido, la sustitución de metavariables necesaria para poder deducir dicha fórmula. Mostramos esto con un ejemplo gráfico en la figura II.13. Supongamos que el cursor apunta a la fórmula* **a(x)→c(x)**, *que se encuentra en la ventana de edición.*

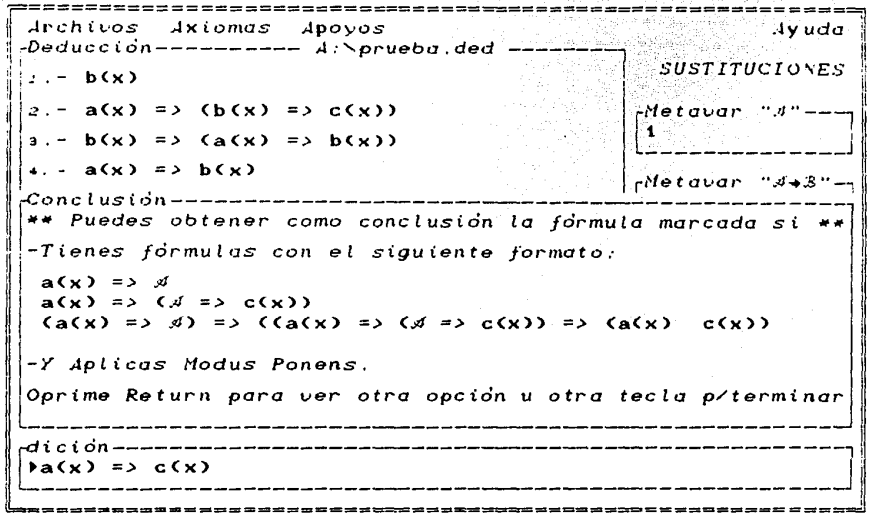


Figura II.13. Aplicación de la opción concluye a la fórmula que se encuentra en la ventana de edición.

Deriva *(Alt-V). Este comando nos va a indicar que tipo de fórmulas son derivables a partir de la fórmula apuntada por el cursor, y bajo que axiomas o regla de inferencia. Veamos un ejemplo: En este caso la fórmula apuntada por el cursor será la cuarta de la lista de la deducción* **a(x)→b(x)** *y la respuesta es que utilizando el axioma Ax_7:*

(a(x) => b(x)) => ((a(x) => (b(x) => A)) => (a(x) => A))

con las sustituciones A/a(x), B/b(x) y C/A, donde A puede ser cualquier fórmula. Posteriormente aplicando la regla R_2, podemos derivar:

(a(x) => (b(x) => $\mathcal{A}$)) => (a(x) => $\mathcal{A}$).

Veamos como apareceria en la pantalla para ello observemos la figura II.14.

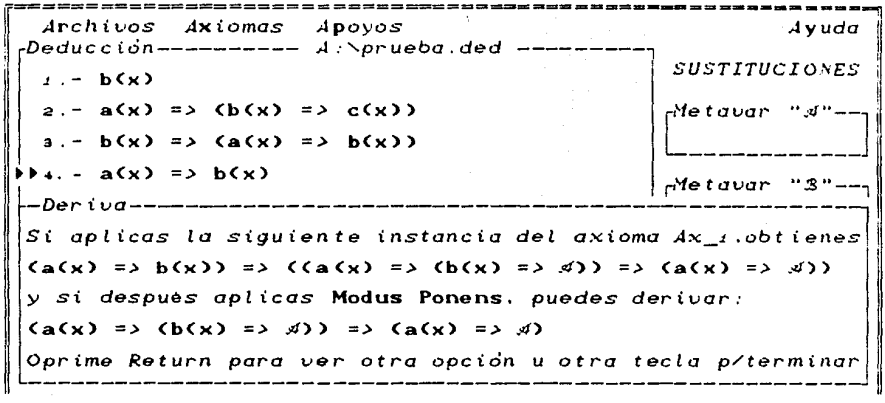


Figura II.14. Aplicación de la opción **Deriva** *a la fórmula* **a(x) → c(x)**, *en la ventana de deducción.*

Si después de ver esta respuesta queremos otra opción presionamos **Return** *y* **Edilog** *nos mostrará otras alternativas.*

Justifica *(Alt-J). Con este comando* **Edilog** *nos propociona la justificación de la fórmula apuntanda por el cursor dentro de la ventana de deducción, aunque en ese momento nos encontremos en cualquier otra ventana. Es decir, que si en cualquier momento deseamos saber cómo se obtuvo una fórmula dentro de la deducción, tenemos que estar seguros de que el cursor esté apuntando a la fórmula en cuestión.*

Veamos en la figura II.15, una justificación de la fórmula 4, de la deducción que aparece en la pantalla.

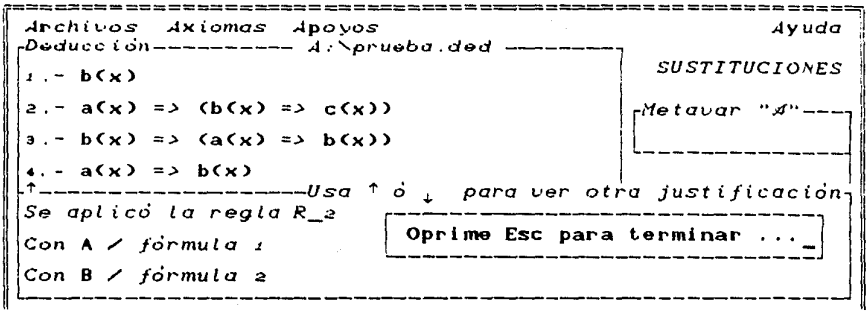


Figura II.15. Justificación de la fórmula 3 de la deducción

Equivanencias (Alt-Q). *Eligiendo ésta opción,* **Edilog** *nos ofrece una serie de fórmulas lógicamente equivalentes a la marcada por la posición del cursor.*

Veamos un ejemplo gráfico, en la figura II.16, la fórmula apuntada por el cursor es **a(x) → b(x)** *en la ventana de deducciones y en la ventana de edición aparece su equivalente,* **¬ a(x) ∨ b(x)**. *Si se desea una nueva fórmula que sea lógicamente equivalente, se presiona cualquier tecla. Si se desea utilizar esta fórmula equivalente que aparece en la ventana de edición, hay que presionar la tecla* **Esc** *y así podemos usarla como una hipótesis. La fórmula apuntada por el cursor podría haber estado también en las ventanas de deducción o bien de las metavariables "Ø", metavar "Ƀ" o metavar "Ɛ".*

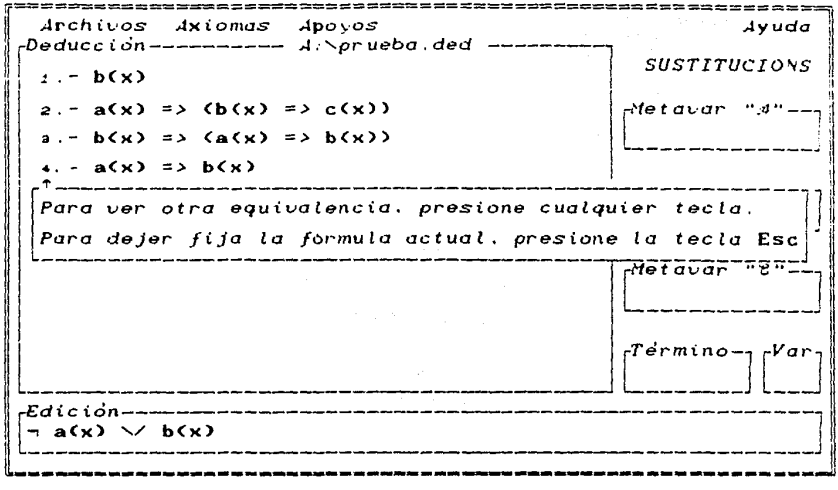


Figura II.16. Equivalencia lógica de **a(x) => b(x)**.

AYUDA

El comando **Ayuda** *del menú, nos ofrece orientación acerca de los comandos y funcionamiento del Editor. Al seleccionar ayuda aparece un submenú como en la figura II.17.*

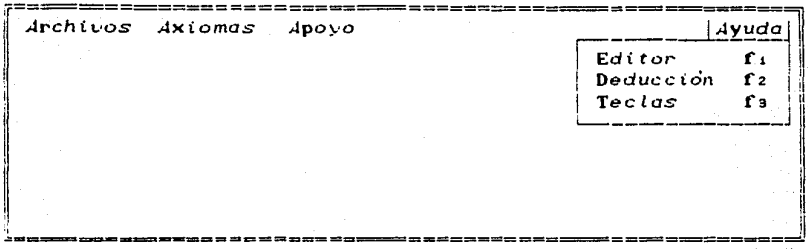


Figura II.17. Susbmenú del comando **Ayuda**

Cada subcomando del menú se accesa en la misma forma que los comandos de los submenús anteriores. Los comandos de este submenú ofrecen ayuda a diferentes niveles del editor. Vamos a describir lo que nos proporciona cada uno de ellos.

Editor (Alt-E). *Este comando nos orienta acerca del funcionamiento general del editor, de cómo movermos dentro de él, qué podemos y qué no podemos hacer, en fin es una guía para aprovechar al máximo las capacidades de edición*

Deducción (Alt-D). *Con esta opción, podemos tener una idea de lo pasos más elementales para empezar una deducción en* **Edilog.** *La ayuda nos indica paso a paso todo el procedimiento necesario para poder generar una deducción completa.*

Teclas (Alt-C). *Esta opción nos despliega una tabla con el funcionamiento de cada una de las teclas.*

A continuación veremos las teclas más importantes.

1.- Combinación de la tecla **Ctrl** y alguna otra.

Ctl-B Borra de la deducción, la fórmula donde se encuentra el cursor.

Ctl-D Brinca a la ventana de **Deducción.**

Ctl-E Brinca a la ventana de **Edición.**

Ctl-H Agrega la fórmula que se encuentra en la ventana de **Edición.** como una hipótesis, al final de la deducción.

Ctl-S Brinca a la primer sub-ventana de **Sustituciones.**

2.- Combinación de la tecla **Alt**, con alguna otra tecla.

Alt-A Lleva a cabo la aplicación, de un axioma o regla de inferencia.

Alt-B Cambia el directorio

Alt-C Concluye

Alt-D Deriva

Alt-E Término Libre.

Alt-G Graba un arhivo.

Alt-I Imprime la deducción.

Alt-J Justifica una fórmula.

Alt-L Lee un archivo.

Alt-M Graba como.

Alt-N Nueva Deducción.

Alt-O Pasa al sistema operativo MS-DOS.

Alt-Q Equivalencias.

Alt-R Despliega el directorio.

Alt-S Despliega la lista de axiomas.

Alt-T Terminar.

Alt-X Anexa archivo.

3.- las teclas **fn**, *en general, despliegan o muestran lo que se indica*

f1	*breve ayuda de los comandos de edición.*
f2	*orientación para poder realizar una deducción.*
f3	*tablas que indican la función de las principales teclas.*
f4	*simbolo* **V**.
f5	*simbolo* **E**.
f6	*simbolo* **=>**.
f7	*simbolo* **<=>**.
f8	*simbolo* **/**.
f9	*simbolo* **\/**.
f10	*simbolo* **¬**.

