UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

UN SISTEMA DE APOYO AUTOMATIZADO PARA LOS CURSOS DE LOGICA MATEMATICA I Y II

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 :

ATOCHA ALISEDA LLERA

MEXICO, D. F. 1989

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

I N D I C E

INTRODUCCION

Como resultado de mi experiencia en varios cursos de Lógica I con el profesor Carlos Torres, cursos en los que se utilizó el lenguaje de programación PROLOG para introducir al alumno al lenguaje de la lógica, surgió la idea de usar una implementación de Prolog para diseñar e instrumentar un sistema de apoyo para la enseñanza de la lógica. La realización de esta idea culminó en la elaboración de esta tesis.

Aprender lógica no sólo involucra aprender un lenguaje, conocer un alfabeto y las reglas para construir las cadenas en ese lenguaje. Además, a través de este lenguaje elaboramos una teoría de la deducción lógica, aprendemos a deducir consecuencias de premisas, estudiamos la verdad o falsedad de enunciados a partir de la verdad o falsedad de otros enunciados, establecemos la consistencia de premisas y verificamos la validez de los argumentos. Dados estos conceptos, el buen manejo de este lenguaje se logra a través de su uso.

Así que, como gran parte del aprendizaje de la lógica se basa en la práctica, la importancia de hacer un sistema automatizado para la enseñanza de la lógica radica en que proporciona al alumno una herramienta para practicar los conceptos aprendidos. Por ejemplo, en clase aprende la definición de fórmula y posteriormente en el sistema, utiliza el verificador de fórmulas para probar si las expresiones que construye lo son.

Un sistema automatizado ayuda al alumno a detectar y corregir sus errores sin ayuda directa del maestro, además de que lo motiva a trabajar fuera de clase.

El objetivo principal de esta tesis es el de proporcionar un sistema automatizado que sirva de apoyo a los cursos de lógica I y II. Un sistema donde el alumno pueda definir diferentes alfabetos, construir términos y fórmulas, ver como se comportan las variables y cuantificadores en las fórmulas, verificar si dos expresiones son unificables y, por último, verificar si una conclusión se deduce de un conjunto de premisas o si una fórmula es o no una tautología.

Basándonos en las notas del curso antes mencionado, se escogieron para su instrumentación, por un lado, los conceptos básicos del curso de lógica I que se refieren a la sintaxis de los lenguajes de primer orden. Por otro lado, se escogieron los conceptos que se basan en métodos algorítmicos, pues son los mas naturales de pasar a una computadora, como lo son el algoritmo de unificación y el método de árboles de verdad.

Esto hace que el material presentado en esta tesis, tanto en su parte teórica como en su instrumentación, no sea una copia fiel de lo que se da usualmente en los cursos de lógica I y II. Sin embargo, puede complementarse con otro material.

Se escogio Prolog para la instrumertación, ya que es un lenguaje para la **programación lógica** (el intento del uso de la logica como lenguaje de programación) y las similitudes entre el lenguaje de la lógica y el de Prolog, permitieron traducir los algoritmos de manera casi directa.

Dado que la programacion logica tiene sus fundamentos en la lógica, el material de esta tesis puede ser visto tambien, como una presentacion de la fundamentación matematica que ha dado origen a los lenguajes lógicos. Aunque este aspecto no se toca en ningun momento de la tesis, es importante señalarlo.

Esta tesis está dividida en tres partes : LENGUAJES DE PRIMER ORDEN (TEORIA), FUNCIONAMIENTO DE LOS SISTEMAS AUTOMATIZADOS e INSTRUMENTACION DE LOS SISTEMAS.
La primera de ellas, es el soporte teórico a los sistemas automatizados. En ella se da una exposición de los conceptos de la logica de 1er. orden necesarios para entender como funcionan los sistemas. Se compone de los capítulos 1 y 2. En el capitulo 1, se da una introducción a los lenguajes formales y se presenta su sintaxis, haciendo una exposicion del alfabeto, los terminos y las fórmulas. En el capitulo 2, se expone la semántica de los lenguajes de 1er. orden, presentando los conceptos de: asignación de verdad a las fórmulas, variables y cuantificadores en las fórmulas, sustituciones y unificacion a través del algoritmo de unificación y, por último, el de inferencias y tautologias a través del metodo de arboles de verdad.

La segunda parte, es el manual de utilización de los sistemas **Lógica I** y **Lógica II**. Se compone de los capítulos 3 y 4. En Cada uno de ellos, se presenta la operación del sistema correspondiente y se expone cada módulo, guiando al usuario a través de ejemplos, tomados en la mayoría de los casos, de la parte de teoria.

La tercera parte, presentada en el capitulo 5, es la exposición del diseño de los sistemas y de los algoritmos principales que se utilizaron en cada modulo.

Por último, en el apendice se da una guia al usuario de los errores más comunes con los que se puede enfrentar en los sistemas, así como recomendaciones de que hacer en caso de daños ocasionados por cuestiones externas a los sistemas.

PARTE I

LENGUAJES DE PRIMER ORDEN (TEORIA)

CAPITULO 1: SINTAXIS

1.1 El LENGUAJE

1.1.1 Lenguaje Natural

Los seres humanos nos comunicamos de muy distintas formas: hablando, escribiendo, haciendo gestos o señales; se llama a un amigo por teléfono para comunicarle que lo vamos a visitar; se escribe una carta a un hermano para comunicarle que le hemos enviado un libro; se indica con luz roja en un semáforo que debemos parar.
Al realizar todas estas acciones con el fin de comunicarnos, estamos utilizando el lenguaje.

Todo lenguaje se basa en un conjunto de símbolos, con los cuales se construyen las estructuras del mismo.

En el Lenguaje Natural (en nuestro caso el Español) el conjunto de símbolos para el lenguaje escrito es el alfabeto, las letras y los símbolos de puntuación :

Símbolos del Alfabeto = {A ... Z, a ... z, ; , : , (,), . , }

Para el lenguaje hablado los símbolos son los sonidos a partir de los cuales se construyen los fonemas.

Con la combinación de estos símbolos vamos construyendo cadenas (sucesiones finitas de símbolos) y debido a las reglas y definiciones del propio lenguaje sabemos si una cierta cadena pertenece al lenguaje y si tiene significado dentro del mismo. Las cadenas del lenguaje natural son las palabras.

El estudio del lenguaje natural se hace bajo los aspectos: Sintáctico (de forma), Semántico (de significado) y Fonológico (sonidos de las palabras).

El lenguaje natural surge de la necesidad de comunicación entre los seres humanos. Los estudiosos del mismo van descubriendo las reglas que lo rigen y analizando su evolución en el tiempo.
Es muy difícil la formalización del lenguaje natural, dada, por ejemplo, una regla ortográfica, no falta la lista de palabras que son excepciones. En el aspecto semántico hay palabras que tienen distintos significados dependiendo del contexto en que se encuentran o del país donde se usen. Esto da lugar a la ambigüedad que es propia de todo lenguaje natural.

Además, el lenguaje natural va cambiando a través del tiempo. El español antiguo es muy diferente del moderno, a pesar de que los cambios no son arbitrarios y responden a evoluciones culturales y lingüísticas que pueden representarse con ciertas reglas de transición, éstas no pueden ser muy precisas.

1.1.2 Lenguajes Formales

En ámbitos como el de las matemáticas o el de la computación, donde se requiere de un lenguaje muy preciso y sin ambigüedades, se utilizan lenguajes especialmente construidos para los que se describe formalmente tanto su sintaxis como su semántica.
Esto es, se crea una notación especial para los símbolos del alfabeto y se describe el conjunto de reglas para operar con esta notación o para transformar los símbolos. Esto es lo que hace que su presentación sea de caracter formal.

En el caso de la computación, se utilizan los lenguajes de programación, que son construidos **"para la descripción de algoritmos y estructuras de información".**[1] Esta descripción tiene que hacerse de manera muy clara y precisa para que una computadora realice una determinada función. Todo lenguaje de programación requiere de una descripción formal en los aspectos sintáctico y semántico.

Los lenguajes naturales y los formales están íntimamente relacionados ya que con estos últimos pretendemos representar objetos y situaciones del mundo real y precisar lo que significan en el lenguaje natural.

1.1.3 Lenguajes de Primer Orden

Los Lenguajes de Primer Orden, a diferencia de otros lenguajes formales han adquirido una enorme importancia en las matemáticas por tres razones: su simplicidad, su precisión y su gran poder expresivo. En ellos es posible formular todos los enunciados dotados de sentido en las matemáticas así como sus reglas de razonamiento. Sus fundadores -Frege, Peano, Russell y Hilbert entre otros - nos muestran que para llevar a cabo dicha tarea no se requiere mas que un número reducido de símbolos y reglas simples que rigen su empleo.

Para entender con claridad los principios de estos lenguajes examinemos primero la composición de una teoría matemática. Esta toma en consideración ciertas categorías de objetos y, dentro de ellas, ciertos objetos particulares, ciertas propiedades que pueden tener los objetos y ciertas relaciones y operaciones que pueden existir entre ellos.

Un ejemplo es la Aritmética la cual estudia las propiedades de los números naturales. La operación fundamental es la sucesión que a cada número natural x le asocia su sucesor inmediato sx.

[1] Elisa Viso. Lenguajes de Programación I. UNAM. FACULTAD DE CIENCIAS. 1987. MEXICO. pp.1

Otro ejemplo es la Geometría, en la que son considerados tres categorías de objetos (puntos, lineas y planos) y dos relaciones fundamentales (incidencia y orden). Todas las proposiciones geométricas se pueden enunciar aludiendo tan sólo a estos objetos y relaciones.

En los Lenguajes de primer orden se define un alfabeto compuesto por los símbolos para denotar a los objetos, a las operaciones y relaciones entre ellos y una serie de reglas para construir y manipular a las cadenas del lenguaje llamadas fórmulas.

La Lógica de primer orden tiene dos aspectos : su sintaxis y su semántica. El aspecto sintáctico trata con lo que son las fórmulas admitidas por la gramática del lenguaje y el aspecto semántico trata con el significado que se le da a los símbolos en las fórmulas.

Así pues, la definición de un alfabeto y las reglas para construir y manipular las cadenas obtenidas a partir de sus símbolos, es lo que constituye un Lenguaje Formal.

1.2 SINTAXIS

1.2.1 Alfabeto

Definimos el siguiente Alfabeto :

S I M B O L O S :

Variables	=	u w x y z
Puntuación	=	( ) ,
Cuantificadores	=	∀ Σ
Conectivos	=	∨ & ¬ -> <->
Constantes	=	a b c
Operadores	=	f g h
Predicados	=	r s t

variables y constantes :

Para denotar a los objetos, se utilizan los símbolos variables y constantes. Los símbolos constantes denotan objetos en particular : **mesa, a, maria.** En el alfabeto hemos definido los símbolos constantes **a, b** y **c.**

Los símbolos variables se consideran como indeterminados y representan objetos cualesquiera (de cierta categoría), por ejemplo, podemos denotar mediante el símbolo **x** a cualquier elemento de la categoría individuos, mediante **y** a cualquier elemento de la categoría máquinas, etc. En el alfabeto que hemos definido escogimos como variables los símbolos **u w x y y z.**

Operadores :

Para denotar a las operaciones entre los objetos se utilizan los símbolos operadores.

Enunciados de la Aritmética como la suma de los objetos **0** y **1** se escriben: **+(0,1).** En el caso de los símbolos operadores está asociado a cada símbolo operador **f** un número natural gr(f) al que se le llama grado de **f.** El grado indica el número de objetos sobre el cual actúa el operador. En el enunciado **+(0,1)**, **+** es símbolo operador de grado 2 pues la operacion que representa es entre dos objetos **0** y **1.** La función grado le asocia a **+** el número natural 2 de la siguiente forma: gr(+) = 2.

Predicados :

Para denotar a las relaciones entre los objetos y a las propiedades de los mismos se utilizan los símbolos predicados.

Enunciados de la Lógica como el objeto a tiene la propiedad P se escriben: P(a), relaciones entre objetos como los objetos a y b están en la relación R se escriben : R(a,b), etc.

En el caso de los símbolos predicados está asociado a cada símbolo predicado r un número natural or(r) al que se le llama orden de r, que indica el número de objetos que están relacionados. En el enunciado R(a,b), R es símbolo predicado de orden 2 pues la relación que representa es entre dos objetos a y b. La función orden le asocia a R el numero natural 2 de la siguiente forma : or(R) = 2.

Simbolos de Puntuación :

Nótese que tanto en +(0,1) como en R(a,b) hemos utilizado símbolos que arriba hemos definido como de puntuación : ( ) ,
Estos son, los parentesis (izquierdo y derecho) y la coma. Sirven para construir correctamente las cadenas del lenguaje y se rigen mediante reglas sintácticas como la siguiente :

Toda cadena que tenga un símbolo predicado r representando la relación entre n objetos debe tener la siguiente sintaxis :

r(obj1,obj2, ...,objn)

Esto es, el símbolo predicado, el paréntesis que abre, los objetos separados por comas terminando en un paréntesis que cierra.

Si P y R son predicados y x,y,z son variables :

P(x)	significa	x tiene la propiedad P
R(x,z)	significa	x y z están en la relación R
P(x,y,z)	significa	x, y y z están en la relación P

Se tienen, de esta manera, ciertas proposiciones elementales llamadas atómicas a partir de las cuales se construyen todas las demás por medio de las operaciones lógicas.

Conectivos y Cuantificadores :

Las operaciones lógicas como la negación, conjunción, disyunción, implicación, doble implicación y las cuantificaciones universal y existencial, se representan por medio de los símbolos especiales como sigue :

¬	no
&	y
v	o
->	implica
<->	doble implica
∀	para todo
Σ	existe

Por ejemplo, si P y Q son proposiciones y x es un símbolo variable :

P&Q	significa	P y Q
¬P	significa	no P
ΣxP(x)	significa	existe un objeto x tal que tiene la propiedad P
P(x)->Q(y)	significa	que el objeto x tenga la propiedad x, implica que el objeto y tenga la propiedad Q

Las variables, los conectivos, cuantificadores y los símbolos de puntuación, salvo pequeñas variaciones, son comunes a todos los lenguajes de primer orden.

Por el contrario, los símbolos que se usan para las constantes, operadores y predicados varían en cada caso.
Tres ejemplos de selecciones de símbolos constantes, operadores y predicados son los siguientes :

i) En un alfabeto para la Teoría de Conjuntos :

constantes	:	∅ (conjunto vacío)
operadores	:	no hay
predicados	:	∈ (pertenencia, de orden 2) = (igualdad, de orden 2)

En este caso los objetos son los conjuntos y los elementos de los mismos.
Los símbolos predicados son los que nos dan la posibilidad de relacionar estos objetos y así construir expresiones en el lenguaje.
Suponiendo que denotamos por a a un elemento y por A a un conjunto, podemos construir expresiones como las siguientes :

Expresión	Interpretación
a ∈ A	el elemento a pertenece al conjunto A
A = ∅	el conjunto A es vacío

Además, dentro del lenguaje hay siempre ciertas convenciones para la notación. En el caso de la teoría de conjuntos los elementos se denotan con letras minúsculas y los conjuntos por letras mayúsculas. La relación de pertenecia se hace entre elementos y conjuntos y la de igualdad entre conjuntos.

ii) En el alfabeto de la Aritmética de Peano :

constantes : 0 (cero)
1 (unidad)

operadores : + (suma, de grado 2)
* (producto, de grado 2)
s (sucesor, de grado 1)

predicados : = (igualdad, de orden 2)

En este caso los objetos son los números naturales. A través de los símbolos operadores +, *, s definimos a las operaciones de suma y producto entre los números y a la función sucesor.
A través de el símbolo predicado = denotamos a la relación de igualdad entre los objetos.
Suponiendo que denotamos por n a un número natural cualquiera, podemos construir expresiones como las siguientes :

Expresión	Interpretación
s(0) = 1	el sucesor del cero es la unidad
s(1) = s(s(0))	el sucesor de la unidad es el sucesor del sucesor del cero
n * 0 = 0	el producto de cualquier natural por cero es igual a cero
s(n) = n + 1	el sucesor de un natural es igual a ese natural mas la unidad
s(x,0) * +(1)	esta expresión es del lenguaje pero no tiene significado

Nótese que los símbolos se pudieron haber representado en forma distinta, pudimos haber escogido para representar a la suma el símbolo suma, el símbolo # o cualquier otro.

iii) En el alfabeto de la Mitología Griega :

constantes : zeus
ares
semele
dionisio

operadores : no hay

predicados : padre_de (orden 2)
madre_de (orden 2)
hermanos (orden 2)
diosa (orden 1)

En este caso los objetos son los personajes de la mitología griega. A través del símbolo predicado **padre_de** de grado 2 denotamos a la relación entre dos objetos donde uno es padre del otro. Análogamente, denotamos las relaciones de **ser madre** y de **ser hermanos**. A través del símbolo **diosa** denotamos la propiedad de un objeto de ser diosa.

Con lo anterior podemos construir expresiones como las siguientes :

Expresión	Interpretación
padre_de(zeus,ares)	zeus es padre de ares
padre_de(zeus,dionisio)	zeus es padre de dionisio
madre_de(semele,dionisio)	semele es madre de dionisio
hermanos(ares,dionisio)	ares y dionisio son hermanos
diosa(semele)	semele es una diosa

Una vez definido el alfabeto, pasemos a ver que cadenas podemos construir con él.

Se llama Expresión a toda cadena (sucesión) finita de símbolos del alfabeto. Nos ocuparemos ahora de un tipo particular de expresiones llamadas términos.

1.2.2 Términos

Definición .-
Un término es una expresión que sirve para denotar individuos u operaciones entre éstos. Son términos:

a) Las variables
b) Las constantes
c) Las expresiones de la forma f(t1 ... tn) cuando f es un operador de grado n y t1, ... ,tn son términos.
d) una expresión es término solo si lo es con base en a),b) y c).

Para saber si una expresión es término es muy importante fijarse, por un lado, que esté construida con símbolos del alfabeto y por otro que esté sintácticamente bien construida. (Buena parentización, los simbolos separados por comas).

En el caso del alfabeto que hemos definido, los siguientes son ejemplos de términos :

x	es término por ser variable
0	es término por ser constante
*(0,1)	es término por ser de la forma f(t1, ... , tn) donde * es símbolu operador de grado n=2 y 0,1 son términos por ser constantes

Los siguientes son ejemplos de expresiones que NO son terminos :

s(0,1))	No es término porque es una expresión mal parentizada, le sobra un paréntesis derecho
*(0,1,0)	No es término porque el símbolo operador se definió de grado 2 y aquí aparece de grado 3
+(j,m)	No es término porque los símbolos j y m no están definidos en el alfabeto

Verifica si las siguientes expresiones son o no términos, en caso que no lo sean, indica cuál es la falla.

s(1,s(s(0)))

0

Y

+(0,x)

+((0,1)

*(1,0))

+0,y))

z(+(0,1))

*(z,0)

+(n,1)

+(s(0),*(+(1,1),y))

+((n,2)

+(1,s(0))

En los ejemplos anteriores, nos hemos fijado solamente en la sintaxis de las expresiones, esto es, si estaban bien construidas de acuerdo a la definición de término con la que empezamos esta sección.

En la Lógica, además de esto nos interesa darle interpretación a las expresiones en el lenguaje. En el caso de los términos, como mencionábamos arriba, denotan individuos y operaciones entre éstos. Veamos las interpretaciones de los términos para algunos ejemplos arriba presentados.

Recordemos antes las interpretaciones de los símbolos que hicimos en el Alfabeto :

Las símbolos variables u,w,x,y,z representan números naturales
Las símbolos constantes 0 y 1 representan al cero y a la unidad
Los símbolos funcionales +, *, s representan a las operaciones de suma(de orden 2), producto(de orden 2) y la función sucesor (de orden 1).

De esta forma las interpretaciones son las siguientes :

Término	Interpretación en el lenguaje
0	constante 0
y	variable y
+(1,y)	suma de la unidad con cualquier número natural
*(1,0)	producto de la unidad con el cero
s(0)	sucesor del cero
+(1,s(0))	suma de la unidad con el sucesor del cero
+(s(0),*(+(1,1),y))	suma del sucesor del cero con el producto de la suma de la unidad con la unidad y cualquier número natural

Dentro de otra teoría matemática damos distintas interpretaciones pues el universo de objetos varía en cada caso.

Con lo anterior, hemos terminado de analizar las expresiones llamadas términos.

Nos ocuparemos ahora de otro tipo de expresiones, de las expresiones llamadas fórmulas.

1.2.3 Fórmulas

Definición .-
Una fórmula es una expresión que sirve para afirmar o negar hechos.

Son fórmulas :

a) Las expresiones de la forma r(t1 ... tn) cuando r es un predicado de grado n y t1, ... ,tn son términos. A éstas se les llama atómicas.

b) La expresiones de la forma ¬A, AvB, A&B, A->B, A<->B, ΣxA, y ∀xA cuando A y B son fórmulas y x es una variable.

c) una expresión es fórmula solo si lo es con base en a) y b).

Los conectivos y cuantificadores se jerarquizan como sigue :

1) ¬ 2) Σ, ∀ 3) &, v 4) ->, <->

Esto significa que en una expresión se pueden omitir paréntesis, los cuales se pueden reponer obedeciendo la jerarquía (o precedencia) de izquierda a derecha. Para restaurar paréntesis, en primer lugar se ponen paréntesis a las fórmulas mínimas que siguen a la negación (primer elemento de la jerarquía). En segundo lugar, se colocan paréntesis a las fórmulas mínimas que siguen a los cuantificadores y así sucesivamente hasta acabar con las implicaciones. En caso de que se tengan dos conectivos del mismo rango, los paréntesis se reponen de izquierda a derecha.Así por ejemplo, la expresión :

AvB&C corresponde a ((A v B) & C)

y la expresión :

A&B -> AvC & (B <-> C&¬D) a ((A&B) -> ((AvC) & (B <-> (C&¬D))))

De acuerdo a esta convención, no siempre es posible eliminar todos los paréntesis de una fórmula, como es el caso de :

R(x) -> (S(x) -> T(x))

Se sobrentiende que toda fórmula o término satisface la definición correspondiente, esto es, que no hay más fórmulas ni terminos que los caracterizados por estas definiciones.

Dado un alfabeto, el Lenguaje de Primer Orden correspondiente consiste de todas las fórmulas que se pueden construir con él.

En el caso del alfabeto que hemos definido las siguientes son ejemplos de fórmulas:

i) $\forall x \forall y (=(+(x,y),+(y,x)))$

Que se lee: "Para todo x, para todo y, la suma de x con y es igual a la suma de y con x"

Que en la Aritmética de Peano es la ley conmutativa de la suma para cualesquiera dos números naturales. Además se acepta que la igualdad se escriba entre los términos : $+(x,y) = +(y,x)$

ii) $\forall x =(s(x),+(x,1))$

Que se lee: "Para todo natural x, el sucesor de x es igual a la suma de x con la unidad"

Decíamos al principio que las fórmulas sirven para afirmar o negar hechos. En los dos ejemplos anteriores hemos afirmado hechos dentro de la aritmética. Afirmar (o negar) una fórmula es darle un valor de verdad, decir si es verdadera o falsa.

Esto no sucede con los términos, un término no puede ser afirmado o negado. Así por ejemplo, El término $+(0,1)$ denota la suma de las constantes 0 y 1 pero no podemos darle un valor de verdad; sería como decidir si la operación suma es o no verdadera. En cambio, a la expresión $=(+(0,1),1)$ que interpretamos como "**la igualdad de la suma de 0 y 1 con el 1**" sí podemos darle un valor de verdad. Esta expresión es una fórmula.

Como ya mencionamos antes, el Lenguaje de la Lógica está muy relacionado con el Lenguaje Natural. Con el pretendemos representar objetos y situaciones del mundo real y precisar lo que una expresión lógica significa en el lenguaje natural.
Gran parte del trabajo en Lógica consiste en la traducción (cuando esta es posible) del lenguaje a otro.

A continuación veremos mas ejemplos de fórmulas y mostraremos como funcionan los conectivos lógicos partiendo de enunciados del lenguaje natural.
Así que por lo pronto dejaremos de lado la Aritmética de Peano.

Partamos del enunciado :

Peano es Hombre

Este enunciado es la afirmación de un hecho. Está enunciando que el objeto Peano tiene la propiedad de ser hombre.
Para traducir este enunciado a una fórmula lógica, tenemos que representar al objeto y a la propiedad. Escojamos para esto al símbolo constante **peano** para Peano y al predicado H para **es hombre**. Así, la representación en lógica quedaría denotada por :

H(peano)

Sigamos con el enunciado :

Todos los Hombres son Matemáticos

para traducirlo, utilicemos M(x) para denotar **es matemático**. Así, el enunciado lo representaremos por :

$$\forall x(H(x) \rightarrow M(x))$$

Que es la fórmula interpretada como **Para toda x, si x es hombre, entonces x es matemático.** En general, un enunciado de la forma **Todo A es B** será susceptible de ser representado por la cuantificación universal de una implicación.

Sea el siguiente enunciado :

Algunos Hombres son Matemáticos

este enunciado se traduce a la lógica en la siguiente fórmula :

$$\exists x(H(x) \ \& \ M(x))$$

Es decir, **existe un individuo tal que es hombre y es matemático.** En general, un enunciado de la forma **Algún A es B** será representado por la cuantificación existencial de una conjunción (&) de predicados.

Consideremos ahora el caso en que se tiene la cuantificación universal y la negación. Un ejemplo es el siguiente :

Ningún Hombre es Matemático

Este enunciado lo representamos por :

$$\neg\exists x(H(x) \ \& \ M(x))$$

y también por :

$$\forall x(H(x) \rightarrow \neg M(x))$$

ya que el enunciado lo podemos expresar indicando que no existen individuos que sean a la vez hombres y matemáticos o indicando que, para todo individuo, si es hombre entonces no es matemático.

De la misma manera, el enunciado :

No todos los Hombres son Matemáticos

Lo representamos por :

$$\neg\forall x(H(x) \rightarrow M(x))$$

y también por :

$$\exists x(H(x) \ \& \ \neg M(x))$$

Más adelante demostraremos las equivalencias de estas formulaciones.

Consideremos ahora una serie de ejemplos de cómo podemos formalizar enunciados un poco más complejos.

Sean los siguientes predicados :

Es(x)	x escapará
Ob(x)	x pone manos a la obra
Co(x)	x es confiable
Er(x)	x comete un error
Nu(x)	x anda en las nubes
Cu(x)	x es culpable

y sean las constantes :

holmes : Holmes
watson : Watson
moriarty : Moriarty
crumm : Crumm

Los enunciados y su formalización son:

Moriarty es culpable si Crumm lo es

Cu(crumm) -> Cu(moriarty)

Watson es confiable solo si Holmes no pone manos a la obra

Co(watson) -> ¬Ob(holmes)

Moriarty escapará si y solo si Holmes comete un error

Es(moriarty) <-> Er(holmes)

Si Holmes comete un error o si Watson anda en las nubes, Moriarty escapará

Er(holmes) v Nu(watson) -> Es(moriarty)

CAPITULO 2: SEMANTICA

2.1 ASIGNACIONES DE VERDAD

En el capítulo 1 presentamos a los símbolos del alfabeto como símbolos del lenguaje, esto es, como objetos sintácticos. De igual forma, los términos y fórmulas se presentaron sintácticamente a través de definiciones que dan las reglas de construcción de estas cadenas del lenguaje de la lógica de primer orden.
En este capítulo 2 estudiaremos estos mismos objetos semánticamente, esto es, desde el punto de vista de su significado.

El concepto de verdad de una fórmula no es absoluto, es relativo a su significado, se determina por la interpretación que se de a los símbolos que aparecen en ella. Una interpretación consiste de: a) un dominio D, sobre el cual variarán las variables y tomarán significado los símbolos de la fórmula, b) una asociación que asigna un individuo del dominio D a cada constante de la fórmula, c) una asociación que defina una función de D en D para cada operador de la fórmula y d) una asociación que aigne una relación en D a cada predicado de la fórmula.