2.5 DESCRIPCION DE CADA VENTANA.

Como se mencionó anteriormente, la pantalla se divide en cuatro ventanas **Menú, Deducción, Edición y Sustituciones**, *ésta última se subdivide a su vez en 5 ventanitas como podemos ver en en la figura II.18.*

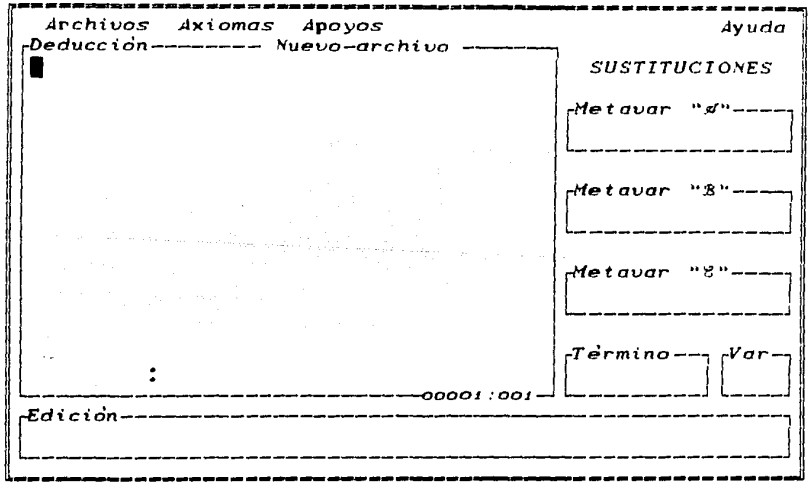
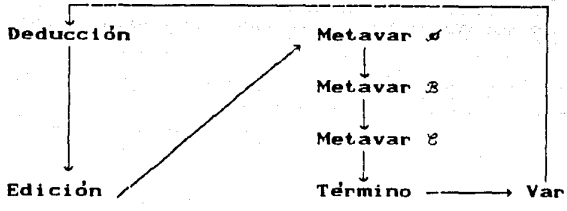


Figura II.18. Ventanas del editor

La ventana del menú nos permite movernos dentro de ciertos comandos. Las otras ventanas son realmente las que nos permiten editar una deducción.

Este conjuto de ventanas, **Deducción, Edición y Sustituciones,** *están de alguna manera numeradas u ordenadas, dicho orden es el siguiente:*



Para poder movernos a través de cada una de estas ventanas, es necesario presionar la tecla **Tab** *o la tecla* **Shif-tab**. *Así que, si utilizamos la tecla* **Tab**, *nos novemos a la ventana siguiente a la actual, es decir, hacia "adelante". Por ejemplo, si el cursor está en la ventana de edición, con un* **tab** *pasará a la Metavar "𝒜". Si utilizamos la combinacion de las teclas* **Shif** *y* **tab**, *entonces nos movemos a la ventana anterior a la actual, hacia "atrás". Por ejemplo, de la ventana Metavar "𝒞", pasaría a Metavar "ℬ".*

En todas las ventanas podemos mover y borrar fórmulas, además es posible copiar fórmulas de una ventana a otra, excepto en la ventana de deducción, donde no se permite agregar fórmulas mediante copiado directo. Esta restricción se debe a que las fórmulas que se encuentran en la ventana de deducción, tienen un orden secuencial y una justificación, por lo que no es posible agregar fórmulas si no conllevan el orden y la justificación adecuada.

La posición de las fórmulas dentro de la deducción es algo muy importante, ya que hay fórmulas que dependen de otras y en su justificación se hace referencia a la posición de las fórmulas de las cuales depende. Además, cuando se da de alta una fórmula ya sea como instancia de un axioma o una hipótesis o aplicación de una regla de inferencia a fórmulas anteriores, el resultado de tal aplicación debe agregarse integramente a la deducción. Por esta razón no está permitido borrarle ni agregarle fracciones a las fórmulas, ya que si éstas se modifican se alteraría su justificación dentro de la deducción.

A continuación pasaremos a describir el funcionamiento de cada una de las ventanas antes mencionadas.

DESCRIPCION DE LA VENTANA DE DEDUCCION.

Cuando el programa inicia su ejecución, el usuario se encuentra directamente en la ventana de **Deducción** *con un cursor ancho"*█*" en el extremo superior izquierdo.*

En la parte superior del marco de Deducción aparece el nombre del archivo que contiene la deducción actual, y en la parte inferior derecha del marco aparece la posicion del cursor, que varia conforme como nos vayamos moviendo dentro del editor. Véase la figura II.18.

Como se vió anteriormente, para cambiarse de una ventana a otra es suficiente con presionar la tecla **Tab** *o* **Shif-tab**. *Por el contrario, si deseamos quedarnos en deducción, presionando cualquier otra tecla distinta el cursor cambiará de forma, de "*█*" a "_", para indicarnos que estamos dentro de ventana que contiene la deducción.*

Una vez dentro de la ventana de deducción, podemos movernos con las flechas ↑, ←, →, ↓, o bien con **Pg-Up, Pg-Dn, Home, End,** *etc. El editor nos permitirá agregar las fórmulas que se deseen, también nos permitirá borrar alguna fórmula apuntada por el cursor, pero lo que* **no** *permitirá, es modificar las fórmulas que ya estén escritas en la deducción.*

Para salir de la ventana de Deducción, *basta con cambiarse de ventana ya sea presionando las teclas* **Tab** *o* **Shif-tab**, *o bien, presionando simultaneamente las teclas Ctrl-E, en cuyo caso seria a la ventana de* **Edición**, *o presionando Ctrl-M, que brincaria a la primera ventana de las* **Sustituciones**, *es decir, a la ventana de la metavariable "Ø".*

DESCRIPCION DE LA VENTANA DE EDICION

Esta ventana es muy importante ya que en ella podemos editar una fórmula, para, por ejemplo, agregarla luego a la deducción.

Esta ventana funciona como un pequeño editor independiente de las demás ventanas, por lo cual podemos escribir la fórmula deseada utilizando las teclas diseñadas para ello. Estas teclas son las que normalmente utiliza cualquier editor: **Del, Ins,** *Flechas (↑, ←, →, ↓),* **Home, End**, *y la típica que borra el caracter que se encuentra antes del cursor.*

También se utlizan las siguientes teclas para desplegar en la pantalla los siguientes símbolos:

f4 para **V**
f5 para **E**
f6 para **=>**
f7 para **<=>**
f8 para **/**
f9 para **\/**
f10 para **¬**

Una vez editada la fórmula, se puede hacer lo siguiente:

1) *Agregar la fórmula a la deducción como una hipótesis (presionando* **Ctrl** *y* **H**).
2) *Preguntar a* **Edilog** *que equivalencias lógicas tiene esta fórmula (***Alt** *y* **Q**).
3) *Saber si un término es libre para cierta variable en esa fórmula. Para ello el término y la variable se escriben en las ventanas de* **Término** *y* **Var**, *de la ventana de* **Sustituciones**.(**Alt** *y* **E**).

Esta ventana también es utilizada por **Edilog** *para escribir el resultado de aplicar un axioma o una regla de inferencia. Después, la fórmula se puede copiar a la ventana de* **Deducción**, *si así se desea.*

DESCRIPCION DE LAS VENTANAS DE SUSTITUCION

Esta ventana que consta de cinco ventanas auxiliares, sirve para llevar a cabo sustituciones en un axioma. Por ejemplo, si deseamos aplicar un axioma o una regla de inferencia, las METAVARIABLES $\mathcal{A}$, $\mathcal{B}$, *y* $\mathcal{C}$ *de la fórmula deben instanciarse*

respectivamente en las ventanas de metavar "X", metavar "&" y metavar "?".

Si deseamos saber si un término es libre para una variable en cierta fórmula, el término y la variable deben escribirse en las ventanas de **Término y Var** *respectivamente y la fórmula en cuestión será la que se encuentra apuntada por el cursor en ese momento.*

Todas estas ventanas de sustitución, permiten también editar fórmulas de la misma manera que en la ventana de edición. De hecho la única ventana que no permite la edición de fórmulas es la de deducción.

2.6 UTILERIAS GENERALES.

Dentro del editor existen una serie de comandos de edición que permite mover fórmulas de un lado a otro. Por ejemplo, para **Mover** *una fórmula, uno debe primero marcarla, colocándose al principio de la misma y moverse luego hacia la derecha oprimiendo* **Ctrl** *y → Esto ocasionará que el texto se empiece a marcar, cambiando de color. La tecla → puede ser tambien una de las siguientes: ↑, →, ←, ↓, o bien la tecla:* **Home, End, PgUp, PgDn.**

Una vez marcado el bloque, o la fórmula, si se oprimen las teclas **Shift** *y* **C** *(la letra* **C** *mayúscula) el bloque será copiado a un* **buffer.** *De manera inmediata el bloque será desmarcado sin alterar el texto original. Una vez que el bloque esta copiado en el* **buffer,** *se puede colocar en otro lado, simplemente oprimiendo simultaneamente las teclas* **Shift** *e* **ins.**

Si marcamos un bloque y en lugar de oprimir **shift** *y* **C,** *se*

presionan las teclas **Shift** y **del**, esto ocasionará que el bloque se copie en el **buffer** y se borre de donde fué marcado

Si después de marcar el bloque se oprime únicamente la tecla **Del**, el texto marcado se borrará, pero no se copiará en el **buffer**. Lo que había antes en el **buffer** quedará intacto.

Recordamos que **Edilog** no permitirá borrar, ni insertar, ni mover fórmulas mediante el uso de bloques dentro de la ventana de deducción.

AGREGAR FORMULAS A LA LISTA DE LA DEDUCCION.

Si deseamos agregar alguna fórmula a la lista de la deducción tenemos dos caminos.

a) Editar una fórmula en la ventana de **Edición** y oprimir las teclas **Ctrl-H** para agregarla a la deducción, verificando que la fórmula esté correctamente escrita (fórmula bien formada). Toda fórmula editada directamente por el usuario será agregada a la deducción únicamente como una **Hipotesis**.

b) **Aplicar** un axioma o una regla de inferencia a alguna(s) fórmula(s) de la deducción. El resultado de esta aplicación se escribirá en la ventana de **Edición**. Una vez allí, **Edilog** nos pregunta si queremos agregarla a la lista de la deducción. Si decidimos agregarla, lo podemos hacer con **Ctrl-C**, su justificación en la deducción será el resultado de aplicar determinado axioma o regla de inferencia a cierta(s) fórmula(s) anteriore(s) en la deducción.

BORRAR FORMULAS.

Hay ocasiones en las que por equivocación agregamos una fórmula a nuestra deducción y luego nos damos cuenta que realmente no nos sirvió, y nos gustaría borrarla. Para hacer esto, necesitamos colocarnos en el renglón en el que se encuentra la fórmula y presionar las teclas **Ctrl** *y* **B**.

Cada vez que se intente borrar una fórmula, **Edilog** *verificará que la fórmula en cuestión no sea una fórmula que haya sido utilizada para deducir otra fórmula. Si* **Edilog** *permite borrar una fórmula, renumerará las fórmulas subsecuentes a ella en la deducción y por lo tanto* **rectificará** *tambien su justificación.*

Creemos que con las acciones de **Agregar y Borrar** *fórmulas son suficientes para obtener un buen funcionamiento y control del editor y que cuando se añade una fórmula a la lista de la deducción podemos tener un buen control de porqué se agregó (ya sea una hipótesis o la aplicación de algún axioma o regla de inferencia).*

Ahora bien, si deseamos borrar una fórmula, dicha fórmula será borrada con todo y su referencia o justificación. Naturalmente que **Edilog** *checará que la fórmula no esté referida por ninguna de las fórmulas anteriores a ella en la deducción. Véase la figura II.19.*

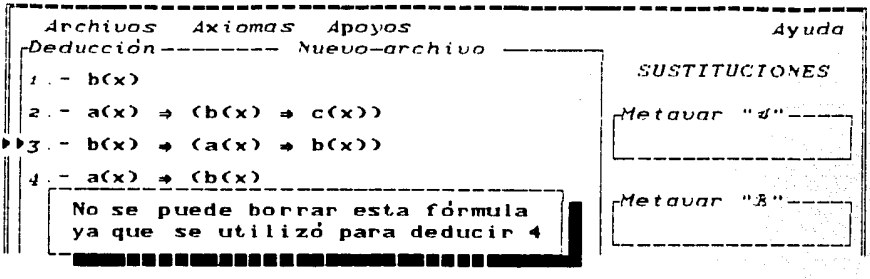


Figura II.19. Mensaje que nos anuncia el impediento para borrar una formula.