A toda fórmula de la Lógica de Primer Orden con una interpretación específica, se le asigna un valor de verdad, es decir, se dice si la fórmula es verdadera o si es falsa bajo esa interpretación. Formalicemos esta idea :

Sea μ = {F | F es fórmula}, el conjunto de todas las fórmulas atómicas. Sea {v,f} el conjunto de valores de verdad donde interpretamos a v como verdadero y a f como falso. Una asignación de verdad para el conjunto μ es una función :

$$\grave{a} : \mu \longrightarrow \{v,f\}$$

que asigna v ó f a cada elemento de μ. Si $F \in \mu$, $\grave{a}(F) = v$ ó $\grave{a}(F) = f$. Por ejemplo, si tenemos la fórmula :

Cu(c)

Que hemos interpretado por :

Crumm es Culpable

y à es una asignación, entonces :

$$\grave{a}\ (Cu(c)) = v \qquad \text{ó} \qquad \grave{a}\ (Cu(c)) = f$$

De esta forma, la función asignación asegura que toda fórmula con una interpretación específica tenga asociado un único valor de verdad, siendo éste verdadero o falso.
Por facilidad, diremos que una fórmula es verdadera cuando su valor de verdad sea v y diremos que es falsa cuando su valor de verdad sea f.

En las fórmulas que hemos presentado que son traducciones de los enunciados referentes a Holmes y Watson, hemos visto como se usan los conectivos lógicos, a continuación se les presentará como **nombres de funciones semánticas**, esto es, asociándoles un significado a través de una función de verdad.

Negación.

Cuando a una fórmula se le antepone el simbolo ¬, el resultado se denomina negación de la fórmula. La palabra no en el lenguaje Español se acostumbra colocarla con el verbo del enunciado, pero en lógica es frecuente encontrarla separada de la fórmula sobre la que actúa.

Así, el aplicar la negación a la fórmula Cu(c), resulta la fórmula:

¬ Cu(c)

que intrepretada en el lenguaje natural es:

Cruma no es Culpable

El valor de verdad de una fórmula negada, está determinado por el valor de verdad de la fórmula misma, esto es, de la operación de negar una fórmula, resulta una fórmula falsa (verdadera) si la fórmula es verdadera (falsa).

De esta forma, podemos asociar a la negación una función de verdad, que denotaremos también con el símbolo ¬. Una función que va de los valores de verdad a los valores de verdad :

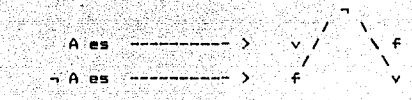
¬ : (v,f) --> (v,f)

donde

¬(v) = f La negación de una fórmula verdadera es una fórmula falsa

¬(f) = v La negación de una fórmula falsa es una fórmula verdadera

Estas condiciones de verdad se pueden representar en un árbol semántico de la siguiente manera :



O bien mediante la siguiente tabla :

A	¬A
v	f
f	v

recordemos que la palabra semántico se refiere a significado; tanto el arbol semántico como la tabla de la negación, muestran el significado de ¬.

Este arbol tiene dos ramas que corresponden a los dos posibles valores para una fórmula A. Al final de cada ruta está indicado el valor de verdad de ¬A correspondiendo al valor asignado a A en esa rama.

Conjunción.

Ahora consideraremos como, conocidos los valores de verdad de dos fórmulas se puede determinar el valor de verdad de su conjunción. En Español esto lo podemos hacer mediante la palabra y; por ejemplo, en el enunciado :

Crumm es Culpable y Moriarty Escapará

la palabra **y** la utilizamos para conjuntar los enunciados **Crumm es Culpable** y **Moriarty Escapará.** El enunciado resultado de la conjunción se traduce a notación lógica en lo siguiente :

Cu(c) & Es(m)

La conjunción, representada por el conectivo &, actúa sobre dos fórmulas, convenimos en que el valor de verdad de una fórmula de este tipo queda determinado por el valor de verdad de las dos fórmulas en cuestión, del mismo modo que en español: la conjunción es verdadera si y solo si ambas componentes son verdaderas.

De esta forma, podemos asociar a la conjunción una función de verdad denotada con el símbolo & que va de los pares de valores de verdad de las fórmulas a los valores de verdad :

$$\& : \{(v,v),(v,f),(f,v),(f,f)\} \longrightarrow \{v,f\}$$

donde

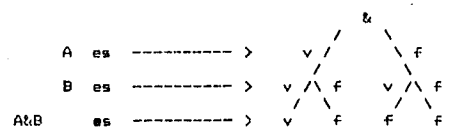
&(v,v) = v La conjunción de dos fórmulas verdaderas es una fórmula verdadera

&(v,f) = f La conjunción de una fórmula verdadera con una fórmula falsa es una fórmula falsa

&(f,v) = f La conjunción de una fórmula falsa con una fórmula verdadera es una fórmula falsa

&(f,f) = f La conjunción de dos fórmulas falsas es una fórmula falsa

De nuevo, podemos representar en un arbol semántico el significado de la conjunción :



O mediante la siguiente tabla :

A	B	A&B
v	v	v
v	f	f
f	v	f
f	f	f

Los valores de verdad indicados al final de cada rama están determinados por los valores asignados a A y a B en esa misma rama.

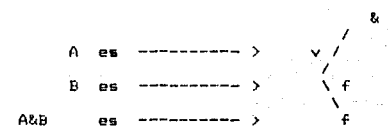
En este caso tenemos cuatro ramas, que son las cuatro posibles combinaciones para los valores verdadero y falso de las fórmulas A y B.
Notese que el único caso en que la conjunción de dos fórmulas es verdadera se da cuando ambas son verdaderas.

Veamos un ejemplo. Sean las fórmulas y sus correspondientes asignaciones:

A : Es(m) à (A) = v

B : ¬ Cu(c) à(B) = f

Para saber el valor de verdad de la fórmula A&B, si empezamos a recorrer el arbol desde la raiz (donde está el simbolo de la conjunción &) y seguimos la rama donde A es verdadera y B es falsa, encontramos que A&B es falsa como se muestra a continuación :



Disyunción.

Así como los valores de verdad de dos fórmulas determinan el valor de verdad de su conjunción, también determinan la verdad o falsedad de su disyunción. Una disyunción se enuncia en Español diciendo por ejemplo:

Holmes comete un Error o Watson anda en las Nubes

la palabra o la utilizamos para hacer la disyunción de los enunciados **Holmes comete un Error** y **Watson anda en las Nubes.** El enunciado resultado de la disyunción se traduce a notación lógica en lo siguiente :

Er(h) v Nu(w)

Que la fórmula A v B sea verdadera significa que al menos una de las dos, A o B es verdadera.

A la disyuncion se le asocia una una función de verdad denotada con el simbolo v que va de los pares de valores de verdad de las fórmulas a los valores de verdad :

$$v : \{(v,v),(v,f),(f,v),(f,f)\} \longrightarrow \{v,f\}$$

donde

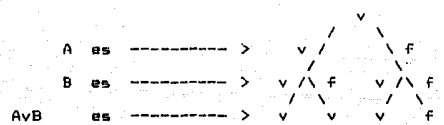
v(v,v) = v La disyunción de dos fórmulas verdaderas es una fórmula verdadera

v(v,f) = v La disyunción de una fórmula verdadera con una fórmula falsa es una fórmula verdadera

v(f,v) = v La disyunción de una fórmula falsa con una fórmula verdadera es una fórmula verdadera

v(f,f) = f La disyunción de dos fórmulas falsas es una fórmula falsa

El arbol semántico para la disyunción es el siguiente :



y la tabla de verdad correspondeinte :

A	B	AvB
v	v	v
v	f	v
f	v	v
f	f	f

Los valores de verdad indicados al final de cada rama están determinados por los valores asignados a A y a B en esa misma rama.

En este caso tenemos también cuatro ramas, que son las cuatro posibles combinaciones para los valores verdadero y falso de las fórmulas A y B.
Nótese que el único caso en que la disyunción de dos fórmulas es falsa se da cuando ambas son falsas.

En el caso de que ambas fórmulas son verdaderas, hemos definido su disyunción A v B como verdadera, a esto se le llama o inclusivo. Si hubiéramos decidido en este caso hacer A v B falsa, habríamos definido o exclusivo. Los dos tipos de disyunción son comunes en Matemáticas, los mostraremos a través de los siguientes ejemplos :

(i) $x \leq y \ \ o \ \ x \geq y$ **o inclusivo**

(ii) $x = \pm 2$ **o exclusivo**

En el primer ejemplo (i), se puede dar el caso de que las dos condiciones sean verdaderas haciendo verdadera a su disyunción, este es el caso cuando $x = y$, $x \leq y$ es verdadera y $x \geq y$ también lo es.

En cambio en el segundo ejemplo (ii), la proposición $x = \pm 2$ indica que x es igual a 2 o que x es igual a -2 no pudiendo darse el caso de que ambas sean verdaderas a la vez. En este ejemplo la exclusión está dada por el contexto.

En matemáticas no es necesario tener dos símbolos para representar a los dos tipos de disyunciones, ya que siempre se puede expresar el o exclusivo con ayuda de los otros conectivos.
Por ejemplo en:

$$x <= y \ \ o \ \ x >= y$$

si queremos excluir el caso cuando $x = y$, simplemente lo escribimos:

$$x <= y \ \ o \ \ x >= y \ \ \& \ \ \neg(x <= y \ \& \ x >= y)$$

En el lenguaje natural el uso de la **o** como inclusivo o exclusivo se da por el contexto en que aparece. Por ejemplo, en el menú de un restaurant que ofrece 'comida corrida' casi siempre se presentan las opciones a escoger de la siguiente manera :

Menu

Sopa del día
ó
Sopa de verduras
ó
Sopa de pasta

En este contexto, sabemos que no podemos pedir las tres sopas a la vez, tenemos que escoger una sola como opcion para comer. Este es el caso de una **o exclusiva** ya que la sopa que escojamos excluye a las demás.
Sin embargo, en ninguna parte del menu viene aclarado que se trata de una **o exclusiva** así que si nos quisiéramos pasar de vivos podríamos exigir las tres sopas apelando a esta falta de aclaración.

Implicación.

Muchas veces queremos expresar que la verdad de un enunciado implica la verdad de otro. Por ejemplo,

Moriarty es Culpable si Crumm lo es

Watson es confiable solo si Holmes no pone manos a la obra

En estos enunciados estamos expresando la implicación mediante las palabras **si, solo si,** en la lógica esto se expresa mediante el conectivo **->**.
En una fórmula del tipo A -> B se le llama antecedente a la primera A y consecuente a la segunda B.
Que A -> B sea verdadera es afirmar que se cumple B si sucede A (B si A) o se interpreta también como : Si A entonces B.

A la implicación se le asocia una función de verdad, denotada por el símbolo -> que va de los pares de valores de verdad de las fórmulas a los valores de verdad :

-> : {(v,v),(v,f),(f,v),(f,f)) --> {v,f}

donde

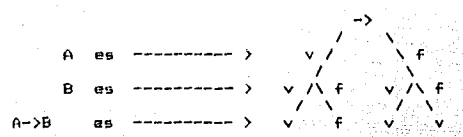
->(v,v) = v La implicación de dos fórmulas verdaderas es una fórmula verdadera

->(v,f) = f La implicación de una formula verdadera con una fórmula falsa es una fórmula falsa

->(f,v) = v La implicación de una fórmula falsa con una fórmula verdadera es una fórmula verdadera

->(f,f) = v La implicación de dos fórmulas falsas es una fórmula verdadera

El arbol semántico para la implicación es el siguiente :



y su tabla es :

A	B	A->B
v	v	v
v	f	f
f	v	v
f	f	v

Los valores de verdad indicados al final de cada rama están determinados por los valores asignados a A y a B en esa misma rama.

Nótese que hemos definido a la implicación como verdadera en los casos en que la primera fórmula A es falsa, independientemente del valor de B. Esto quiere decir que si el antecedente es una fórmula falsa, puede implicar cualquier cosa haciendo verdadera a la implicación.

Por ejemplo, la implicación :

Si yo pudiera volar, entonces <cualquier enunciado>

es siempre una implicación verdadera ya que de un antecedente falso puede seguir cualquier cosa.

En los otros casos, una implicación es verdadera cuando tanto antecedente como consecuente son verdaderos y es falsa cuando el antecedente es verdadero y el consecuente falso.

Doble Implicación.

Muchas veces queremos afirmar no solo que A -> B es verdadera sino que B -> A también lo es. Esto lo hacemos generalmente mediante la frase si y solo si y lo expresamos en matemáticas en frases como una condición necesaria y suficiente.

x+2 = 3 si y solo si x = -1

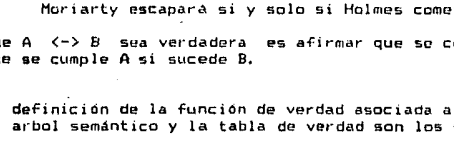
Para que un número sea divisible por 15 es necesario y suficiente que sea divisible por 3 y por 5

Moriarty escapará si y solo si Holmes comete un error

Que A <-> B sea verdadera es afirmar que se cumple B si sucede A y que se cumple A si sucede B.

La definición de la función de verdad asociada a la doble implicación, el arbol semántico y la tabla de verdad son los siguientes :

<-> : {(v,v),(v,f),(f,v),(f,f)} --> {v,f}



A	B	A<->B
v	v	v
v	f	f
f	v	f
f	f	v

2.2 VARIABLES Y CUANTIFICADORES

En esta sección analizaremos el papel que juegan las variables y los cuantificadores en las fórmulas.

2.2.1 Definiciones

Cuando una variable aparece en una fórmula, se dice que la variable ocurre en la fórmula y se refiere a ella como una **ocurrencia** o una **presencia**.

Sean las fórmulas :

[i] $\Sigma x(\forall y \in(y,x))$

[ii] $\forall y \in(y,x)$

en [i] ocurren las variables x y y, cada una de ellas ocurre dos veces (es decir, aparece en la fórmula dos veces), x ocurre como variable del cuantificador Σ y dentro del predicado $\in(y,x)$. En [ii] ocurren también las variables x y y, x ocurre una vez y y ocurre dos veces.

Volviendo al ejemplo que mencionamos al principio del alfabeto de la teoría de Conjuntos, interpretemos estas fórmulas en esa teoría (al símbolo predicado $\in$ como pertenecia y a las variables x y y como elementos y/o conjuntos cualesquiera).

Así, la fórmula [i] se interpreta como :

Existe un conjunto tal que todo conjunto pertenece a él.

la fórmula [ii] sólo puede interpretarse en el lenguaje natural como un enunciado incompleto :

Todo conjunto es miembro de ____.

No podemos completar este enunciado sin saber quien es x. En casos como éste diremos que la variable x ocurre **libre** en la fórmula $\forall y \in(y,x)$.

Por el contrario, en la fórmula [i], ninguna variable ocurre **libre** en $\Sigma x(\forall y \in(y,x))$.

Pasemos a definir formalmente la noción de variable libre. Definiremos esta noción de tres modos distintos y complementarios.

Definición 1

Sea x una variable y sean α y β fórmulas.

1) Si α es una fórmula atómica y x es un símbolo de α, entonces toda ocurrencia de x en α es libre.
Si α es una fórmula atómica y x NO es un símbolo de α, entonces x NO ocurre libre en α.

2) Una ocurrencia de x es libre en ¬α <-> x ocurre libre en α.

3) Una ocurrencia de x es libre en α v β <-> x ocurre libre en α ó x ocurre libre en β.

4) Una ocurrencia de x es libre en ∀zα <-> x ocurre libre en α y x \= z.

Aplicando la definición anterior no es difícil cerciorarse de que la variable x ocurre libre en ∃yR(x,y) y no ocurre libre en ∀x∃yR(x,y). Un problema con esta definición es que no da lugar a una definición sencilla del concepto de variable acotada. Un segundo problema es que no deja ver que una variable puede ocurrir libre y acotada en una misma fórmula. Para ello es necesario hacer referencia a las presencias de una variable de una fórmula.

Definición 2

Sea S una sucesión s0 ... sn de símbolos primitivos y sean i,j tales que 0≤i≤j≤n. Se denota Sij a la subsucesión si ... sj. Sea α una fórmula. α es una subfórmula de S <-> hay una subsucesión Sij de S tal que α = Sij.
Sea s un símbolo. s ocurre en la posición i de S <-> s = si.

1) Sea x una variable y 0 ≤ k ≤ n. sk es una presencia acotada de x en S <-> hay una subfórmula Sij de S tal que i<k≤j, sk=x y Sij=∀xα o Sij=∃xα.

2) Sea x una variable y 0 ≤ k ≤ n. sk es una presencia libre de x en S <-> sk=x y la presencia no es acotada.

En esta definición estamos presentando a las fórmulas como cadenas (sucesiones) donde los símbolos que las conforman ocurren en cierta posición. Presentemos a manera de ejemplo la siguiente fórmula :

```
posiciones :    0 1   4 6        11 13
                | |   | |         | |
          α :   ∀ x R(x,y) -> R(x,y)
```

longitud(α) = 14

Donde anotamos las posiciones del cuantificador universal ∀ y de las variables x y y. En esta fórmula, la variable x ocurre en las posiciones 1,4 y 11 y la variable y ocurre en las posiciones 6 y 13.

En la fórmula α la variable x tiene presencias libres y presencias acotadas; s1 y s4 son presencias acotadas y s11 es libre. Mientras que la variable y tiene todas sus presencias libres (s6 y s13).

Como se puede observar, las variables acotadas siempre ocurren en expresiones de la forma ∀xα ó Σxα donde α es una fórmula. Es por ello que presentamos una tercera definición.

Definición 3

Sea α una fórmula.
En la expresión ∀xα (Σxα respectivamente) se dice que α es el alcance del cuantificador ∀x (Σx respectivamente).

Una ocurrencia de una variable x en una fórmula es acotada <-> x es la variable de un cuantificador o está dentro del alcance de un cuantificador ∀x ó Σx en la fórmula.

En los casos restantes se dice que la ocurrencia es libre.

En nuestro ejemplo la primera ocurrencia de la variable x (s1) es acotada porque es variable del cuantificador ∀ :

```
                 1
                 |
α :    ∀xR(x,y)->R(x,y)
```

La segunda ocurrencia (s4) es también acotada porque está dentro del alcance del cuantificador ∀ como se muestra a continuación :

```
                4
                |
∀xR(x,y)->R(x,y)
|
∀x (x  )
```

El alcance del cuantificador ∀ va de la posición 1 a la posición 7 (que es la posición del paréntesis que cierra).

La primera ocurrencia de la variable y (s6) es libre ya que no es variable de ningún cuantificador ni está dentro de ningún alcance :

```
                  6
                  |
∀xR(x,y)->R(x,y)
|
∀x (  y)
```

Aunque la ocurrencia de la variable está dentro de los paréntesis, el cuantificador ∀ está cuantificando a la variable x, no así a la variable y.

La tercera ocurrencia de la variable x (s11) es libre ya que no está dentro del alcance del único cuantificador que podría acotarla :

por último, la segunda ocurrencia de la variable y (s13) también es libre por no ser variable de cuantificador ni estar dentro de ningún alcance :

Definamos ahora el conjunto de variables libres.

Definición 4

Sea V(α) = {x | x es una variable que ocurre en α} el conjunto de variables de α.
Se define el conjunto VL(E) de variables libres de E por recursión como sigue :

1) Si E es una fórmula atómica, entonces VL(E) = V(E).

2) Si E = ¬α, entonces VL(E) = VL(α).

3) Si E = α ∨ β, entonces VL(E) = VL(α) ∪ VL(β).

4) Si E = ∀xα ó E = ∃xα, entonces VL(E) = VL(α) − {x}.

A partir de las definiciones de variables libres y acotadas se caracterizan a las fórmulas como enunciados y se introducen nuevas nociones como las de **fórmula cerrada, cerradura universal** y **cerradura existencial**, de las que hablaremos a continuación.

2.2.2 Enunciados

Una fórmula α es enunciado <-> VL(α) = ∅, esto es, si no tiene ocurrencias libres de variables.

Volviendo a los dos ejemplos de fórmulas ([i] y [ii]) que presentamos al principio de la sección, recordemos que la primera no tenía variables libres :

$$\Sigma x(\forall y \in (y,x))$$

Existe un conjunto tal que todo conjunto pertenece a él.

y ésta, deacuerdo a la definicion, es un enunciado. En cambio, en la segunda fórmula :

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

Todo conjunto es miembro de ____.

decíamos que no podíamos completar este enunciado, esto es porque la variable x ocurre libre. De acuerdo a la definición, esta fórmula NO es enunciado. A las formulas que no contienen variables libres se les denomina también **Fórmulas Cerradas.**

2.2.3 Cerradura Universal y Cerradura Existencial

Definición

Si F es una fórmula, entonces $\forall(F)$ denota la **cerradura universal** de F, que es la fórmula cerrada que se obtiene añadiendo un cuantificador universal para cada variable que tenga una ocurrencia libre en F. Análogamente, $\Sigma(F)$ denota la **cerradura existencial** de F, que se obtiene añadiendo un **cuantificador existencial** para cada variable que tenga una ocurrencia libre en F.

El orden en que se van añadiendo los cuantificadores es arbitrario. En los ejemplos que se muestran a continuación se añaden recorriendo la fórmula de izquierda a derecha.

Ejemplos

Fórmula F	$P(x,y)\&Q(x)$
$\forall(F)$	$\forall x \forall y(P(x,y)\&Q(x))$
$\Sigma(F)$	$\Sigma x \Sigma y(P(x,y)\&Q(x))$
Fórmula F	$\forall y \in (y,x)$
$\forall(F)$	$\forall y \forall x \in (y,x)$
$\Sigma(F)$	$\forall y \Sigma x \in (y,x)$
Fórmula F	$\forall x1R(x1,x2) \rightarrow \Sigma x2 \forall x1R(x1,x2)$
$\forall(F)$	$\forall x1 \forall x2R(x1,x2) \rightarrow \Sigma x2 \forall x1R(x1,x2)$
$\Sigma(F)$	$\forall x1 \Sigma x2R(x1,x2) \rightarrow \Sigma x2 \forall x1R(x1,x2)$

2.3 SUSTITUCIONES Y UNIFICACION

2.3.1 Definiciones

Definición

Una sustitución θ es un conjunto finito de parejas ordenadas $((x_1,t_1),\ldots,(x_r,t_r))$ en el que cada x_j es una variable (las variables $x_1 \ldots x_r$ todas distintas entres sí) y cada t_j un término distinto de x_j. En lugar de (x_j,t_j) se acostumbra escribir x_j/t_j.
Sea θ una sustitución y E una expresión. Eθ denota la expresión que se obtiene al sustituir en E cada ocurrencia de x_j por t_j para $1 \leq j \leq r$.

Ejemplos

Sean E = p(x,y,f(a)) y θ=(x/b,y/x)
Eθ = p(b,x,f(a))

Sean E = p(x,y)&q(x,h(x)) y θ=(x/zeus,y/ares)
Eθ = p(zeus,ares)&q(zeus,h(zeus))

Definición

Sean $\theta=(u_1/s_1,\ldots,u_m/s_m)$ y $\sigma=(w_1/t_1,\ldots,w_r/t_r)$ dos sustituciones. La composición $\theta\sigma$ de θ y σ es la sustitución que se obtiene del conjunto:

$$\theta\sigma = \{u_1/s_1\sigma,\ldots,u_m/s_m\sigma,w_1/t_1,\ldots w_r/t_r\}$$

habiéndose eliminado las parejas $(u_j,s_j\sigma)$ en caso de que $u_j=s_j\sigma$ y las parejas (w_j,t_j) en caso de que $w_j \in \{u_1,\ldots,u_m\}$ (para evitar doble asignación a una variable).

Un caso particular es el de la sustitución vacía $\emptyset$ (o {}) con la propiedad de que para toda sustitucion θ, $\theta\emptyset = \emptyset\theta = \theta$.

Ejemplos

Sean θ=(x/f(y),y/z) y σ=(x/a,z/b),
entonces $\theta\sigma$=(x/f(y),y/b,z/b)

Sea E = p(x,y,g(z)), entonces Eθ=p(f(y),z,g(z))
y (Eθ)σ = p(f(y),b,g(b))
Además E($\theta\sigma$) = p(f(y),b,g(b)) = (Eθ)σ

Sean E = p(x,y)&q(x,h(x))
θ_1=(x/f(z)) θ_2=(x/a,y/b) θ_3=(x/1,y/z,z/x)

Eθ_1 = p(f(z),y)&q(f(z),h(f(z)))
Eθ_2 = p(a,b)&q(a,h(a))
Eθ_3 = p(1,z)&q(1,h(1))

E$\theta_1\theta_3$ = [p(f(z),y)&q(f(z),h(f(z)))]θ_3 = p(f(x),z)&q(f(x),h(f(x)))
E$\theta_3\theta_1$ = [p(1,z)&q(1,h(1))]θ_1 = p(1,x)&q(1,h(1))

Definición

Sean E y F dos expresiones. E y F son **variantes** entre si existen dos sustituciones θ y σ tales que $E=F\sigma$ y $F=E\theta$.

Observese que las variantes se obtienen cambiando variables por variables (es decir, el tránsito de una variable a otra no se lleva a cabo sustituyendo una variable por algo que no sea una variable).

Ejemplos

E=p(x,y)&q(x,h(z)) y F=p(y,x)&q(y,h(u)) son variantes una de la otra ya que si θ={x/y,y/x,z/u) y σ={y/x,x/y,u/z), entonces $E\theta$=p(y,x)&q(y,h(u))=F y $F\sigma$=p(x,y)&q(x,h(z))=E.

Sean E = p(f(x,y),g(z),a), F = p(f(y,x),g(u),a) y θ={x/y,y/x,u/z), σ={y/x,x/y,z/u). Entonces:
$F\theta$ = p(f(x,y),g(z),a) = E
$E\sigma$ = p(f(y,x),g(u),a) = F

A diferencia de los ejemplos anteriores, p(x,y) y p(x,x) no son variantes entre si.

Definición

Sean E y F dos términos. Una sustitución θ en un **unificador** de E y F si y solo si $E\theta=F\theta$.

Un unificador es **mínimo** (o unificador **más general**: **umg**) si para cualquier otro unificador σ existe una sustitución Γ tal que $\sigma=\theta\Gamma$ (es decir, θ involucra menos sustituciones). Esta definición se extiende a un conjunto finito de Terminos.

Ejemplos

Sean E = p(f(x),z), F = p(y,a) y σ={y/f(a),x/a,z/a}. Claramente σ es un unificador de E y F ya que $F\sigma$ = p(f(a),a) = $E\sigma$. Pero hay un unificador que involucra menos sustituciones: θ={y/f(x),z/a} y es por esto que θ es el unificador mas general (umg) de E y F. Nótese que $\sigma=\theta$(x/a).

Sean E = p(f(x),a) y F = p(y,f(w)). E y F no son unificables ya los argumentos **a** y **f(w)** no pueden unificarse.

Se sigue de la definición de **unificador mínimo** que si θ y σ son ambos umg's de dos expresiones E y F, entonces $E\theta$ es variante de $F\sigma$.

A continuación presentamos un algoritmo denominado **Algoritmo de Unificación.** Este descubre si las expresiones consideradas son o no unificables. Cuando sí lo son, encuentra un unificador mínimo. Por el contrario, cuando las expresiones no son unificables, el algoritmo lo detecta y termina sin más.

La idea intuitiva que hay detrás de este algoritmo es la siguiente. Supongamos que queremos unificar dos expresiones E1 y E2. Imaginemos dos apuntadores, cada uno apuntando al primer símbolo de cada expresión. Los apuntadores se van moviendo hacia la derecha de cada expresión hasta que apunten a símbolos diferentes. Se hace entonces un intento por unificar a E1 y E2 empezando con estos símbolos a través de una sustitución θ. Si el intento tiene exito, el proceso se repite con las expresiones sustituidas E1θ y E2θ. Si el intento falla, se concluye que las expresiones no son unificables. Si los apuntadores llegan al final de las epresiones, la composición de todas las sustituciones son el umg de E1 y E2.