Tener éste control acerca de cuando borrar fórmulas y cuando no es necesario para obtener una deducción correcta y justificada, ya que si se permitiera borrar o mover fracciones de fórmulas arbitrariamente, sería muy fácil perder la coherencia en una deducción.

2.7 DESCRIPCION DE UNA SESION.

Describiremos ahora una sesión completa con el editor. Para ejecutar **Edilog**, *escribimos la palabra "edilog" enseguida del prompt del sistema operativo* **>edilog**.

De inmediato, aparecerá una presentación del sistema y después las ventanas de **Menú, Deducción, Fórmula y Sustitución.**

Lo primero que podemos hacer para iniciar una sesión, es colocar nuestras hipótesis, si es que las hay. Es decir, supongamos

que queremos que las fórmulas **a(x)** *y* **c(x)** *sean hipótesis, entonces nos movemos a la ventana de* **Edición** *con* **Ctrl** *y* **E** *y allí escribimos nuestra primera hipótesis* **a(x)**. *Una vez escrita, presionamos* **Ctrl** *y* **H** *para agregarla a la ventana de deducción como una hipótesis. Después hacemos lo mismo con* **c(x)** *y obtenemos en la ventana de la deducción lo que se ve en la figura II.2o.*

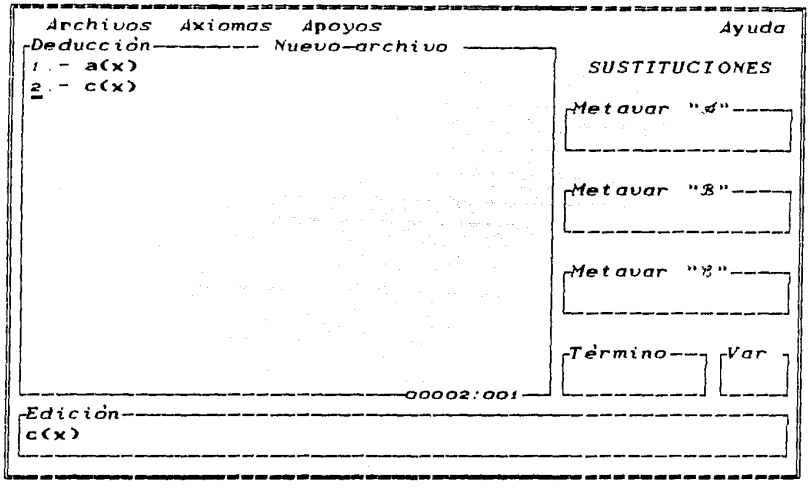


Figura II.2o. Inicio de una deducción.

Ahora, si deseamos instanciar un esquema axiomático, tenemos que movernos al menú principal oprimendo **Esc**, *para escoger la opción* **Seleccionar** *que se encuentra en el submenú de* **Axiomas**. *Así aparecerá una ventana con los axiomas. Supongamos que seleccionamos el esquema axiomático Ax_o.*

$$A \rightarrow (B \rightarrow A)$$

Como consecuencia de la opción **seleccionar**, *el cursor se moverá a la primer ventana de* **sustituciones**, *para que allí se escriba la fórmula que sustituirá a la metavariable "ℵ", del axioma. Si deseamos utlizar la fórmula 1 de la deducción, es suficiente con escribir el número 1 en metavar "ℵ".*

Posteriormente pasamos a la ventana metavar "ℜ" y allí colocamos la fórmula que sustituirá a dicha metavariable, por ejemplo la fórmula **p(x).**

Una vez escritas las instancias de "ℵ" y "ℜ", presionamos **Alt.** *y* **A** *para aplicar el axioma que fue seleccionado con las sustituciones capturadas. En seguida nos preguntará si deseamos agregar el resultado a la deducción. Vease la figura II.21.*

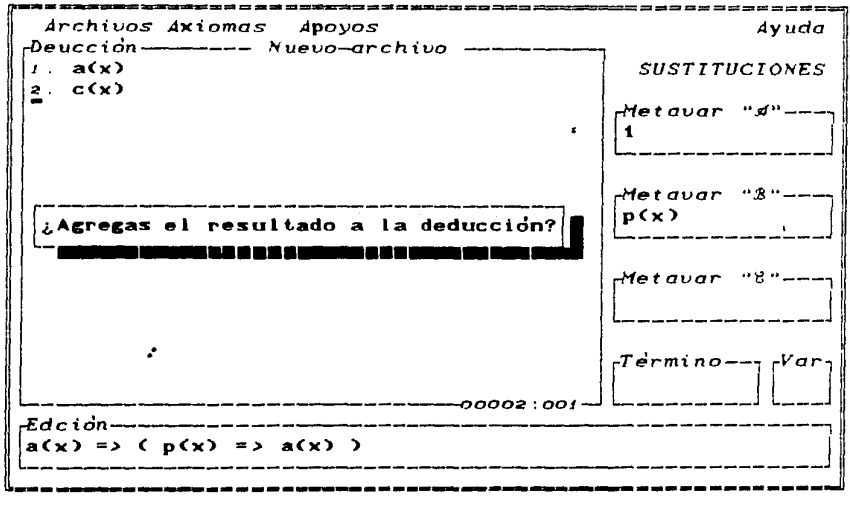


Figura II.21. Como seleccionar y aplicar el esquema axiomático Ax_0.

La pregunta podría contestarse afirmativamente y entonces **Edilog** *agregará el resultado de la instancia al final de la deducción quedando como puede verse en la figura II.22.*

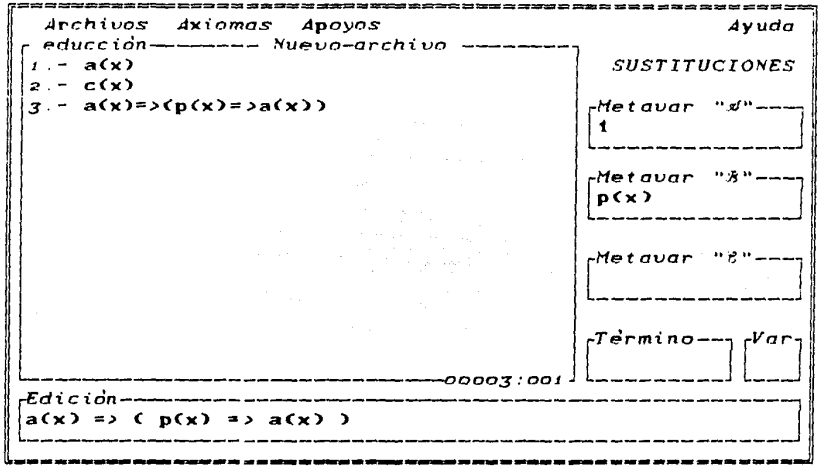


Figura II.22. Resultado de agregar a la deducción una instancia el esquema axiomático Ax_o.

*Para continuar con el ejemplo, aplicaremos la regla R_2 (***modus ponens***), a las fórmulas 1 y 3 de la deducción. Para indicarle a* **Edilog** *que deseamos utilizar tal regla, seleccionamos la regla R_2 y posteriormente en las ventanas de sustitución, indicamos las fórmulas que serán utilizadas para llevar a cabo tal apilicación. Por último, realizaremos la aplicación.*

Supongamos que en nuestro ejemplo aplicamos R_2 (modus ponens) a las fórmulas 1 y 3 de la deducción, lo cual lo indicamos en las ventanas de metavar α y metavar β respectivamente como se puede ver en la figura II.23.

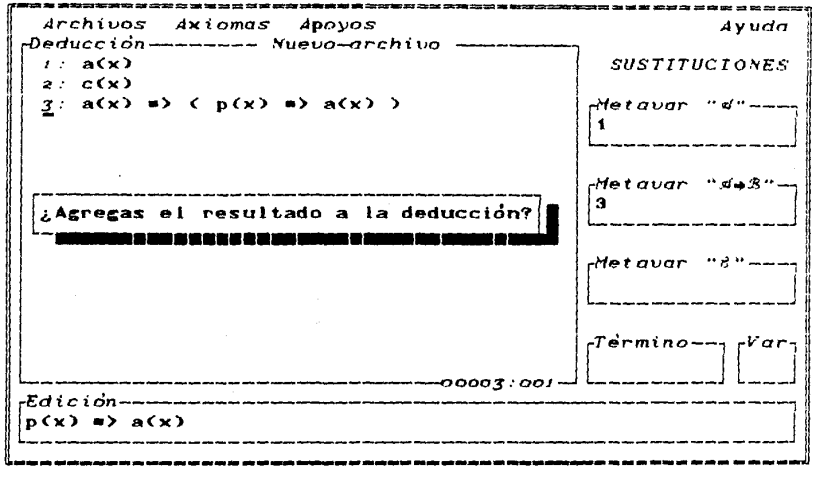


Figura II.23. Aplicación del **modus pones** *a las fórmulas 1 y 3 de la deducción.*

A la pregunta, si queremos agregar el resultado de la aplicación del modus ponens, que aparece en la ventana de edición, contestaremos afirmativamente. Así, el resultado que obtendremos puede verse en la figura II.24.

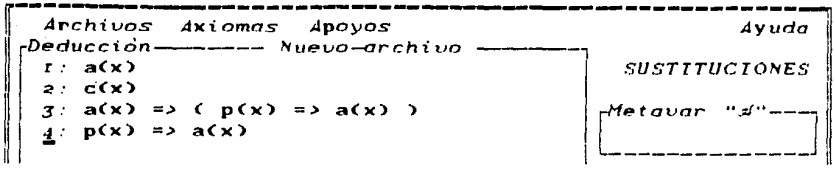


Figura II.24. Resultado de agregar la fórmula obtenida en II.23 a la deducción

Si deseamos grabar la pequeña decucción realizada hasta el momento, primero debemos darle un nombre al archivo que guardará dicha deducción, supongamos que el nombre del archivo será "deducc-1". Para ello utilizamos la opción de **Graba_como**, *presionado las teclas* **Alt** *y* **M**, *y edilog nos preguntará por el nombre del archivo, le damos deducc-1 y con esto se grabará la deducción en el disco bajo el nombre "deducc-1.ded".*

Para terminar la sesión, simplemente presionando **Alt** *y* **T**, *y edilog preguntará si realmente deseamos terminar, a lo que responderemos afirmativamente.*

CAPITULO III

LA PROGRAMACION DE EDILOG

3.1 I N T R O D U C C I O N.

En este capitulo, se describirá la programación en **Prolog** *del sistema* **Edilog**.

Primero hablaremos de la sintaxis utilizada para las fórmulas en **Edilog**, *posteriomente se mencionará la estructura general del programa y por último se describirán los predicados de mayor relevancia en el apoyo al desarrollo de una deducción.*

La razón por la cual **Edilog** *fué programado en prolog, radica en las facilidades de manipulación simbólica que el lenguaje ofrece y su recursividad intrinseca.*

El intérprete y compilador utlizado para este programa fué **Arity Prolog**.