Definición

Sean $E=e_1...e_m$ y $F=f_1...f_n$ dos fórmulas atómicas. Compárense signo por signo ambas expresiones hasta localizar la menor j tal que $e_j \neq f_j$. Sean s y t los terminos que ahí comienzan (es decir, tales que el primer signo de s es e_j y el primer signo de t es f_j). Se define la **diferencia** de E y F como el conjunto D=(s,t).

Ejemplo

Sean E = p(f(x),h(y),a) y F = p(f(x),z,a) entonces la diferencia de E y F es el conjunto D=(h(y),z)

2.3.2 Algoritmo de Unificación

1.- Sea $k=0$ y $\theta_0=\emptyset$.

2.- Si $E\theta_k=F\theta_k$ el proceso de unificación ha terminado y θ_k es un unificador mínimo de E y F. Por el contrario, si $E\theta_k \neq F\theta_k$, fórmese la diferencia D_k de $E\theta_k$ y $F\theta_k$.

3.- Si la diferencia D_k es de la forma (x,t) con x una variable y t un término en el que no ocurre x, entonces fórmese la sustitución $\theta_{k+1}=\theta_k(x/t)$ y repítase el paso (2) para $k+1$.

4.- Si D_k no tiene la forma descrita en (3), entonces el algoritmo termina y se concluye que E y F no son unificables.

Ejemplos

1) Sean E = p(f(a),g(x)) y F = p(y,y).

$\theta_0=\emptyset$
$D_0=\{f(a),y\}$ por el inciso 3, $\theta_1=\{y/f(a)\}$

$E\theta_1$ = p(f(a),g(x)) y $F\theta_1$ = p(f(a),f(a)). Como $E\theta_1 <> F\theta_1$, se forma la diferencia D_1.

$D_1=\{g(x),f(a)\}$. Como D_1 no tiene la forma descrita en el inciso 3, se concluye que las expresiones E y F NO son unificables.

2) Sean E = q(a,x,f(g(y))) y F = q(y,f(z),f(z))

$\theta_0=\emptyset$
$D_0=\{a,y\}$. Por el inciso 3, $\theta_1=\{y/a\}$

$E\theta_1$ = q(a,x,f(g(a))) y $F\theta_1$ = q(a,f(z),f(z)). Como $E\theta_1 <> F\theta_1$, se forma D_1.

$D_1=\{x,f(z)\}$. Por el inciso 3, $\theta_2=\{x/f(z)\}$

$E\theta_2$ = q(a,f(z),f(g(a))) y $F\theta_2$ = q(a,f(z),f(z)). Como $E\theta_2 <> F\theta_2$, se forma D_2.

$D_2=\{z,g(a)\}$. Por el inciso 3, $\theta_3=\{z/g(a)\}$

$E\theta_3$ = q(a,f(g(a)),f(g(a))) y $F\theta_3$ = q(a,f(g(a)),f(g(a))). Como $E\theta_3=F\theta_3$ el proceso de unificación termina en este punto. Un unificador mínimo de E y F es:

$$\theta_1\theta_2\theta_3 = \{y/a\}\{x/f(z)\}\{z/g(a)\} = \{y/a,x/f(g(a)),z/g(a)\}$$

El Algoritmo de Unificación no solo decide si un conjunto de expresiones son o no unificables, sino que en su proceso va construyendo las sustituciones que conforman el unificador mínimo y en caso de llegar con exito al final, su salida es precisamente el umg.

2.4 INFERENCIAS

2.4.1 Introducción

Definición

Un conjunto de Premisas implica una conclusión si no hay un conjunto de valores de verdad para el cual todas las premisas sean verdaderas y la conclusión falsa.

Ejemplo. Las premisas A v B y ¬A implican la conclusión **B.**

		premisas		conclusión
A	B	A v B	¬A	B
v	v	v	f	v
f	v	v	v	v
v	f	v	f	f
f	f	f	v	f

En el único caso (2a. línea) en el que las dos premisas son verdaderas, la conclusión es tambien verdadera.
Por otro lado, las premisas A -> B y B no implican la conclusión A ya que hay un caso (2a. línea) en el que las dos premisas son verdaderas y la conclusión es falsa.

		premisas		conclusión
A	B	A -> B	B	A
v	v	v	v	v
f	v	v	v	f
v	f	f	f	t
f	f	v	f	f

Así que, decir que un conjunto de premisas implica una conclusión es negar que se pueda encontrar un caso en el que todas las premisas sean verdaderas y la conclusión falsa. A tal caso se le llama **contraejemplo.**

Decir que las premisas

A -> ¬B
¬C -> A

implican la conclusion

B -> C

es decir que ninguno de los 8 casos concernientes a la verdad o falsedad de las letras A, B y C es un contraejemplo: ninguno de ellos hace verdaderas a las dos premisas y falsa a la conclusion.

La manera mas directa de verificar lo anterior es la de hacer la tabla de verdad (como se hizo para los ejemplos) tomando en cuenta los valores de las dos premisas y la conclusion y en cada uno de los casos verificar que nunca suceda que las dos premisas sean verdaderas y la conclusión falsa. Pero como el número de casos se duplica cada vez que una nueva letra se añade, esta manera de verificarlo resulta muy laboriosa y practicamente imposible de llevar a cabo.[2]

Aunque en el ejemplo que estamos presentando hay solo tres letras sentenciales y por lo tanto 8 son los casos a verificar, se pierde mucho esfuerzo en una comprobación 'a mano' donde hay que ir viendo en cada caso si es o no un contraejemplo.

Utilizando un poco de sutileza, podríamos reducir la labor; en realidad los casos que nos interesan son aquellos en que las premisas son todas verdaderas, así que necesitamos de un método que elimine de entrada a todos los demás casos. Una técnica sistemática de esta índole está dada por el Método de **Arboles de Verdad** que a continuación se presenta.

[2] En general, con n letras sentenciales, en la que cada una puede ser verdadera o falsa independientemente de las demás, hay N = 2^n casos distintos que conciernen a la verdad o falsedad para toda n. De esta forma, cada vez que se añade una nueva letra, el número de casos se duplica.

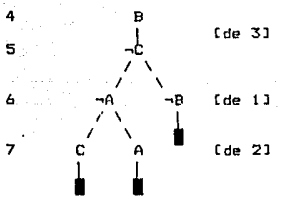
Número de letras: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ...
Número de casos : 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, ...

2.4.2 Método de Arboles de Verdad

A manera de ejemplo utilizaremos la inferencia ya mencionada :

$$\begin{array}{l} A \rightarrow \neg B \\ \neg C \rightarrow A \\ \hline B \rightarrow C \end{array}$$

El árbol de verdad de esta inferencia es como sigue :



Pasos para la construcción del árbol

Paso Uno. El primer paso en la construcción de este arbol es el de listar todas las premisas y la negación de la conclusión. Los contraejemplos son los casos en los que tanto las premisas como la negación de la conclusión son todas verdaderas. Queremos ver si hay alguno de estos casos.

Paso Dos. No hace ninguna diferencia cual de las líneas 1, 2 o 3 escojamos primero para examinar. Comencemos con la línea 3. Cualquier contraejemplo deberá hacer esta línea verdadera. Por lo tanto, deberá hacer B -> C falsa y esto sucede si y solo si B es verdadera y C falsa (o ¬C verdadera) como se ilustra en la tabla de verdad :

B	C	B -> C
V	V	V
F	V	V
V	F	F
F	F	V

Esto lo indicamos escribiendo B y ¬C como líneas 4 y 5 del arbol. De esta forma hemos tomado en cuenta todos los casos posibles en los que ¬(B->C) es verdadera. Este procedimiento lo podemos resumir en la siguiente regla de inferencia en la que el * y la τ son letras sentenciales cualesquiera :

$$\neg\ (* \rightarrow \tau)$$
$$*$$
$$\neg\tau$$

Lo anterior se da por las siguientes equivalencias lógicas:

$$\neg(* \rightarrow \tau) \equiv \neg(\neg * \vee \tau)$$
$$= * \ \& \ \neg\tau$$

lo que da lugar a dos premisas:

premisa1: *
premisa2: ¬τ

donde el árbol:

*

representa el caso en que * es verdadera y el árbol:

*
¬τ

representa el caso en que ambas son verdaderas.

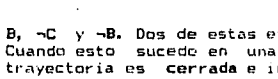
Paso Tres. Tomemos como siguiente la línea 1. Cualquier contraejemplo deberá hacer también esta línea verdadera. Aquí la regla de inferencia es:

Los casos en los que la implicación es verdadera son aquellos en los que el antecedente es falso o en los que el consecuente es verdadero (Un caso cae en las dos categorías).

*	τ	* -> τ
v	v	v
f	v	v
v	f	f
f	f	v

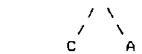
De esta forma, la bifurcación de esta regla para implicaciones tiene el sentido de que una implicación * -> τ es verdadera si y solo si el antecedente * es falso o el consecuente τ es verdadero (o las dos). Así transformamos A -> ¬B en ¬A v ¬B y añadimos al árbol una bifurcación escribiendo del lado izquierdo ¬A (negación del antecedente) y del derecho ¬B (el consecuente). Con esto hemos tomado en cuenta todas las posibilidades en los que la línea 1 se hace verdadera. Representando en la rama izquierda aquellos en que ¬A es verdadera y en la rama derecha aquellos en que ¬B es verdadera.

Paso Cuatro. Recorriendo el árbol desde su raíz vemos que en la trayectoria derecha tenemos lo siguiente :



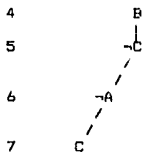
B, ¬C y ¬B. Dos de estas expresiones se contradicen entre sí: B y ¬B. Cuando esto sucede en una trayectoria del árbol se dice que la trayectoria es **cerrada** e indicamos este hecho marcando el final de la ruta con "■". Una trayectoria es cerrada si contiene una letra sentencial junto con su negación. Esto significa que es imposible que todas las premisas sean verdaderas en esa trayectoria y por lo tanto se cierra, o dicho de otra forma, es imposible encontrar un contraejemplo en esa trayectoria.

Paso Cinco. La única línea que falta por integrar al árbol es la línea 2. Cualquier contraejemplo deberá hacer esta línea verdadera de la misma forma que se hizo para las líneas 1 y 3. Como la línea 2 es una implicación de la forma ¬C -> A, la trataremos igual que se hizo para la línea 1 en el paso tres. Ahora, el asterisco representa a ¬C y la letra tao representa a A. Aplicando la regla, añadimos una bifurcación al final de cada trayectoria abierta, escribiendo C en el lado izquierdo y A en el derecho :

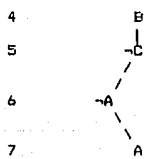


Esto nos da la línea 7.

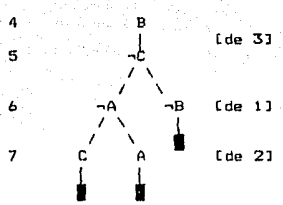
Paso Final. Sin tomar en cuenta la trayectoria que ya cerramos, nos fijamos en las otras dos trayectorias desde la raíz del árbol hasta sus nodos hojas. Cada una de éstas es una trayectoria cerrada. Una de ellas contiene a C y a ¬C :



y la otra contiene a A y ¬A :



Por lo tanto procedemos a cerrarlas con la marca de ∎. Como consecuencia de todos los pasos anteriores nos queda el árbol que presentamos al principio :



Todas Las trayectorias son cerradas. Se dice entonces que el árbol es cerrado y se ha probado que la inferencia es válida.

Las dos reglas de inferencia utilizadas están diseñadas para que cuando se analice una premisa, ésta sea transformada de tal manera que se tomen en cuenta todos los casos posibles en los que la premisa sea verdadera.

Las diferentes trayectorias del arbol representan los diferentes caminos posibles en los que las premisas con las que se construye el árbol son verdaderas y cada posibilidad está representada por una trayectoria. Si una trayectoria se cierra, entonces las posibilidades que representa no son en realidad posibles. Si todas las trayectorias se cierran, es imposible para todas las premisas ser verdaderas, es decir: **no hay contraejemplos.**

Se tendra un contraejemplo de una fórmula si se tiene una rama que muestre que la negación de la fórmula es satisfacible (una rama no cerrada). Un argumento:

$$\begin{array}{l} P_1 \\ P_2 \\ \cdot \\ \cdot \\ \cdot \\ P_n \\ \hline C \end{array}$$

coresponde a la fórmula:

$$P_1 \,\&\, P_2 \,\&\, \ldots \,\&\, P_n \rightarrow C$$

equivalente a:

$$\neg(P_1 \,\&\, P_2 \,\&\, \ldots \,\&\, P_n) \vee C$$

y su negacion equivalente a:

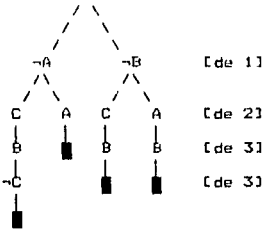
$$P_1 \& P_2 \& \ldots \& P_n \& \neg C$$

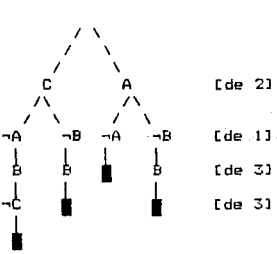
Entonces un contraejemplo corresponderá a un camino en que las premisas originales sean verdaderas y tambien ¬C -que por ser un conjugando adicional lo llamaremos tambien **premisa**, la premisa P_{n+1} - sea verdadera. Si cualquiera de las premisas al normalizarla, es a su vez una conjunción de fórmulas más simples, pues la partimos en esas premisas más simples:

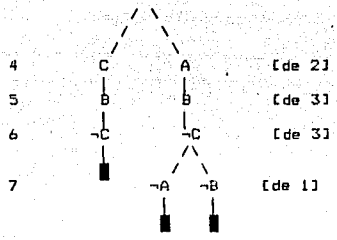
$$P_k \text{ equivalente a } P_{k1} \& P_{k2} \& \ldots \& P_{km}$$

La forma en que se escogieron las premisas para construir el árbol fue arbitraria ya que el orden en que se vayan seleccionando no afecta el resultado. A continuación se muestran algunos de los árboles que se generan al tomar en distinto orden las premisas del ejemplo.

```
1   A -> ¬B      [premisa]
2   ¬C -> A      [premisa]
3   ¬(B -> C)    [negacion de la conclusion]
```







Si nos fijamos, todos los árboles anteriores son cerrados, mostrando en cada caso que la inferencia es válida. La diferencia entre ellos es solo de forma, algunos se abren mas que otros. A manera de estrategia, se recomienda analizar primero las negaciones de implicaciones: ¬(φ -> τ). Esto hace que el árbol no se abra innecesariamente, aunque el método funciona independientemente del orden en que se analicen las premisas.

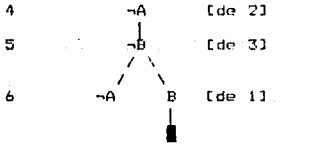
En contraste, mostraremos a continuación que sucede al examinar una inferencia **inválida**. Tomemos como ejemplo el siguiente:

A -> B
¬A

B

donde la conclusión B no es implicada por las premisas. El árbol de verdad que se construye es :

1 A -> B [premisa]
2 ¬A [premisa]
3 ¬B [negación de la conclusión]



La trayectoria abierta representa un contraejemplo: el caso en el que ambas premisas son verdaderas y la conclusión falsa. Para ver que caso es éste. nótese que las letras que aparecen en la trayectoria abierta son ¬A y ¬B. El contraejemplo es el caso en el que A y B son ambas falsas.

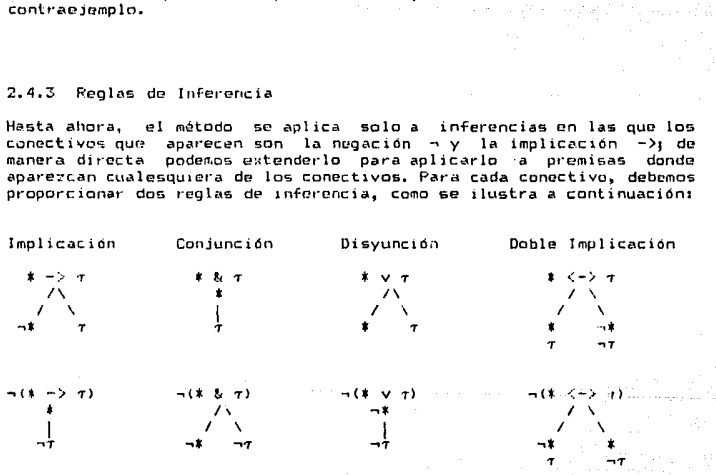
	premisas		conclusión
A B	A -> B	¬A	B
f f	v	v	f

El caso en que las premisas A -> B y ¬A son verdaderas pero la conclusión B falsa.

El metodo de Arboles de Verdad no solo muestra si una inferencia es invalida sino que indica para cada trayectoria abierta cual es el contraejemplo.

2.4.3 Reglas de Inferencia

Hasta ahora, el método se aplica solo a inferencias en las que los conectivos que aparecen son la negación ¬ y la implicación ->; de manera directa podemos extenderlo para aplicarlo a premisas donde aparezcan cualesquiera de los conectivos. Para cada conectivo, debemos proporcionar dos reglas de inferencia, como se ilustra a continuación:



Con estas reglas, podemos trabajar con fórmulas que involucren los demás conectivos lógicos. Por ejemplo, podemos utilizar el método para ver si la conclusión AvB se deriva de la premisa **A&B** utilizando las reglas correspondientes a la conjunción y a la negación de la disyunción.

2.4.4 Tautologías

El método de Arboles de Verdad puede utilizarse también para verificar si una cierta fórmula es tautología. Una tautología es una inferencia a partir de cero premisas, por lo que la fórmula a la que se busca el contraejemplo:

$$P_1 \& P_2 \& \ldots \& P_n \& \neg C$$

se reduce a:

$$\neg C$$

que es precisamente la negación de la fórmula original.

Para aplicar el método se empieza a construir el árbol con la negación de la fórmula y se continua como si se estuviera verificando la validez de una inferencia. Si el árbol se cierra, la fórmula es una tautología, en caso contrario, la fórmula no es tautología.

Ejemplo

Veamos si la fórmula **A -> (B -> A)** es una tautologia. Primero negamos la fórmula:

¬(A->(B->A)) = A&(B&¬A) = A&B&¬A

y aplicamos el método al resultado.

1 A [premisa]

2 B [premisa]

3 ¬A [premisa]

4 A [de 1]

5 B [de 2]

6 ¬A [de 3]

De este árbol, resultó una trayectoria cerrada, por lo que se concluye que la fórmula es una tautologia.

PARTE II

FUNCIONAMIENTO DE LOS SISTEMAS AUTOMATIZADOS

CAPITULO 3: SISTEMA DE LOGICA I

3.1 OPERACION DEL SISTEMA

3.1.1 Instalación

El sistema de Lógica I necesita para su operación de dos elementos:

- El intérprete de Prolog en su implementación de Arity/Prolog
- Los Programas correspondientes al sistema de Lógica

Cada uno de estos viene en un diskette. Así que para su instalación, basta con tener estos dos diskettes y copiarlos al disco.
Antes de proceder a copiarlos, crea un subdirectorio en tu disco duro y llámalo **Lógica**. Una vez creado el subdirectorio, trasládate a el y deberá aparecer el prompt del sistema operativo como :

C:\LOGICA>

Ahora copia los dos discos a este subdirectorio con el comando **copy**. Una vez realizado lo anterior, estará instalado el sistema.

3.1.2 Entrada al Sistema.

Para entrar al sistema de Lógica, basta con teclear desde Sistema Operativo **api** seguido de la tecla <return> de la siguiente forma:

C:\LOGICA> api <return>

Esto ocasionará la entrada al intérprete de Prolog. Una vez adentro aparecerá el prompt del intérprete :

?-

el intérprete esperará a que le des alguna instrucción. Para cargar los programas que conforman la parte I del sistema de Lógica, teclea enseguida del prompt lo siguiente :

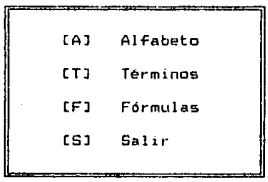
?-consult(logica1). <return>

En el intérprete de Prolog, todas las instrucciones terminan en punto, no olvides ponerlo al final. Con esta instrucción, Prolog 'leerá' todos los programas del sistema, este proceso es tardado. Sabrás que ha terminado cuando aparezca en la siguiente linea la palabra yes (si) y debajo el prompt ?- de Prolog.

Ahora, teclea la siguiente instrucción para entrar al sistema de Lógica :

?-logica1. <return>

La instrucción anterior ocasionará la llamada al sistema y desplegará el Menú principal a través de la siguiente pantalla :

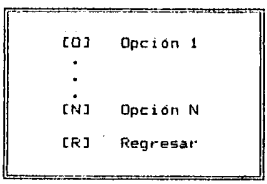


Desde este Menú, se accesan cada uno de los módulos: Alfabeto, Términos y Fórmulas o se termina la sesión (con la opción S) saliendo de nuevo a nivel de sistema operativo.

3.1.3 Manejo del Sistema

El sistema de Lógica está diseñado a base de Menús, cada uno tiene un conjunto de opciones de las que podrás elegir la que quieras dentro de las disponibles.

Todo menú tiene la siguiente estructura:



En el cuadro central, se despliegan las distintas opciones y el cursor que estará posicionado a la derecha de la leyenda :

Escoja una Opción :

esperará a que la selecciones. Para seleccionar la opción deseada, basta con teclear la letra que aparece a la izquierda de esa opción dentro de los paréntesis cuadrados. El sistema acepta en este tipo de menús la mayúscula y la minúscula de la letra como válidas para activar la opción escogida.

Al seleccionar una opción del menú, pueden suceder 2 cosas:

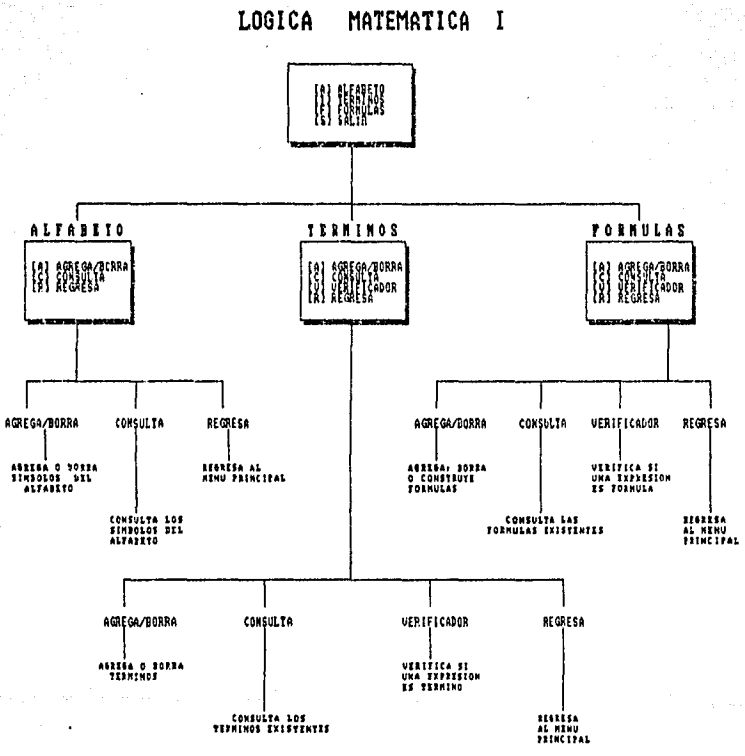
- El despliege de otro menú
- La ejecución de algún programa

Esto dependerá del nivel donde te encuentres en el sistema. Para regresar al menú anterior, o salir de cualquier programa estará siempre a disposición la opción de regreso con la letra r. Esta tecla siempre te regresará al nivel inmediato anterior de donde te encuentres en el sistema.

Dentro de los programas habrá también opciones a escoger. Casi siempre aparecerán en la última linea de la pantalla. Todas estas opciones se seleccionan también con la letra que esté indicada entre paréntesis cuadrados, solo que en estos casos el sistema sólo reconoce la letra indicada (siempre minúscula).

3.1.4 Organizacion de los Menus del Sistema

Al continuacion se presenta un arbol que ilustra la organizacion de los modulos del sistema.



3.2 MODULOS DEL SISTEMA

Mostraremos el funcionamiento del sistema de Logica I a traves de varios ejemplos. Para los módulos Alfabeto y Terminos seguiremos el ejemplo presentado en la sección de teoría referente al Alfabeto de la Aritmética de Peano (1.2.1 y 1.2.2). Utilizaremos el módulo Alfabeto para definir los símbolos de la aritmetica y el de Terminos para construir términos en ese lenguaje.

Para el modulo de Fórmulas ampliaremos el alfabeto y mostraremos su funcionamiento con los ejemplos que se dieron en la parte referente a la teoría (1.2.3). Utilizaremos este módulo para verificar expresiones y construir fórmulas.

A traves de estos ejemplos, describiremos paso a paso lo que el programa y el usuario van haciendo.

3.2.1 Modulo Alfabeto

Entra al sistema de Lógica I. (ver 3.1.2 Entrada al sistema).
Aparecerá en la pantalla el menú principal con las siguientes opciones:

[A] Alfabeto

[T] Términos

[F] Fórmulas

[S] Salir

Selecciona la opcion [A] para entrar al Módulo del Alfabeto. Aparecerá en pantalla el menú del Alfabeto con las siguientes opciones :

[A] Agrega/Borra

[C] Consulta

[R] Regresar

La primera opción permite manipular los símbolos del alfabeto, esto es, agregar nuevos símbolos o borrar existentes.
La segunda permite consultar todos los símbolos del alfabeto.
La tercera es para regresar al menú principal.

Consulta de los Símbolos del Alfabeto

Veamos que símbolos están definidos en el alfabeto. Para esto, selecciona la opción [C]. A continuación aparecen en la pantalla todos los simbolos del alfabeto y enseguida el sistema indica que presiones cualquier tecla para regresar al menú del Alfabeto con la leyenda :

Oprima cualquier tecla para regresar ...

Recuerda que los símbolos para las variables, cuantificadores, conectivos y símbolos de puntuación son símbolos que ya están fijos y serán los que utilizarás para cualquier alfabeto que se defina mas adelante.

Para el ejemplo que nos concierne, las variables denotadas por u,w,x,y,z representan números naturales.
A continuación definiremos los símbolos para las constantes, operadores y predicados.

Agregar o Borrar Símbolos del Alfabeto

Selecciona la opción [A]. En la parte inferior de la pantalla aparece lo siguiente :

S I M B O L O S :

[c] constantes [o] operadores [p] predicados

Estos son las tres clases de símbolos que se pueden modificar en el alfabeto. Selecciona la opción [c] para las constantes. A continuación aparecerán en pantalla hasta arriba los símbolos constantes que están definidos y abajo las opciones para manipularlos:

[a] agrega (para añadir algun símbolo),
[b] borra (para borrar algun símbolo existente),
[r] regresa (para regresar al menú del alfabeto).

Primero borremos los símbolos constantes que están definidos para después agregar los del ejemplo.

Para borrar, selecciona la opción [b]. El sistema espera a que escribas el símbolo a borrar, escribe a seguido de <return> y verás que ahora aparece la lista de símbolos omitiendo el símbolo a que acabas de borrar.
Repite el proceso para borrar los símbolos restantes : b,c,d y e.

Una vez borrados estos símbolos, verás que no aparece ninguno, esto quiere decir que en este momento el conjunto de símbolos constantes es vacío. Para salir del modo de borrado, presiona <return> como se indica.