3.2 SINTAXIS UTILIZADA EN EL PROGRAMA.

Cuando hablamos de sintaxis utilizada en el programa, nos referimos a la forma correcta de expresar una fórmula, un término, una variable, etc. y todo lo que tiene que ver con una deducción formal.

Edilog *maneja dos sintaxis diferentes, una interna con la que trabaja el programa para llevar a cabo la manipulación de los símbolos y fórmulas, y una externa que es la que el programa le permite utilizar al usuario para escribir fórmulas.*

A continuación describiremos cada una de las dos sintaxis.

SINTAXIS EXTERNA (Exclusiva para el usuario).

- *Para el manejo de los cuantificadores, se utilizan la letras* **V** *y* **E**, *que definimos de la siguiente manera:*

  - *Una fórmula del tipo* **∀x F**, *para toda* **x F**, *se escribe con el símbolo* **V** (**f4** *en el teclado*):

    Vx F

  - *Una fórmula del tipo* **∃x F**, *existe* **x** *tal que* **F**, *se escribe con el símbolo* **E** (**f5** *en el teclado*):

    Ex F

- *Para los conectivos lógicos, la notación es la siguiente:*

  - *Una conjunción se escribe con el símbolo* **/** (**f6** *en el teclado*).

    F /\ G

- *Una disyunción se escribe con el simbolo \/ (f7 en el teclado).*

F \/ G

- *Una implicación se escribe con el simbolo => (f8 en el teclado)*

F => G

- *Una Equivalencia se escribe con el simbolo <=> (f9 en el teclado)*

F <=> G

- *Para negar una fórmula se usa el simbolo ¬ (f10 en el teclado)*

¬F

:

SINTAXIS INTERNA (Exclusiva para el programa).

- *Para manejar los cuantificadores dentro del programa se utilizaron los predicados:* **e_x** *y* **p_t**, *que definimos de la siguiente manera:*

- *Una fórmula del tipo* $\forall x$ F, *internamente se manejará como un predicado de dos argumentos* x *y* F.

  p_t(x, F).

- *Una fórmula del tipo* $\exists x$ F, *internamente se manejará como un predicado de dos argumentos* x *y* F.

  e_x(x, F).

El primer argumento del predicado, en este caso x, *nos indica cual es la variable del cuantificador y el segundo argumento del predicado nos indica la fórmula que se encuentra bajo el alcance de la variable del cuantificador.*

Para los conectivos lógicos, la representación interna es la siguiente:

- *Una fórmula del tipo* $\neg$ F, *internamente se representará con el símbolo* .

  #F.

- *La conjunción y la disyunción se manejarán internamente en forma infija y con el mismo símbolo que en la sintaxis externa :* $\wedge$ *y* $\vee$ *respectivamente.*

  F $\vee$ G. y F $\wedge$ G.

- *La equivalencia y la implicación, también se representarán en forma infija y con los mismos*

simbolos que en la sintaxis externa: <=> y =>.

F => G. y F <=> G.

3.3 E S T R U C T U R A G E N E R A L

La estructura general del programa está constituida por dos grupos de predicados. El primero corresponde a los predicados relacionados con todo el ambiente de manejo del editor, este módulo se opcupa de crear las ventanas del ambiente para la edición.

El otro grupo de predicados, es el que está relacionado con los operadores del editor como son: la instanciación de los axiomas, el remombramiento de variables, el chequeo de las fórmulas, etcétera.

A continuación pasaremos a detallar cada uno de los grupos de predicados antes mencionados, empezando por los predicados del ambiente.

PREDICADOS DEL AMBIENTE.

Este grupo de predicados tiene como función realizar todo el ambiente de interacción con el usuario como son: las ventanas de trabajo y el menú.

Los dos predicados fundamentales para el ambiente son: **Dialog** *y* **Menú**. *Estos predicados son independientes, tanto en sus estructuras como en sus funciones dentro del programa.*

A continuación describiremos a grandes razgos cada uno de estos predicados, empezaremos por el predicado **dialog.**

DIALOG.

El predicado **Dialog**, *es un predicado intrinseco de* **Arity Prolog**, *que permite crear, ejecutar y controlar las ventanas de la deducción; inclusive, puede activar también el menú principal del programa. Este predicado es el motor del ambiente de* **Edilog**, *maneja una ventana marco que a su vez contiene una serie de ventanas de trabajo, o controles, llamados* **dialog boxes**, *que permiten realizar las funciones del editor.*

Un **Dialog boxes**, *puede verse como un conjuto de predicados que definen las ventanas de trabajo. Cada ventana tiene asociado un predicado* **Ctrl** *que define su propia forma de manejo. El formato de un* **Ctrl** *para definir una ventana es el siguiente:*

ctrl(Funcion,M,T,(Xo,Yo),(X1,Y1),(At,Br),F,Nombre).

El número de argumentos del **Ctrl** *puede variar de acuerdo al tipo de función definida en el primer parámetro, esta descripción corresponde al tipo de función* **edit_box** *que explicaremos más adelante.*

Función : *Este parámetro nos indica la forma de edición de la ventana: un renglón, múltiples renglones, menú, etcétera, y tiene valores predefinidos en arity.*

M: *Este parámetro nos indica, si al usuario se le permite moverse dentro de la ventana (t=si, o=no).*

T: *En este parámetro podemos escribir un texto que aparecerá como título del control; es opcional.*

(X0,Y0): *Con este par representamos las coordenadas de la esquina superior izquierda de la ventana del editor*

(X1,Y1): *Con este par repesentamos las coordenadas de la esquina inferior derecha de la ventana del editor.*

(At,Br): *Con este par indicamos los colores de la ventana (texto y fondo) y el color del marco.*

F: *En este parámetro inicamos si se le permite al usuario modificar directamente el contenido del texto (***r** *indica únicamente lectura, y* **rw** *indica lectura y escritura).*

Nombre: *Con este parámetro le podemos asociar un nombre interno a la ventana para diferenciarla de otra con las mismas características.*

El primer parámetro, **Función**, *varía dependiendo de como se desee utilizar la ventana de trabajo y puede tomar los siguientes valores:*

edit_box, efield, choice_box, text, ...

Por ejemplo, si al crear una de las ventanas de trabajo del **Dialog boxes**, *uno le asocia la función* **edit_box**, *esta ventana podrá automaticamente hacer uso de funciones de edición sobre múltiples renglones. Si por el contrario, uno le asocia la función* **efield**, *la ventana creada permitirá editar solamente sobre un renglón.*

En el caso de la función **edit_box**, *hay que asociar además un nombre a la ventana creada. En el caso de* **efield** *sólo habrá que ordenar las ventanas que utilizarán ésta función, y se tomará encuenta el orden que se le dió a cada ventana, para diferenciar una de otra.*

La función **choice_ box**, *asociada a una ventana, permite manejar un menú dentro de la ventana. En caso de haber varias ventanas, para diferenciarlas unas de otras es suficiente con el orden que se les dé, como en el caso de* **efield**.

La función **text**, *cuando se asocia a una ventana, permite definir un texto fijo dentro de la ventana.*

Por ejemplo, para definir el texto de las ventanas de sustituciones, se utilizó en **Edilog** *el predicado* **Ctrl** *de la siguiente manera:*

```
ctrl(text,0,$ S U S T I T U C I O N E S $,(2,51),15,27).
```

Una vez que se activa el predicado **Dialog**, *éste se encuentra en modo de lectura continua y dependiendo de la tecla que se presione puede hacer lo siguiente:*

- **Pasar del ambiente de ventanas de trabajo al menú y viceversa.**
- **Procesar información en las ventanas.**
- **Moverse a través de las ventanas.**

El lugar donde se encuentra el cursor al activar dialog, es el lugar donde se espera recibir la información del usuario.

En la figura III.1, mostramos el funcionamiento general de **Dialog**.

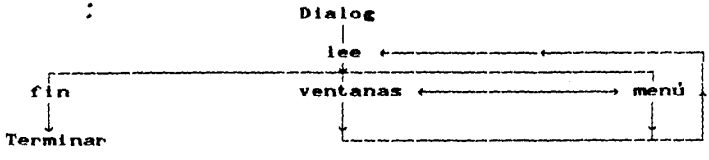


figura III.1. Funcionamiento General de Dialog

Como se mencionó anteriormente, el predicado **Dialog** *es el motor del programa y para realizar sus funciones se apoya en otros predicados predefinidos del intérprete de* **Arity Prolog**. *La estructura de las definiciones para ejecutar el predicado* **Dialog** *en* **Edilog**, *es la siguiente:*

```
begin_dialog(principal,'',(0,0),(23,79),(112,-112),47,popup).


      ctrl(edit_box,1,$$,(1,0),(18,50),(112,127),r,nuevo).
            begin_choices(nuevo).
            $ $.
            end_choices(nuevo).

      ctrl(efield,1,_,(19,0),(15,127),75,$$).
      ctrl(efield,1,_,(4,51),(15,127),24,$$).
      ctrl(efield,1,_,(8,51),(15,127),24,$$).
      ctrl(efield,1,_,(12,51),(15,127),24,$$).
      ctrl(efield,1,_,(16,51),(15,127),18,$$).
      ctrl(efield,1,_,(16,72),(15,127),3,$$).

      ctrl(text,0,$Edición$,(19,1),112,7).
      ctrl(text,0,$Metavar "A"$,(4,52),112,11).
      ctrl(text,0,$Metavar "B"$,(8,52),112,11).
      ctrl(text,0,$Metavar "C"$,(12,52),112,11).
      ctrl(text,0,$Término$,(16,52),112,7).
      ctrl(text,0,$Var$,(16,73),112,3).
      ctrl(text,0,$Nuevo-archivo ----------$,(1,20),127,30).
      ctrl(text,0,$ S U S T I T U C I O N E S $,(2,51),15,27).
      ctrl(text,0,$ Deducción $,(1,1),112,11).


end_dialog(principal).
```

Los predicados **begin_dialog** *y* **end_dialog**, *permiten definir un ambiente de trabajo a base de ventanas, las cuales deben definirse en forma de lista como se vió anteriormente.*

En las ventanas de trabajo los letreros están definidos por los **Ctrl's**, *ya que dependiendo del primer parámetro, la ventana se puede ver como un editor de pantalla (***Edit_box***), un editor sobre una sola linea (***Efield***), una ventana en la cual se pueden manejar opciones (***Choice_box***) o escribir un titulo permanente (***Text***). Así el primer* **Ctrl** *define la ventana de deducción, y ésta utiliza los predicados* **begin_choices**, **end_choices** *cuyo parámetro nos indica el nombre del archivo que contendrá el texto del editor, el texto inicial se debe indicar entre los dos predicados antes mencionados, en este caso el texto inicial es un espacio y lo representamos así:* **$ $**.

El siguiente grupo de **Ctrl's**, *definen las ventanas de trabajo como son la de* **edición**, *las tres de* **metavar**, *la de* **término** *y la* **var**. *Con los parámetros que tienen nos indican la posición, el tamaño, el color del texto, fondo y marco de cada ventana.*

El último grupo de **Ctr's**, *definen los titulos de cada una de las ventanas. Con los parámetros que tienen nos indican la posición, el tamaño, el color del texto y fondo de cada titulo.*

Para ejecutar el predicado **Dialog** *se debe realizar una llamada del siguiente tipo:*

```
dialog_run(P,F).
```

en donde el parámetro **P** *es un identificador para el* **dialog** *y el segundo parámetro* **F**, *es el nombre del predicado que nos ayuda a controlar o redefinir las teclas utilizadas.*

En **Edilog**, *la llamada a dialog es como sigue:*

```
dialog_run(principal,prin).
```

Al ejecutarse **dialog_run**, *se activan cada una de las ventanas que se definieron en* **begin_dialog** *y el programa espera una lectura.*

En la siguiente sección veremos como **prin** *define las teclas a utilizar.*

LA FUNCION PRIN.