A continuación, agreguemos los símbolos constantes que representarán al cero y a la unidad, escogiendo para esto los símbolos 0 y 1. Selecciona la opción [a]. El sistema espera a que escribas el símbolo a agregar, escribe 0 seguido de <return>, a continuación verás que se ha agregado a la lista. Siguiendo el mismo procedimiento agrega el símbolo 1.

A continuación trabajemos con los símbolos operadores. Para ello, regresa al menú del alfabeto seleccionando la opción [r]. Selecciona la opción [A] y escoge la correspondiente a los símbolos operadores. Aparecerá en la pantalla la lista de símbolos ya definidos así como las opciones para manipularlos :

```
[a] agrega      (para añadir algun símbolo),
[b] borra       (para borrar algun símbolo existente),
[g] grados      (muestra los grados asociados a cada símbolo)
[r] regresa     (para regresar al menú del alfabeto).
```

Antes de proseguir a agregar los símbolos operadores del ejemplo, borra los existentes (f,g,h). Esta opción funciona análogamente a la correspondiente en los símbolos constantes, (escoge la opción borra, escribe el símbolo a borrar seguido de <return> y te aparece la lista omitiendo el símbolo borrado).

A continuación, agreguemos los símbolos que representarán a las operaciones de suma, producto y a la función sucesor, escojamos para esto los símbolos: +, *, y s respectivamente.
Selecciona la opción [a] y enseguida el sistema espera a que escribas el símbolo a agregar, escribe + seguido de <return>. A continuación el sistema te pedirá que indiques el grado del símbolo, en este caso, la suma, que es una operación binaria (de dos) le corresponde el grado 2, escribelo y verás como aparece gr(+) = 2 indicando que la función grado asocia al símbolo + el número natural 2.
Enseguida aparece en la lista este nuevo símbolo. Repite el proceso anterior para agregar los símbolos * y s de grados 2 y 1 respectivamente.

Antes de continuar, es importante señalar que, por facilidad en la notación, cuando se agreguen símbolos operadores o predicados, el sistema guardará sus correspondientes grado y orden pero éstos no aparecerán en el símbolo como se acostumbra en algunos libros de texto (en algunas notaciones, se presenta al símbolo f de grado 1 como : f,1). Sin embargo, estos se tomarán en cuenta mas adelante.

Con esto ya tenemos definidos los símbolos operadores de nuestro ejemplo. Por último, agreguemos el símbolo predicado para representar la igualdad. Regresa al menú del alfabeto, escoge la opción para agregar símbolos predicados, dentro de ésta borra r, s y t y agrega el símbolo = de orden 2.

Consulta de Grados u Ordenes de Símbolos Operadores y Predicados

En los casos de los símbolos operadores y predicados, existe una opción para consultar los grados u órdenes asociados a cada uno. Para los operadores se consulta con la opción [g] y para los predicados con la opción [o].
Selecciónа la opción [g] dentro de los operadores y verás que aparece en la pantalla lo siguiente :

SIMBOLO	GRADO
+	2
*	2
s	1

Si quieres consultar el orden asociado al símbolo de igualdad, hazlo a través de la opción [o] dentro de los predicados.

En este momento tenemos ya definido el alfabeto correspondiente al Alfabeto de la Aritmética de Peano.
Regresa al menú del alfabeto y escoge la opción de [C] para verlo completo.

Si seguiste los pasos anteriores como se indicaron debiera aparecer el siguiente alfabeto :

S I M B O L O S :

Variables	:	u w x y z
Puntuación	:	( ) , .
Cuantificadores	:	∀ ∃
Conectivos	:	v & ¬ -> <->
Constantes	:	1 0
Operadores	:	s * +
Predicados	:	=

3.2.2 Módulo Términos

Desde el menú principal se accesa la opción del módulo Términos. Selecciónala y enseguida aparecerá en la pantalla el menú correspondiente a este módulo con las siguientes opciones :

[A] Agrega/Borra

[C] Consulta

[V] Verificador

[R] Regresar

La primera opción permite manipular los términos, esto es, agregar nuevos o borrar existentes.
La segunda permite consultar los términos existentes
La tercera es el verificador, sirve para verificar si una expresión dada es o no un término.
La última es para regresar al menú principal.

Verificador

En esta opción el usuario escribe una expresión y el sistema verifica si es o no término de acuerdo a la definición dada en la sección de teoría (1.2.2).

Utilicemos esta opción para verificar si las expresiones 0 y s(1,0) son términos.
Para ejecutar el verificador, escoge desde el menú del módulo de Términos la opción [V]. Al seleccionar esta opción, el sistema espera a que escribas la expresión a verificar, escribe 0 terminando en <return> y enseguida el sistema escribirá lo siguiente:

La constante 0 es Término

Como se definió el símbolo 0 como constante en la parte del alfabeto (3.2.1), el sistema lo reconoce como tal.

En la parte inferior de la pantalla aparecen las opciones correspondientes a verificar otra expresión o regresar al menú de términos. Selecciona [v] para la siguiente expresión. Escribe s(1,0) terminando en <return> y enseguida se desplegará el mensaje:

Expresión bien parentizada
Todos los símbolos están en el Alfabeto
El grado del símbolo s se definió como 1 y aquí aparece como 2
La expresión NO es Término

Como lo indica el mensaje, la expresión está bien parentizada y todos los símbolos estan en el alfabeto pero el grado del símbolo operador s se definió como 1 y en la expresión aparece actuando sobre 2 argumentos, por lo que se concluye que la expresión no es termino.

A continuación se da la lista de expresiones que se presentaron en la parte de teoría (1.2.2). Verifícalas con el sistema intentando anticipar si son o no términos.

```
s(1,s(s(0)))
'0
y
+(0,x)
+((0,1)
*(1,0))
+0,y))
z(+(0,1))
*(z,0)
+(n,1)
+(s(0),*(+(1,1),y))
+((n,2)
+(1,s(0))
```

Ahora regresa al menú de términos con la opción [r] regresa.

Agregar o Borrar Términos

Selecciona la opción [A]. Esta opción funciona igual a la del alfabeto, con la modalidad de que si intentas agregar una expresión que no sea término, el sistema no lo permitira.

Al entrar a esta opción, verás que no aparece ningun término en la lista, esto es para que comiences con la lista vacía y queden solo los que tu vayas agregando.
Selecciona algunos términos de la lista y agrégalos al sistema. Después de realizar lo anterior, regresa al menú de terminos.

Consulta de Términos

Selecciona la opción [C] desde el menú de terminos. A continuación se desplegarán en pantalla los términos que has agregado.

Si estas todavía en el módulo de terminos, regresa al menú principal para continuar con la siguiente sección.

3.2.3 Módulo Fórmulas

Desde el menú principal se accesa la opcion del módulo Fórmulas. Selecciónala y enseguida aparecerá en la pantalla el menú correspondiente con las siguientes opciones :

[A] Agrega/Borra

[C] Consulta

[V] Verificador

[R] Regresar

La primera opción permite manipular a las fórmulas, esto es, agregar nuevas, construir a partir de otras o borrar existentes.
La segunda permite consultar las formulas existentes.
La tercera es el verificador, sirve para verificar si una expresión dada es o no una fórmula.
La última es para regresar al menú principal.

Agregar o Construir Formulas

Empecemos por agregar algunas fórmulas al sistema. Para ello, selecciona desde el menu de Fórmulas la opción [A]. A continuación aparecera en la parte superior de la pantalla la fórmula [1] : H(x) y abajo las siguientes opciones :

[a] agrega (para añadir una nueva fórmula)
[c] construye (para construir nuevas fórmulas utilizando los conectivos)
[b] borra (para borrar fórmulas existentes)
[r] regresa (para regresar al menú de formulas)

Antes de proseguir aclaremos algunas cuestiones. Para facilitar el manejo de las fórmulas, a cada fórmula deberá estar asociado un numero de referencia como lo está en la fórmula hasta ahora definida, numerada como [1].

El número de la fórmula será su referencia y éste debe ser único, es decir, no puede haber dos o mas fórmulas con el mismo número.

Otra cuestión por aclarar es la referente a la sintaxis de las fórmulas. La fórmula definida hasta ahora es H(x), de acuerdo a la definición (1.2.3), esta no lo es ya que el símbolo H no es predicado. Para que lo fuera, habría que definirlo como tal en el alfabeto.

Como el objetivo de esta sección es trabajar con fórmulas y no queremos limitarnos al lenguaje de la Aritmética, para no hacer tedioso el uso del sistema haciendo que definas todos los símbolos que usaremos, en este módulo usaremos los siguientes símbolos para el alfabeto :

Símbolos:

variables	: u w x y z	(con o sin índices)
constantes	: cualquier sucesión de letras que empiece con minúscula (a .. z)	
operadores	: f g h	(con o sin índices)
predicados	: cualquier sucesión de letras que empiece con mayúscula (A .. Z)	

Los símbolos para los conectivos, cuantificadores y puntuación son los mismos que antes.
De acuerdo con esta convención, la fórmula definida H(x) ya cumple con la definición.

Agregar una Fórmula

Pasemos a agregar la siguiente fórmula :

M(x)

Para esto, selecciona la opción [a]. A continuación verás que aparece en la parte superior de la pantalla lo siguiente :

F3 : ∀ F4 : ∃ F5 : ∨ F6 : & F7 : ¬ F8 : -> F9 : <->

Como en el teclado de la máquina no son de fácil acceso los símbolos para los conectivos y cuantificadores, la teclas de función están programadas para que los escribas. Así por ejemplo para escribir el símbolo de cuantificación universal ∀ basta con presionar la tecla de función F3.Este mensaje es para inidcar a que tecla corresponde cada símbolo.

Después del mensaje anterior, el sistema te pide que escribas la fórmula, escribe M(x) terminando en <return> y enseguida el sistema te pide que escribas el número de referencia que le vas a dar a esta nueva fórmula, escribe 2 terminando en <return>. Por último, verás que la fórmula que agregaste aparece en la lista.

Siguiendo la misma secuencia anterior, agrega la fórmula ∀x(H(x)->M(x)) dándole como número de referencia el [3]. Para salir del modo agrega, simplemente presiona <return> como se indica.

Construir una Fórmula

A continuación agreguemos la fórmula **ΣxH(x) & M(x)** a través de la opción [c]. Seleccionala y verás que en la parte inferior de la pantalla aparece lo siguiente :

[a]	[b]	[c]	[d]	[e]	[f]	[g]
¬A	ΣxA	∀xA	A&B	AvB	A->B	A<->B

Estas son las opciones de los diferentes tipos de construcciones que se pueden realizar a partir de fórmulas ya definidas. Así por ejemplo, podemos construir una fórmula conjuntando las fórmulas A y B obteniendo como resultado la fórmula A&B.

Para construir la fórmula en cuestión, haremos primero la conjunción de H(x) y M(x) y enseguida aplicaremos al resultado el cuantificador existencial.

Para lo primero, como lo que queremos es obtener una fórmula del tipo A&B, escoge la opción correspondiente marcada con [d]. Una vez hecho esto, el sistema espera a que escribas el número de la fórmula A, escribe 1 seguido de <return> y aparecerá a la derecha la fórmula asociada a [1] que en este caso es H(x), enseguida el sistema espera a que escribas el número de referencia de la fórmula B, escribe 2 seguido de <return>, verás que también aparece la fórmula asociada a este número que es M(x).

Por último, el sistema te pide que escribas el numero que quieres darle a la nueva fórmula, escojamos el 4, escribelo y a continuación verás que la fórmula resultado de la conjunción de H(x) y M(x) estará agregada a la lista de fórmulas **[4] : H(x)&M(x)**.

Ahora, escoge de nuevo la opción [c] para aplicarle a esta nueva fórmula el cuantificador existencial.
Escoge la opción [b] que es la que realiza esta construcción. El sistema preguntará por la fórmula y por la variable para construir una del tipo ΣxA. Escribe 4 y la variable x. Como numero para la nueva fórmula escribe 5.

Al final de este proceso tendrás como fórmula :

[5] : Ex(H(x)&M(x))

Depende de como quieras ir agregando las fórmulas al sistema si lo haces a través de la opción [a] donde tendrás que escribir la fórmula completa o a través de [c] si quieres utilizar las existentes.

A manera de ejercicio agrega (con la opción que mas te plazca) las siguientes fórmulas :

Cu(crumm) -> Cu(moriarty)

¬Ob(holmes) -> Co(watson)

Es(moriarty) <-> Er(holmes)

Er(holmes) v Nu(watson) -> Es(moriarty)

Que son las referentes a los ejemplos de Holmes y Watson que dimos en la parte de teoria de fórmulas. (1.2.3).

Borrar una Fórmula

Para borrar una fórmula existente, selecciona la opción [b] y enseguida el sistema te pide el número de la fórmula a borrar y la remueve de la lista. Una vez borrada una fórmula, podra utilizarse de nuevo su número de referencia para otra fórmula.

Verificador

En esta opción el usuario escribe una expresión y el sistema verifica si es o no fórmula de acuerdo a la definición dada en la seccion de teoría (1.2.3) y tomando como alfabeto el descrito al principio de esta seccion.

Utilicemos esta opción para verificar si las expresiones **mortal(x)** y **Hombre(x)->Mortal(X)** son fórmulas.

Para ejecutar el verificador, escoge desde el menú de Fórmulas la opción [V]. Al seleccionar esta opción, el sistema espera a que escribas la expresion a verificar, escribe **mortal(x)** terminando en <return> y enseguida el sistema desplegará el siguiente mensaje:

Expresión bien parentizada
El Símbolo mortal NO es Predicado, no empieza con letra mayúscula

La expresión NO es Fórmula

Como se indica, la expresión está bien parentizada pero el símbolo **mortal** no puede ser predicado ya que no empieza con letra mayúscula, está interpretado como una variable y ésta no puede tener argumentos asociados. Por lo anterior se concluye que la expresión no es fórmula.