Este predicado corresponde al valor del segundo parámetro **F** *del predicado* **dialog_run** *y tiene la siguiente definición:*

prin(C,K).

El parámetro **C** *contiene los números* **ASCII** *de la(s) letra(s) presionada(s) y el parámetro* **K** *nos indica el nombre bajo el cual se identifica la ventana de* **Dialog**.

Prin es el predicado más importante de **Dialog**, *ya que por él pasan todas las opciones o tareas que se deseen realizar. Veamos algunos ejemplos de definiciones de la función* **prin:**

a) **prin(char(0,38),principal):- !,leer(Key).**

indica que si se presionaron las teclas **Alt y L** *(codigos 0 y 38), entonces se ejecuta el predicado leer que permite al usuario ver el directorio y cargar un archivo.*

b) **prin(char(27,1),Key):- !,mns(principal).**

indica que si se presionó la tecla **esc** *(codigos 27 y 1), entonces se pase al menú y lo active.*

c) **prin(char(13,28),Key):- !,**
which_control(Ctrl),
cambia_foco(Ctrl,Key).

indica que si se presionó la tecla **return**, *se pase a la siguiente ventana.*

PREDICADOS DE APOYO AL PREDICADO DIALOG.

El predicado **dialog** *se apoya también en otros predicados, auxiliares. Algunos de ellos serán mostrados a continuación.*

a) **which_control(Ctrl).**

Este predicado nos indica en que ventana se encuentra el cursor.

b) **ef_set_text(OldText,NewText).**

Este predicado reemplaza un texto, **OldTex**, *por otro,* **NewText**, *que fué definido con un* **ctrl** *del tipo* **editfied.**

c) **eb_get_pos((Row,Col),(TRow,TCol)).**

Este predicado nos indica la posición, (Trow,Tcol), del cursor con respecto a la ventana principal de **dialog** *y la otra, (Row,Col), con respecto a la ventana de deducción.*

d) **eb_insert(LineNum,Text).**

Por último, este predicado inserta un texto, **Text**, *en una linea,* **LineNum**, *dentro de la ventana de deducción.*

Ahora describiremos a grandes razgos, el otro predicado base para el ambiente, el predicado del menú.

MENU.

El predicado del **Menú**, *maneja una serie de comandos que tienen asociadas sus propias ventanas.*

La ventana principal del menú nos presenta cuatro comandos: **Archivos, Axiomas, Apoyos, Ayuda**. *Cada comando posee a su vez un submenú. Cuando uno de los comandos es seleccionado, se despliegan las opciones de su submenú correspondiente.*

Una vez que se activa el predicado **Menú**, *éste siempre se encuentra en lectura continua y dependiendo de la tecla que se presione puede hacer lo siguiente:*

- **Regresar a la ventana de trabajo (esc).**
- **Ejecutar una opción (Return).**
- **Seleccionar una opción (flechas).**

A continuación mostraremos con el esquema de la figura III.2, el funcionamiento general del predicado **Menú**.

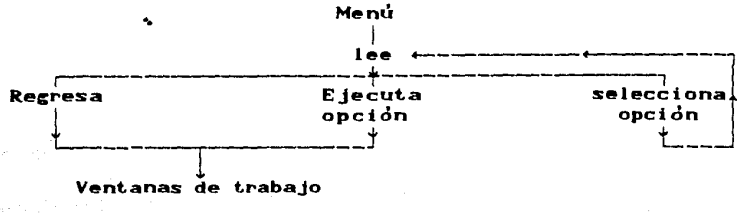


figura III.2. Funcionamiento General de Menú

La dedinición del predicado que fué utilizado para ejecutar el menú en el programa **Edilog** *es el que aparece a continuación:*

```
mns(Key):-
     send_menu_msg(activate(menu,(0,0)),Ret_val),
     case (Ret_val
          Ret_val = selecc  -> axiomas(Key),
          Ret_val = termina -> terminar(Key),
          Ret_val = aplica  -> aplicar(principal),
          Ret_val = dos     -> shell,
          Ret_val = graba   -> grabar(Key),
          Ret_val = lee     -> leer(Key),
          Ret_val = anexa   -> anexar(Key),
          Ret_val = nueva   -> nuevo(Key),
          Ret_val = g_como  -> grabar_como(Key),
          Ret_val = impri   -> impresion,
          Ret_val = dir     -> dirs(Key),
          Ret_val = cdir    -> cdirs(Key),
          Ret_val = hedi    -> ay(1),
          Ret_val = justi   -> altjust(Key),
          Ret_val = equiv   -> equivalencias(Key),
          Ret_val = deriva  -> derivacion(Key),
          Ret_val = concluye-> conclusion(Key),
          Ret_val = htec    -> ay(2),
          Ret_val = hded    -> ay(3)
          ).
```

El predicado **mns**, *lo primero que hace es activar el menú y después con un* **case** *analiza la opción que fué elegida al terminar la ejecución del menú.*

Para activar el menú, usamos el siguiente predicado.

```
send_menu_msg(activate(menu,(0,0)),Ret_val).
```

Este predicado, predefinido en **Arity prolog**, *activa la ventana del menú y nos permite movernos a través de él para seleccionar alguna opción del menú. Este predicado regresa en la variable* **Ret_val** *una clave asociada a la opción elegida y dependiendo de la clave que regrese el menú, se realiza la acción correspondiente con el* **case** *que se describe en el predicado* **mns**.

Como vimos anteriormente, con el predicado **send_menu_msg** *activamos el menú, pero para activar dicho menú, primero temenos que definir su estructura y con ella los elementos que lo componen; para hacer esto* **Arity prolog** *nos proporciona predicados que nos permite hacer un menú; los predicados son los siguientes:* **begin_menu, end_menu, item** *y algumos otros que no utilizaremos. Primero veremos la estructura del menú y después la explicaremos.*

```
begin_menu(menu,78,colors((79,48),(79,112),(123,27),(75,79))).
    item($A~rchivos$,[
        item($~Nueva            Alt-N $,nueva),
        item($~Lee...           Alt-L $,lee),
        item($Ane~xa...         Alt-X $,anexa),
        item($~Graba            Alt-G $,graba),
        item($Graba co~mo...    Alt-M $,g_como),
        break,
        item($~Imprime...       Alt-I $,impri),
        break,
        item($Directo~rio...    Alt-R $,dir),
        item($Cam~bia dir...    Alt-B $,cdir),
        item($D~OS              Alt-O $,dos),
        item($~Terminar         Alt-T $,termina)]).
```

```
        item($A~xiomas$,[
              item($~Seleccionar      Alt $,selecc),
              item($~Aplicar           Alt-A $,aplica)]).
        item($A~poyos$,[
              item($~Concluye         Alt-C $,concluye),
              item($~Deriva           Alt-D $,deriva),
              item($~Justifica        Alt-J $,justi),
              item($~Equivalencias    Alt-E $,equiv)]).
        item(right($~Ayuda$),[
              item($~Editor           f1 $,hedi),
              item($~Deducción        f10 $,hded),
              item($~Teclas           Alt-S $,htec)]).
    end_menu(menu).
```

Los predicados **begin_menu**, **end_menu**, *nos permiten definir el inicio y el fin de la lista de los comandos y opciones de un menú, el predicado* **item** *se utilizará para indicar cuales serán los componentes de dicho menú.*

El predicado **item** *tiene dos argumentos, el primero es el nombre del comando y el segundo puede ser una lista de* **items** *o un término, si se trata de una lista de* **items** *esto nos indicará que es un submenú asociado al comando correspondiente, en caso de que sea un término, ésto nos indica que se trata de una opción del menú y dicho término fungirá como una clave que posteriormente nos identificará la opción que fué elegida al terminar la ejecución del menú.*

Como hemos visto, la jerarquía de los submenús está dada por la indentación de los **items**. *Los comandos principales son:* **Archivos**, **Axiomas**, **Apoyos** *y* **Ayuda**, *y éstos a su vez tienen sus opciones, por ejemplo la estructura de* **Axiomas** *es la que se muestra a continuación.*

```
item($A~rchivos$,[
        item($~Nueva            Alt-N $,nueva),
        item($~Lee...           Alt-L $,lee),
        item($Ane~xa...         Alt-X $,anexa),
        item($~Graba            Alt-G $,graba),
        item($Graba co~mo...    Alt-M $,g_como),
        break,
        item($~Imprime...       Alt-I $,impri),
        break,
        item($Directo~rio...    Alt-R $,dir),
        item($Cam~bia dir...    Alt-B $,cdir),
        item($D~OS              Alt-O $,dos),
        item($~Terminar         Alt-T $,termina)]).
```

Como podemos observar este comando tiene las opciones de: **Nueva**, **Lee**, **Anexa**, **Graba**, **Graba como**, **Imprime**, **Directorio**, **Cambia**, **DOS** *y* **Terminar** *que se encuentran en una lista a la derecha de* **Archivos** *en el predicado* **item** *y cada una de estas opciones tienen a su vez asociada un término clave. Por ejemplo:*

```
item($~Nueva            Alt-N $,nueva),
```

Nueva *tiene asociado el término* **nueva** *que es la clave que nos indicará que fué elegida esta opción. Por ejemplo, si ejecutamos el predicado* **mns**, *entonces éste a su vez activa el menú con el predicado* **send_menu_msg** *y ya en el menú, seleccionamos la opción de* **Nueva**, *entonces el predicado regresrá la clave* **nueva** *y el predicado se verá así:*

```
send_menu_msg(activate(menu,(0,0)),nueva).
```

Esta clave será tratada en un **case** *y se realizará la acción correspondiente como se muestra a continuación.*

```
case ([
      Ret_val = selecc  -> axiomas(Key),
      Ret_val = termina -> terminar(Key),
      Ret_val = aplica  -> aplicar(principal),
      Ret_val = dos     -> shell,
      Ret_val = graba   -> grabar(Key),
      Ret_val = lee     -> leer(Key),
      Ret_val = anexa   -> anexar(Key),
      Ret_val = nueva   -> nuevo(Key),
      Ret_val = g_como  -> grabar_como(Key),
      Ret_val = impri   -> impresion,
      Ret_val = dir     -> dirs(Key),
      Ret_val = cdir    -> cdirs(Key),
      Ret_val = hedi    -> ay(1),
      Ret_val = justi   -> altjust(Key),
      Ret_val = equiv   -> equivalencias(Key),
      Ret_val = deriva  -> derivacion(Key),
      Ret_val = concluye-> conclusion(Key),
      Ret_val = htec    -> ay(2),
      Ret_val = hded    -> ay(3)
      ]).
```

Aquí nuestra elección fue **Nueva** *y ésta regreso la clave* **nueva**, *entonces se ejecutrá el predicado* **nuevo(Key)**, *el cual nos permitirá iniciar una nueva deducción.*

A continuación describiremos la definición de los predicados que son ejecutados mediante las opciones elegidas en el **Menú**. *Empezaremos con las opciones correspondientes al submenú de* **Archivos**.

ARCHIVOS.

Este submenú, como se habló en el capitulo anterior, tiene las opciones: Nueva, Lee, Anexa, Graba, Graba_como, Imprime, Directorio, Cambia_dir, DOS y Terminar.

- **Nueva**. *Esta opción, borra la deducción actual para empezar una nueva deducción. La definición de este predicado es la siguiente:*

```
nueva(Key):-
        grabas(G,Key),
        nuevo_archivo(Key),
        send_control_msg(ef_set_text(_,$$),2,Key),
        send_control_msg(ef_set_text(_,$$),3,Key),
        send_control_msg(ef_set_text(_,$$),4,Key),
        send_control_msg(ef_set_text(_,$$),5,Key),
        send_control_msg(ef_set_text(_,$$),6,Key),
        send_control_msg(ef_set_text(_,$$),7,Key),
        limpiajus.
```

La ejecución de **nueva** *hace lo siguiente:*

grabas(G,Key): *Pregunta si se graba la deducción actual y la graba en caso afirmativo.*

nuevo_archivo(Key): *Limpia la ventana de deducción y actualiza el nombre archivo-nueva.*

```
send_control_msg(ef_set_text(_,$$),2,Key),
send_control_msg(ef_set_text(_,$$),3,Key),
send_control_msg(ef_set_text(_,$$),4,Key),
send_control_msg(ef_set_text(_,$$),5,Key),
send_control_msg(ef_set_text(_,$$),6,Key),
send_control_msg(ef_set_text(_,$$),7,Key),
```

Con estas instrucciones limpia las ventanas de **edición, Metavar "X", Metavar "B", Metavar "C", Término y Var.**

limpiajus: *Limpia la justificación de la deducción anterior.*

- **Lee.** *Esta opción, nos permite leer una nueva deducción del disco. La definición de este predicado es la siguiente:*

```
leer(Key):-
     directorio(NewCurr),
     ifthen(NewCurr \= cancela,
          (grabas(G,Key),
          limpiajus,
          actualiza_file(Key,NewCurr),
          lee_jus(Key,NewCurr,0,0),
          )).
```

Es decir, **leer** *realiza lo siguiente:*

directorio(NewCurr): *Muestra el directorio.*

Si el nombre del archivo no es cancelar entonces ejecuta los cuatro siguientes predicados:

grabas(G,Key): *Preguntas si se graba la deducción actual y la graba en caso que se desee.*

limpiajus: *Limpia la justificación de la deducción anterior.*

actualiza_file(Key,NewCurr): *Agrega la deducción nueva al editor y actualiza el nombre del archivo que se leyó.*

lee_jus(Key,NewCurr,0,0): *Agrega la justificación a la base de datos.*

- **Anexa**. *Esta opción, nos permite hacer un "merge" de dos deducciones. La definición de este predicado es la siguiente:*

```
anexar(Key):-
      directorio(NewCurr),
      ifthen(NewCurr \= cancela,
           anexa_file(Key,NewCurr,R1)).
```

Es decir, **anexar** *realiza lo siguiente:*

directorio(NewCurr): *Muestra el directorio.*

Si el nuevo archivo no es cancela entonces ejecuta:

anexa_file(Key,NewCurr,R1) : *Realiza el "merge" antes indicado.*

- **Graba**. *Graba la deducción que se encuentra en pantalla al disco, con el nombre vigente. La definición de este predicado es la siguiente:*

```
grabar(Key) :-
        curr_file(File),
        gba(Key,File),
        gba_jus(FF).
```

grabar *realiza lo siguiente:*

curr_file(File) : *Toma en* **File** *el nombre del archivo actual.*

gba(Key,File): *Graba la deducción al disco.*

gba_jus(FF): *Graba la justificación de la deducción.*

- **Graba_como**. *Almacena la deducción actual con algún nombre en el disco. La definición de este predicado es la siguiente:*

```
grabar_como(Key):-
        obten_nombre(NewFile,'Nvo. Nombre:'),
        ifthen(NewFile \= cancela,
                gba(Key,NewFile),
                gba_jus(NewFile)).
```

Es decir, al ejecutarse **grabar_como** *se realiza lo siguiente:*

obten_nombre(NewFile,'Nvo. Nombre:'): *Pide el nuevo nombre para grabar la deducción que se encuentra en pantalla.*

Si el nuevo nombre no es cancela entonces ejecuta los siguientes dos predicados

gba(Key,NewFile): *Graba la deducción al disco.*

gba_jus(NewFile): *Graba la justificación de la deducción.*

- **Imprime**. *Manda a la impresora una deducción. La definición de este predicado es la siguiente:*

```
impresion:-
     directorio(FileNewCurr),
     ifthen(NewCurr \= cancela,
          imprimir(NewCurr)).
```

impresion *realiza lo siguiente:*

directorio(FileNewCurr): *Muestra el directorio y permite seleccionar un archivo de allí.*

Si el archivo no es cancelar entonces ejecuta:

imprimir(NewCurr)): *Imprime la deducción seleccionada.*

- **Directorio**. *Nos presenta los archivos del disco con un formato especifico. La definición de este predicado es la siguiente:*

```
dirs(Key) :-
     obten_nombre(Forma,'Formato : '),
     ifthen(Forma \= cancela,
          (cambia_forma(Forma),
          directorio(_))).
```

Es decir, al ejecutar **dirs** *se realiza lo siguiente:*

obten_nombre(Forma,'Formato : '): *Pide el formato de los archivos que se desean ver.*

Si el formato no fué cancela entonces ejecuta los siguientes dos predicados:

(cambia_forma(Forma): *Actualiza en nuevo formato.*

directorio(_))): *Muestra el directorio.*

- **Cambia_dir**. *Nos permite cambiar el path del direcorio. La definición de este predicado es la siguiente:*

```
:

cdirs(Key):-
     obten_nombre(NewDir,'Nuevo dir:'),
     ifthen(NewDir \= cancela,
          cambia_dir(NewDir)),
```

Al ejecutar **cdir** *hace lo siguiente:*

obten_nombre(NewDir,'Nuevo dir:'): *Obtiene el formato del nuevo path.*

Si no se canceló la opción, entonces ejecuta:

cambia_dir(NewDir)): *Cambia el path para el directorio.*

- **Dos**. *Simplemente ejecuta un predicado intrinseco de* **Arity Prolog** *llamado* **shell**.

  Su definición es la siguiente:

  ```
  dos :- shell.
  ```

- **Terminar**. *termina la ejecución del programa. Su definición es la siguiente:*

  ```
  terminar(Key) :-
       ifthen(0 == 1,
            grabas(1,Key),
       limpia_ventanas,
       exit_dbox(Key)).
  ```

  Al ejectar **terminar**, *realiza lo siguiente:*

  Si hubo modificaciones, pregunta si se graban y lo hace si se responde afirmativamente.

  limpia_ventanas: *Limpia todas las ventanas.*

  exit_dbox(Key): *Termina la ejecución de* **dialog_run**.

AXIOMAS.

Este submenú, tiene las opciones de **Seleccionar** *y* **Aplicar**. *A continuación describiremos cada una de ellas.*

- **Seleccionar**. *Nos presenta los axiomas disponibles para una posible aplicación. Su definición es la siguiente.*

```
axiomas(Key) :-
        ax,
        brinca(Key,Cua,Ida).
```

axiomas *realiza lo siguiente:*

ax: *Muestra los axiomas y permite seleccionar alguno.*

brinca(Key,Cua,Ida): *Brinca a la primera ventana de sustituciones.*

- **Aplicar**. *Lleva a cabo la aplicación del axioma seleccionado con las correspondientes sustituciones. Su definición es la siguiente.*

```
aplicar(Key) :-
        acax(Key),
        agregas.
```

Al ejecutar **aplicar** *realiza lo siguiente:*

acax(Key): *Aplica el axioma o regla elegida.*

agregas: *Pregunta si se agrega el resultado de la aplicación a la lista de la deducción.*

APOYOS.

Este submenú nos brinda cuatro opciones: **Concluye, Deriva, Justifica, y Equivalencias**, *a continuación describiremos cada uno de ellos.*

- **Concluye**. *Le sugiere al usuario los pasos para concluir una fórmula. Su definición es la siguiente.*

```
conclusion(Key):-
        getfor(Key,Ctrl,F),
        correcta(F,_,$Fórmula$),
        concluc(F).
```

Es decir, **conclusion** *realiza lo siguiente.*

getfor(Key,Ctrl,F): *Obtiene la fórmula a donde se encuentra apuntando el cursor.*

correcta(F,_,$Fórmula$): *Checa si la fórmula obtenida es correcta.*

concluc(F): *Despliega la conclusión deseada.*

- **Deriva.** *Sugiere al usuario lo que puede deducir a partir de cierta fórmula. Su definición es la siguiente.*

```
derivacion(Key):-
        getfor(Key,Ctrl,F),
        correcta(F,_,$Fórmula$),
        deri(F).
```

Es decir, **derivacion** *realiza lo siguiente.*

getfor(Key,Ctrl,F): *Obtiene la fórmula a donde se encuentra apuntando el cursor.*

correcta(F,_,$Fórmula$): *Checa si la fórmula obtenida es correcta.*

deri(F): *Despliega la sugerencia acerca de la derivación.*

- **Equivalencias.** *Muesta algunas fórmulas lógicamente equivalentes a una fórmula dada. Su definición es la siguiente:*

```
equivalencias(Key):-
        getfor(Key,Ctrl,F),
        correcta(F,For,$Fórmula$),
        equival(For,E1,E2),
        putfor(Key,Ctrl1,E1).
```

Es decir, al ejecutarse **equivalencias**, *se realiza lo siguiente:*

getfor(Key,Ctrl,F): *Obtiene la fórmula donde se encuentra apuntando el cursor.*

correcta(F,For,$Fórmula$): *Checa que la fórmula esté correctamente escrita.*

equival(For,E1): *Encuentra una fórmula lógicamente equivalente.*

putfor(Key,Ctrl1,E1): *Coloca el resultado en la ventana de* **edición.**

- **Justificación.** *Muesta la justificación de alguna fórmula que pertenezca a la lista de la deducción. Su definición es la siguiente:*

```
altjust(Key):-
        send_control_msg(eb_get_pos((Rr,_),(Rl,_),Key),
        despliega_jus(Rl).
```

Cuando se ejecuta el predicado altjust, se realiza lo siguiente:

send_control_msg(eb_get_pos((Rr,_),(Rl,_),Key): *Obtiene la posición del cursor en la ventana de deducción.*

despliega_jus(Rl): *Obtiene la justificación de* **Rl** *y la despliega.*

Como dijimos anteriormente. Estos dos predicados, **Dilalog** *y* **Menú** *son la base de todo el ambiente. Falta conocer ahora la parte que realiza el trabajo de la lógica, ésto se mostrará en el siguiente inciso.*

3.4 LOS PREDICADOS DE APOYO A LA GENERACION DE UNA DEDUCCION.

Este módulo esta constituido por un conjunto de predicados que realizan el trabajo relacionado con la deducción. Por ejemplo, el chequeo de las fórmulas (que estén bien formadas), la aplicación de los axiomas, el renombramiento de variables, etc.

Estos predicados no están agrupados bajo un predicado general, ya que cada uno de ellos fué diseñado en forma independiente. Sin embargo, existen predicados auxiliares que se utilizan en la definición de otros.

Algunos de estos predicados corresponden directamente a acciones solicitadas por el usuario y otras son auxiliares a estas acciones.

A continuación describiremos a grandes razgos algunos de los predicados mas relevantes y su definición puede verse el el anexo.

- **acot(X,P).** *Checa si la variable* **X** *está acotada en la fórmula* **P**.

  :

- **alc(X,Y,F).** *Checa si la variable* **X** *está bajo el alcance de algún cuantificador de la variable* **Y** *dentro de la fórmula* **F**.

- **fbf(F).** *Checa si la fórmula* **F**, *es una fórmula bien formada.*

- **libre_para(X,T,F).** *Este predicado checa si* **X** *es libre para el término* **T** *en la fórmula* **F**.

- **reemplaza(X,T,F,P).** **P** *es el resultado que se obtiene al reemplazar el término* **T** *por la variable* **X** *en la fórmula* **F**, *checando que se cumplan las condiciones requeridas para realizar este reemplazo.*

A continuación describiremos a grandes razgos el predicado que se encarga de la aplicación de los axiomas o reglas de inferencia. Dicho predicado, tiene los siguientes argumentos:

apl_ax(Ax,S1,S2,S3,Ter,Var,Res)).

Ax: *Nos indica el número de axioma, que se pretende instanciar.*

S1, S2, S3: *Contienen la fórmulas que se sustituiran a las metavariables* **A, B,** *y* **C** *del axioma en cuestión.*

Ter: *Contiene el término a checar si es libre para la variable* **Var.**

Var: *Contiene la variable en la cual* **Ter** *debe ser libre.*

Res: *Es el argumento que se utiliza para regresar la fórmula resultante de la aplicación del axioma ó regla de inferencia en cuestión.*

El predicado **apl_ax**, *se define de manera distinta para cada uno de los axiomas del sistema. Para ejemplificar presentaremos la definición de algunos de ellos.*

Para los axiomas

ax_0: A ⇒ (B⇒A)

ax_1: (A⇒B) ⇒ ((A ⇒ (B⇒C)) ⇒ (A⇒C))

R_2: A, A⇒B ⊢ B

Ax_12: ∀xA(x) ⇒ A(t)

se utilizan las siguientes definiciones:

```
apl_ax(0,A,B,_,_,_,A => (B => A) ):- !.

apl_ax(1,A,B,C,_,_,(A=>B) => ( (A => (B=>C)) => (A=>C))) :- !.

apl_ax(2,A,A => B,_,_,_,B):- !.
apl_ax(2,_,_,_,_,_,_,_):-
      !,
      create_popup('',(10,15),(13,60),(63,63)),
      write('No se puede aplicar modus ponens'),nl,
      write('oprime cualquier tecla para continuar ...'),
      keyb(_,_),exit_popup,ctr_set(20,1),ctr_dec(10,R),
      ifthen(R == 1, ctr_set(10,1)).


apl_ax(12,p_t(X,Ax),_,_,X,T,p_t(X,Ax) => At):- !,
      aplica_ax(12,p_t(X,Ax),_,_,X,T,p_t(X,Ax) => At).
apl_ax(12,_,_,_,_,_,_,_):-
      mensaje($La fórmula no tiene la forma ∀xA(x)  o $),
      mensaje($Var no coincide con la de la fórmula.$),
      ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.
```

```
aplica_ax(12,p_t(X,Ax),_,_,X,T,p_t(X,Ax) => At):-
     [ !libre_para(T,X,Ax) !],reemplaza(X,T,Ax,At), !.
aplica_ax(12,_,_,_,_,_,_):-
     !,mensaje($El término no es libre para la variable$),
     ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.
```

Ahora veremos una instanciación de cada uno de los axiomas antes mencionados. Si tenemos que la metavariable ∅, será instanciada por **p(x)**, *y la metavariable ℬ por* **m(x,y)**, *y se desea aplicar el axioma* **ax_0**, *entonces el predicado* **apl_ax** *quedará instanciado como sigue:*

```
apl_ax(0,p(x),m(x,y),_,_,_,p(x) => (m(x,y)=>p(x))).
```

La instanciación del axioma **ax_1**, *con las mismas sustituciones del ejemplo anterior y donde la metavariable ℬ será sustituida por* **r(y)**, *quedaría de la siguiente forma:*

```
apl_ax(1,p(x),m(x,y),r(y),_,_
(p(x)=>m(x,y)) => ( (p(x)=>(m(x,y)=>r(x))) => (p(x) => r(x))) )
```

Nótese que en el caso de los axiomas **ax_0** *y* **ax_1**, *los argumentos* **A**, **B**, *y* **C** *no tienen ninguna restricción, mientras que en el caso de* **modus ponens**, *el tercer argumento no puede ser cualquier fórmula, sino una del tipo ∅ ⇒ ℬ, en donde ∅ es la misma ∅ del segundo argumento.*

La aplicación de la regla R_2 puede verse con el siguiente ejemplo, si ∅ es sustituida por **m(x)** *y ∅⇒ℬ por* **m(x)⇒p(x)**, *entonces el predicado* **apl_ax**, *se verá así:*

```
apl_ax(2,m(x),m(x)=>p(x),_,_,_,p(x)).
```

y la fórmula que regresará como resultado de la aplicación, será la siguiente:

```
p(x)
```

Por último veremos que pasa al tratar de aplicar la regla R_2 con las siguientes sustituciones $\mathcal{A}$ por **p(x)**, *y $\mathcal{A}\Rightarrow\mathcal{B}$ por* **m(x,y)**. *Como la primera definición de la regla R_2 es de la siguiente forma para el axioma 2 es de la siguiente forma:*

```
apl_ax(2,A,A => B,_,_,_,B):- !.
```

al instanciar **p(x)** *con el segundo argumento* **A** *del predicado, no habrá ningún problema, pero al tratar de instanciar el tercer argumento que tiene la forma* **A=>B** *con* **m(x,y)**, *fallará y por lo tanto pasará a al siguiente definición de la regla R_2:*

```
apl_ax(2,_,_,_,_,_,$$):-
      !,
      create_popup('',(10,15),(13,60),(63,63)),
      write('No se puede aplicar modus ponens'),nl,
      write('oprime cualquier tecla para continuar ...'),
      keyb(_,_),exit_popup,ctr_set(20,1),ctr_dec(10,R),
      ifthen(R == 1, ctr_set(10,1)).
```

la cual le indicará al usuario, que no es posible aplicar éste axioma.

Ahora veremos un ejemplo de instaciación del axioma Ax_12, para ejemplificarlo utilizaremos la fórmula $\forall x\exists y p(x,y)$, el término será **q(x)** *y la variable será* **x**. *La fórmula antes mencionada está escrita en notación clásica o externa, sin embargo dicha fórmula debe estar escrita en forma interna, es decir,*

p_t(x,e_x(y,p(x,y))), *al instanciar* el predicado quedará de la siguiente forma:

apl_ax(12,p_t(x,e_x(y,p(x,y))),_,_,x,q(x),p_t(x,e_x(y,p(x,y)))=>At)

y a su vez instanciará el predicado

aplica_ax(12,p_t(X,Ax),_,_,X,T,p_t(X,Ax) => At)

el cual checará que el término **q(x)** *sea libre para la variable* **x** *en la fórmula* **e_x(y,p(x,y))**, *para lo cual utilizará el siguiente predicado:*

libre_para(q(x),x,e_x(y,p(x,y)))

en caso de dicho término sea libre para la fórmula, el siguiente predicado realizará el reemplazo y la instancia se verá así:

reemplaza(x,q(x),e_x(y,p(x,y)),At)

el resultado del reemplazo lo regresará en la variable **At**, *para verse así:*

reemplaza(x,q(x),e_x(y,p(x,y)),e_x(y,p(q(x),y))).

Si la fórmula no tiene la forma requerida, es decir **∀xA(x)**, *o si la variable no concide con la variable del cuantificador, entonces fallará la primer definición del predicado y se ejecutará la segunda definición:*

```
apl_ax(12,_,_,_,_,_):-
        mensaje($La fórmula no tiene la forma ∀xA(x)  o $),
        mensaje($Var no coincide con la de la fórmula.$).
```

el cual indicará al usuario la razón del fallo. Ahora que si la fórmula tiene el formato requerido y la variable coincide con la

variable del cuantificador, pero el término no es libre para la variable en la fórmula, entonces ejecutará el siguiente predicado:

```
aplica_ax(12,_,_,_,_,_,_):-
        mensaje($El término no es libre para la variable$).
```

que indicará al usuario que el término no es libre para la variable en la fórmula.

No nos extenderemos más sobre las definiciones de los otros predicados que definen los comandos de apoyo a la deducción, pero en el anexo pueden verse una lista de ellos y de algunas de sus utilerías.

CONCLUSIONES

Programar un sistema como **Edilog** *significó entre otras cosas: la manipulación de fórmulas y la revisión de la correctez de las mismas; la aplicación de axiomas y reglas de inferencia; el chequeo de condiciones sobre variables y términos para la aplicación correcta de los axiomas y reglas de inferencia. Todo ello se hizo de manera elegante utilizando la recursión y el poder de manipulación simbólica de Prolog.*

Las facilidades que ofrece **Edilog** *en cuanto a: la interacción con el sistema operativo para actualizar y grabar archivos, imprimir, ver directorios, etcétera; la creación de un ambiente amigable con ventanas, ayudas y menús; las facilidades propias de edición que implican insertar, copiar y borrar, entre otras, fueron tareas complejas que requirieron esfuerzo, dedicación y mucho tiempo y que se facilitaron en cierta medida gracias al uso de Arity Prolog.*

Lo que hemos presentado en este trabajo es una primera versión de un editor para la ayuda de deducciones formales en la lógica de primer orden que tiene seguramente todavía varias limitaciones.

Se ha previsto ya una segunda versión extendida de **Edilog** *que contempla nuevos predicados que incluyan la aplicación de teoremas como el de la deducción y otros mencionados en las secciones 1.4 y 1.5 y que, por falta de tiempo, ya no se ioncluyeron en esta versión.*

Sería conveniente también que el conjunto de axiomas definidos pudiese cambiarse a gusto del usuario. Ya sea que **Edilog** *cuente con distintas axiomáticas y sea posible escoger alguna de ellas, o bien, que pueda darse de alta alguna axiomática particular.*

Otra posible extensión, más ambiciosa, tiene que ver con la posibilidad de que **Edilog** *pueda ayudar efectivamente al usuario a pensar en como desarrollar una deducción. En este sentido se incluyeron en esta primera versión dos comandos que permiten visualizar: 1) tipos de fórmulas derivables a partir de una hipótesis (DERIVA) y 2) axiomas que permiten concluir una fórmula dada (CONCLUYE). Sin embargo, ninguna de estas dos opciones puede considerarse como una ayuda "inteligente" para el usuario. Una vez que se cuente con una versión más depurada del editor, se pretende explorar la posibilidad de integrar un módulo "inteligente" que pueda ayudar al usuario a guiar una deducción.*

Los ejemplos antes mencionados son sólo unas cuantas posibles extensiones y agreagados que podemos hacerle a **Edilog**, *pero estamos seguros de que hay muchas más ideas que irán surgiendo con la práctica del uso y que podrán ir sofisticando cada vez más el editor.*

Finalmente, queremos mencionar que **Edilog** *forma parte de un proyecto de apoyo a la lógica, AMBILOG, más extenso y que pretende crear todo un ambiente de trabajo para la lógica de primer orden. Dentro de este proyecto se incluye también, actualmente, un traductor de fórmulas de la lógica a formas clausulares y programas Prolog, que se reporta en [6], [7], [8], [11], y cuyas implantaciones ayudaron también al desarrollo de este programa.*

Es nuestro deseo que este trabajo contribuya en algo al desarrollo de los cursos de lógica matemática con un enfoque más moderno y que sea en general de utilidad para los que trabajan en esta área de las matemáticas.

BIBLIOGRAFIA

[1] ARITY PROLOG 5.5. Manual de Referencia.

[2] BUILDING ARITY/PROLOG APPLICATIONS, 1986, Arity Corporation.

[3] ERIKSSON A y JOHANSSON A., NATEDED, A Derivation Editor. UPMail Techinical Report No. 3, 1981-10-01, Uppsala, Suecia.

[4] ERIKSSON A y JOHANSSON A., Towards a Derivation Editor. UPMail Techinical Report No. 11, 1982-08-02, Uppsala, Suecia.

[5] KLEENE, Stephen Cole, INTRODUCTION TO METAMATHEMATICS. Toronto-London, Van Nostrand Co.,1967.

[6] LOYO, Cristina y SABRE, Sonia, TRANSFORMACION AUTOMATICA A FORMA CLAUSULAR. Memorias del congreso PASADO PRESENTE Y FUTURO DE LA COMPUTACION EN MEXICO. Facultad de Ciencias, UNAM. Junio de 1988.

[7] LOYO, Cristina y SABRE, Sonia, "Mejoras al algoritmo clásico de Skolemización". Aportaciones matemáticas, comunicaciones No. 6. Artículo de investigación, páginas 13-34, 1988.

[8] LOYO, Cristina y Rios Figueroa Homero, "Hacia la implantación de un algoritmo óptimo de Skolemización". Por aparecer.

[9] MANNA Zohar y WALDINGER Richard, THE LOGICAL BASIC FOR COMPUTER PROGRAMMING. ADDISON-WESLEY PUBLISHING COMPANY, 1985.

[10] MENDELSON, Elliott, INTRODUCTION TO MATHEMATICAL LOGIC. Toronto-London, Van Nostrand Co.,1967.

[11] Sabre Sonia, "Traducción Automática a Forma Clausular". Tesis de Licenciatura en Matemáticas, Facultad de Ciencias, UNAM. Septiembre de 1988.

[12] STERLING, L. y SHAPIRO, E., THE ART OF PROLOG. Boston Mass., MIT Press, 1986.

[13] TURBO PROLOG 2. Manual de Referencia.

ANEXO

Definición de algunos predicados de apoyo a una deducción:

```
libre(X, P) :- ocurrencias_libres(X, P).


libre_para(T, X, P <=> Q) :-
     !,
     (libre_para(T, X, P) ; libre_para(T, X, Q)).
     libre_para(T, X, P => Q) :-
     !,
     (libre_para(T, X, P) ; libre_para(T, X, Q)).
     libre_para(T, X, P ∧ Q) :-
     !,
     (libre_para(T, X, P) ; libre_para(T, X, Q)).
libre_para(T, X, P∨ Q) :-
     !,
     (libre_para(T, X, P) ; libre_para(T, X, Q)).
libre_para(T, X, # P) :-
     !,
     libre_para(T, X, P).
libre_para(T, X, P) :-
     libre(X,P),
     lista(T,Variables_en_T),
     separa_var(P,En_cuant,_),
     ajenas(En_cuant,Variables_en_T).


acotada(X,F):-not(ocurrencias_libres(X,F)).



termino_libre_para(T,X,F):-
     ocurrencia(X,F),!,termino_libre_(T,X,F).
termino_libre_para(_,_,_):-
```

```
termino_libre_para(T,X,F):-
     ocurrencia(X,F),!,termino_libre_(T,X,F).
termino_libre_para(_,_,_):-
     mensaje($La variable no ocurre en la fórmula$).
termino_libre_(X,X,_):-
     ),mensaje($El término es libre para la variable$).
termino_libre_(T,X,F):-
     libre_para(T,X,F),!,
     mensaje($El término es libre para la variable$).
termino_libre_(_,_,_):-
     mensaje($El término NO es libre para la variable$).
```

***** aplicación de los axiomas *****

```
apl_ax(0,A,B,_,_,_, A => (B => A)) :-!.
apl_ax(1,A,B,C,_,_, (A=>B) => ( (A => (B=>C)) => (A=>C)) ) :-).
apl_ax(2,A,A=>B,_,_,_, B) :-!.
apl_ax(2,_,_,_,_,_,_) :-
     !,mensaje($No se puede aplicar Modus Ponens$),
     ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.

apl_ax(3,A,B,_,_,_, A => B => (A ∧ B)) :-!.
apl_ax(4,A,B,_,_,_,(A ∧ B) => A) :-!.
apl_ax(5,A,B,_,_,_,(A ∧ B) => B) :-!.
apl_ax(6,A,B,_,_,_, A => A∨B) :-!.
apl_ax(7,A,B,_,_,_, B => A∨B) :-!.
apl_ax(8,A,B,C,_,_, (A=>C) => (B=>C) => A∨B => C) :-!.
apl_ax(9,A,B,_,_,_, (A=>B) => (A=> # B) => # A) :-!.
apl_ax(10,# # A,_,_,_,_,_, A) :- !.
apl_ax(10,_,_,_,_,_,_) :-
     mensaje($La fórmula no tiene la forma ¬(¬A)$),
     ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.
```

```
aplica_ax(11,C => Ax,_,_,X,_, C => p_t(X,Ax) ):-
     acotada(X,C),!.
aplica_ax(11,C => Ax,_,_,X,_, C => p_t(X,Ax) ):-
     mensaje($La variable no est! acotada en la fórmula$),
     ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.


apl_ax(12,p_t(X,Ax),_,_,X,T,p_t(X,Ax) => At):- !,
     aplica_ax(12,p_t(X,Ax),_,_,X,T,p_t(X,Ax) => At).
apl_ax(12,_,_,_,_,_,_):-
     mensaje($La fórmula no tiene la forma Ax A(x)  ^ $),
      mensaje($Var no coincide con la de la fórmula.$),
     ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.

aplica_ax(12,p_t(X,Ax),_,_,X,T,p_t(X,Ax) => At):-
     [)libre_para(T,X,Ax))],reemplaza(X,T,Ax,At),!.
aplica_ax(12,_,_,_,_,_,_):-
     ),mensaje($El t rmino no es libre para la variable$),
     ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.

apl_ax(13,At,_,_,X,T, At => e_x(X,Ax)):-
     ocurre_term(T,At),!,
     aplica_ax(13,At,_,_,X,T, At => e_x(X,Ax)).

apl_ax(13,_,_,_,_,_,_):-
     !,mensaje($El término no ocurre en la fórmula.$),
     ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.

aplica_ax(13,At,_,_,X,T, At => e_x(X,Ax)):-
     [!reemplaza(T,X,At,Ax)!],libre_para(T,X,Ax),!.

aplica_ax(13,_,_,_,_,_,_):-
     mensaje($El término no es libre para la variable.$),
     ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.
```

```
aplica_ax(13,At,_,_,X,T, At => e_x(X,Ax)):-
     [!reemplaza(T,X,At,Ax)!],libre_para(T,X,Ax),!.

aplica_ax(13,_,_,_,_,_,_):-
     mensaje($El término no es libre para la variable.$),
     ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.

apl_ax(14,Ax => C,_,_,X,_,e_x(X,Ax) => C ):-!,
     aplica_ax(14,Ax => C,_,_,X,_,e_x(X,Ax) => C ).

apl_ax(14,_,_,_,_,_,_):-
     mensaje($La fórmula no tiene la forma A(x) => C.$),
     ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.

aplica_ax(14,Ax => C,_,_,X,_,e_x(X,Ax) => C ):-
     acotada(X,C),!.
aplica_ax(14,_,_,_,_,_,_):-
     mensaje($La variable no está acotada en la fórmula$),
     ctr_set(20,1),ifthen(R == 1,ctr_set(10,1)),fail.


***** la definición de fórmulas bien formadas*****

fbf(p_t(X,F)):-!.
fbf(e_x(X,F)):-!.
fbf(A <=> B):-!,fbf(A),fbf(B).
fbf(A => B):-!,fbf(A),fbf(B).
fbf(A ∨ B):-!,fbf(A),fbf(B).
fbf(A ∧ B):-!,fbf(A),fbf(B).
fbf(# A):-!,fbf(A).
fbf(A):-A =..[C,R],!,C  ','.
fbf(A):-A =..[C|R],!,C  ','.
fbf(A).
```

```
ocurrencias_libres(X, P <=> Q) :-!,
     (ocurrencias_libres(X, P) ; ocurrencias_libres(X, Q)).
ocurrencias_libres(X, P => Q) :-!,
     (ocurrencias_libres(X, P) ; ocurrencias_libres(X, Q)).
ocurrencias_libres(X, P ∧ Q) :-!,
     (ocurrencias_libres(X, P) ; ocurrencias_libres(X, Q)).
ocurrencias_libres(X, P ∨ Q) :-!,
     (ocurrencias_libres(X, P) ; ocurrencias_libres(X, Q)).
ocurrencias_libres(X, ~ P) :-!,
     ocurrencias_libres(X, P).
ocurrencias_libres(X,P):-en(X,P).


reemplaza(X,T,P,Q):-t_del_reemplazo(X,T,P,[],Q).

t_del_reemplazo(X,T,p_t(X,P),_,p_t(X,P)):-!.
t_del_reemplazo(X,T,e_x(X,P),_,e_x(X,P)):-!.
t_del_reemplazo(X,T,p_t(Y,P),L,p_t(Y,Q)):-!,
     concatena([Y],L,L1),
     t_del_reemplazo(X,T,P,L1,Q).
t_del_reemplazo(X,T,e_x(Y,P),L,e_x(Y,Q)):-!,
     concatena([Y],L,L1),
     t_del_reemplazo(X,T,P,L1,Q).


t_del_reemplazo(X,T,P <=> Q,L,P1 <=> Q1 ):-!,
     t_del_reemplazo(X,T,P,L,P1),
     t_del_reemplazo(X,T,Q,L,Q1).
t_del_reemplazo(X,T,P  => Q,L,P1  => Q1 ):-!,
     t_del_reemplazo(X,T,P,L,P1),
     t_del_reemplazo(X,T,Q,L,Q1).
t_del_reemplazo(X,T,P  => Q,L,P1  => Q1 ):-!,
     t_del_reemplazo(X,T,P,L,P1),
     t_del_reemplazo(X,T,Q,L,Q1).
t_del_reemplazo(X,T,P  ∧ Q,L,P1  ∧ Q1 ):-!,
     t_del_reemplazo(X,T,P,L,P1),
```

```
t_del_reemplazo(X,T,P <=> Q,L,P1 <=> Q1 ):-!,
      t_del_reemplazo(X,T,P,L,P1),
      t_del_reemplazo(X,T,Q,L,Q1).
t_del_reemplazo(X,T,P  => Q,L,P1  => Q1 ):-!,
      t_del_reemplazo(X,T,P,L,P1),
      t_del_reemplazo(X,T,Q,L,Q1).
t_del_reemplazo(X,T,P  => Q,L,P1  => Q1 ):-!,
      t_del_reemplazo(X,T,P,L,P1),
      t_del_reemplazo(X,T,Q,L,Q1).
t_del_reemplazo(X,T,P  ∧ Q,L,P1  ∧ Q1 ):-!,
      t_del_reemplazo(X,T,P,L,P1),
      t_del_reemplazo(X,T,Q,L,Q1).
t_del_reemplazo(X,T,P∨ Q,L,P1 ∨ Q1 ):-!,
      t_del_reemplazo(X,T,P,L,P1),
      t_del_reemplazo(X,T,Q,L,Q1).
t_del_reemplazo(X,T, # P ,L, #Q):-!,
      t_del_reemplazo(X,T,P,L,Q).
t_del_reemplazo(X,T, P ,L, Q):-
      P=..[Cab|Cuerpo],
      lista(T,Lista),
      ajenas(L,Lista),!,
      reemplaza_en_lista(X,T,Cuerpo,Reemplazo),
      Q=..[Cab|Reemplazo].
t_del_reemplazo(_,_, P ,_, P).
      pareja_gen([X|R],[[X,Y]|L]):-!,
      gensim(y,Y),
      pareja_gen(R,L).
      pareja_gen([],[]).
```