En la parte inferior de la pantalla aparecen las opciones correspondientes a verificar otra expresión o regresar al menú de fórmulas. Selecciona [v] para la siguiente expresión. Escribe **Hombre(x)->Mortal(x)** terminando en <return> y enseguida se desplegará el mensaje:

Expresión bien parentizada
Todos los símbolos están en el Alfabeto

La expresión es Fórmula

Como los predicados **Hombre** y **Mortal** empiezan en letra mayúscula, el sistema los reconoce como tales y además la expresión está sintácticamente bien construida (Predicado, operador implicación, Predicado), se concluye entonces que la expresión es Fórmula.

A continuacion se da la lista de expresiones, verificalas con el sistema intentando anticipar si son o no fórmulas.

```
∀x1(Hombre(x1)->¬Matematico(x1))

¬∀x(Hombre(x)->Matematico(x))

hombre(z3)

∀z(Estudia_logica(z)->Matematico(z))

¬∀z(Matematico(z)->Estudia_logica(z))
```

Ahora regresa al menú de fórmulas con la opción [r] regresa.

Consulta de Fórmulas

Para consultar la lista de fórmulas existentes, selecciona desde el menú de fórmulas la opción [C]. A continuación se desplegarán en la pantalla todas las fórmulas que agregaste con sus respestivos números de referencia.

Por último, si estas todavía en el módulo de fórmulas, regresa al menú principal.

CAPITULO 4: SISTEMA DE LOGICA II

4.1 OPERACION DEL SISTEMA

4.1.1 Instalación

El sistema de Lógica II necesita para su operación de dos elementos:

- El intérprete de Prolog en su implementación de Arity/Prolog
- Los Programas correspondientes al sistema de Lógica II

Cada uno de estos viene en un diskette. Así que para su instalación, basta con tener estos dos diskettes y copiarlos al disco.
En el mismo subdirectorio donde están los programas del sistema de Lógica I copia los programas del sistema de Lógica II con el comando copy. Una vez realizado lo anterior, estará instalado el sistema.

4.1.2 Entrada y Manejo del Sistema.

La entrada al sistema de Lógica II es análoga a la del Sistema de Lógica I (3.1.2), con la diferencia de que ahora es otro archivo el que se pide consultar. Una vez dentro del intérprete de Prolog, teclea enseguida del prompt lo siguiente :

?-consult(logica2). <return>

Cuando el sistema indique que ha terminado, teclea la siguiente instrucción para entrar al sistema de Lógica II:

?-logica2. <return>

La instrucción anterior ocasionará la llamada al sistema y desplegará el Menú principal a través de la siguiente pantalla :

L O G I C A II

[A] Alcance

[U] Unificación

[I] Inferencias

[C] Consulta Fórmulas

[S] Salir

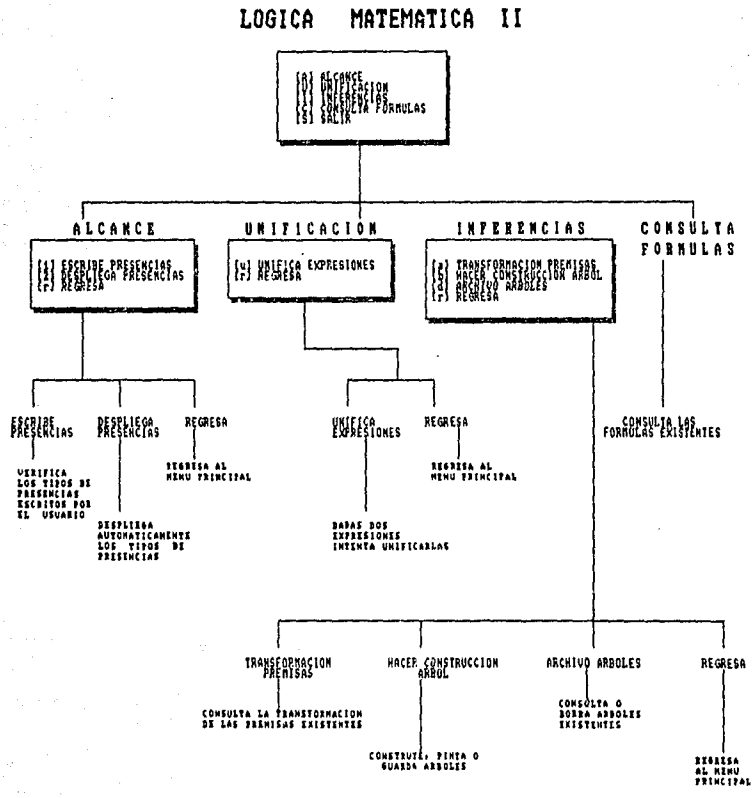
Escoja una Opción :

Desde este Menú, se accesan cada uno de los módulos: Alcance, Unificación, Inferencias y Consulta Fórmulas o se termina la sesión (con la opción S) saliendo de nuevo a nivel de sistema operativo.

El manejo del sistema de Lógica II es análogo al sistema de Lógica I (3.1.3). Esta diseñado a base de menus formados por conjuntos de opciones a elegir.

4.1.3 Organizacion de los Menus del Sistema

Al continuacion se presenta un arbol que ilustra la organizacion de los modulos del sistema.



4.2 MODULOS DEL SISTEMA

Mostraremos el funcionamiento del sistema de Lógica II a través de varios ejemplos. Para el módulo Alcance, el sistema tiene fórmulas ya definidas (que se consultan a traves de la última opción). Utilizaremos èste módulo para trabajar con una de ellas mostrando las presencias libres y acotadas de las variables, basándonos en la definición 2 de la sección de teoría (2.2.1).

Para el módulo Unificación, seguiremos los ejemplos presentados en la sección de teoría referente al Algoritmo de Unificación (2.3.2). Utilizaremos este módulo para aplicar este algoritmo a pares de expresiones que el usuario de como entrada.

Para el módulo Inferencias, seguiremos el ejemplo presentado en la sección de teoría referente al Método de Arboles de Verdad (2.4.2). Utilizaremos este módulo para verificar la validez de inferencias y tautologías a través de dicho método.

A través de estos ejemplos, describiremos paso a paso lo que el programa y el usuario van haciendo.

4.2.1 Módulo Alcance

Entra al sistema de Lógica II (ver 4.1.2 Entrada y Manejo del Sistema). Aparecerá en la pantalla el menú principal con las siguientes opciones :

[A] **Alcance**

[U] Unificación

[I] Inferencias

[C] Consulta Fórmulas

[S] Salir

Las tres primeras opciones son las referentes a los módulos de Alcance, Unificación e Inferencias. La cuarta permite consultar las fórmulas existentes que se utilizarán para el primer módulo. La última es la opción para salir del sistema.

El Objetivo del **Módulo Alcance** es el de analizar las variables de una fórmula; viendo si sus presencias son libres o acotadas. La fórmula a analizar la escribe el usuario o puede tomarse de las existentes en la base de datos. La fórmula es analizada de izquierda a derecha.
El análisis puede hacerse de dos maneras : en una el usuario escribe si las las presencias de las variables son libres o acotadas y el sistema indica aquellas presencias que estén mal indicando la falla.

En la otra el despliege es automático, es decir, dada una fórmula el sistema va recorriendo las presencias de las variables e indicando en cada una si la presencia es libre o acotada.

Pasemos a usar el sistema con un ejemplo.

Selecciona desde el menú principal la opción de. Alcance. A continuación aparecerán en la parte inferior de la pantalla las opciones para leer una fórmula o consultar una existente. Selecciona la correspondiente a **fórmula existente**. Enseguida, el sistema esperará a que escribas el número de referncia de la fórmula, escribe 6 seguido de <return> ya que es el número de la fórmula que usaremos de ejemplo.

Se escribirá en la pantalla la siguiente fórmula :

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

y se pintarán en tono mas fuerte todas las presencias de las variables x y y. A continuación aparecerán en la parte inferior de la pantalla las opciones a escoger :

[1] Escribe Presencias [2] Despliega Presencias [r] regresa

Primero pidamos la opción **Desplige de Presencias** para que el sistema nos vaya presentando automáticamente cada una de ellas. Para esto selecciona la opción [2] y verás que el sistema escribe de nuevo la fórmula e indica que la primera presencia de la variable x es acotada:

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

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

$\forall x$

La ocurrencia de la variable x
está dentro del alcance del cuantificador $\forall$

En la línea donde se pinta de nuevo la fórmula, el sistema va ir pintando cada presencia de la variable que va analizando. En una línea más abajo, el sistema va ir indicando si la presencia de la variable que está analizando está acotada por algún cuantificador o es libre.

En unas líneas más abajo, el sistema indica la presencia de la variable en cuestión y si ésta es acotada, indica el alcance del cuantificador que la acota.

En este caso, la presencia de la variable x es acotada por ser variable de un cuantificador.

Para continuar el proceso, el sistema pide que presiones cualquier tecla, hazlo y verás como ahora muestra la segunda presencia de la variable x :

```
∀x(R(x,y)->∀yR(x,y))

∀ (  x              )
```

La ocurrencia de la variable x
está dentro del alcance del cuantificador ∀

En este caso indica que la presencia está dentro del alcance del cuantificador ∀, porque está dentro del paréntesis y ∀ acota a x.

Presiona cualquier tecla para ver la siguiente :

```
∀x(R(x,y)->∀yR(x,y))
```

La ocurrencia de la variable y ES LIBRE

En este caso, pinta la presencia de la variable y en tono más fuerte e indica que es libre.

A continuación muestra la siguiente presencia de la variable y y es claro que es acotada porque es variable del cuantificador ∀y :

```
∀x(R(x,y)->∀yR(x,y))
      ∀y
```

La ocurrencia de la variable y
está dentro del alcance del cuantificador ∀

La siguiente es la presencia de la variable x, donde se ve claramente que está dentro del alcance del cuantificador ∀x :

```
∀x(R(x,y)->∀yR(x,y))
∀ (       x   )
```

La ocurrencia de la variable x
esta dentro del alcance del cuantificador ∀

Por último, el sistema muestra la presencia de la variable y indicando cómo está acotada por el cuantificador ∀y :

```
∀x(R(x,y)->∀yR(x,y))
      ∀  (  y)
```

La ocurrencia de la variable y
está dentro del alcance del cuantificador ∀

Una vez terminado este proceso, aparece de nuevo el menú de opciones. Ahora selecciona la opción [1] para que tu vayas indicando que tipo de presencia tiene cada variable.

Al seleccionar esta opción, se escribe de nuevo la fórmula y se pintan rayas debajo de cada variable para que tu vayas escribiendo que tipo de variable es cada una. Esto lo haces escribiendo una a en caso de que sea acotada o una l en caso de que sea libre. Después de escribir la letra correspondiente (a o l), presiona <return> para seguir a la siguiente raya.
Hazlo y después de escribir la última, el sistema pintará en tono más fuerte las variables que indicaste correctamente. Si fallaste en alguna, te mostrará el error.
Al final quedarán pintadas en tono más fuerte aquellas variables que indicaste bien.

A manera de ejemplo, supongamos que se escribió lo siguiente :

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

a a a a l l

El sistema pintará en tono más fuerte en la fórmula las primeras dos ocurrencias de la variable x porque se indicaron correctamente con una a como acotadas; después mostrará que la primera presencia de y es libre, pintará en tono más fuerte la segunda presencia de y y por último, mostrará en las dos últimas que son acotadas y no libres como está escrito.

Al final quedarán pintadas en tono más fuerte las presencias que se anotaron correctamente como se muestra a continuación :

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

a a a a l l

Una vez analizada esta fórmula, regresa al Menú Principal (a través de la opción [r] regresa) y consulta las fórmulas que están definidas en el sistema. Escoge algunas y analízalas como se hizo para el ejemplo.

Si quieres analizar alguna otra que no esté definida, entra al módulo de Alcance, escoge la opción [1] **leer fórmula** y escríbela. (Las teclas de función están programadas para que tengas de fácil acceso a los conectivos lógicos).

4.2.2 Módulo Unificación

El Módulo Unificación es la instrumentación del Algoritmo de Unificación (2.3.2) para dos expresiones. Se dan como entrada 2 expresiones y el sistema va unificándolas. Si tiene exito, el proceso termina con el despliege del unificador mas general y si falla, el proceso se interrumpe indicando que no fueron unificables.

Utilizaremos esta opción para los siguientes conjuntos de expresiones:

1) E = p(f(a),g(x))
   F = p(y,y)

2) E = q(a,x,f(g(y)))
   F = q(y,f(z),f(z))

Desde el Menú principal del sistema de lógica II se accesa esta opción. Selecciónala y enseguida aparecerá en la parte superior de la pantalla un mensaje indicándote que escribas la primera expresión. Despues de E = escribe p(f(a),g(x)) terminando en punto y presionando la tecla de <return>, enseguida aparecerá F = para que escribas la segunda. Escribe p(y,y). y presiona <return>.
A continuación se desplegará la primera sustitución como se indica:

```
E = p(f(a),g(x)).

F = p(y,y).

θo=∅
D0={f(a),y}      θ1={y/f(a)}
Eθ1 = p(f(a),g(x))
Fθ1 = p(f(a),f(a))
```

Esto indica primero que la sustitución inicial es vacia, después se da la Diferencia D0 de las 2 expresiones que es el conjunto {f(a),y}, por lo que la primera sustitución θ1 es {y/f(a)}. Por último, se muestran las expresiones sustituidas Eθ1 y Fθ1.
El proceso continua y el sistema indica que se ha interrumpido porque g(x) y f(a) no unifican, por lo que se concluye que estas expresiones no son unificables.

En la parte inferior de la pantalla aparecen las siguientes opciones:

[u] unifica expresiones [r] regresa

para probar con otras dos expresiones o regresar al menú principal. Para seguir con el segundo ejemplo, selecciona la correspondiente a unificar expresiones. Al igual que se hizo con el primer ejemplo, escribe como expresión E=q(a,x,f(g(y))). y como expresión F=q(y,f(z),f(z)). A continuación el sistema mostrará las sustituciones:

```
E = q(a,x,f(g(y))).

F = q(y,f(z),f(z)).

θ0=∅
D0=(a,y)      θ1=(y/a)
Eθ1 = q(a,x,f(g(a)))
Fθ1 = q(a,f(z),f(z))


D1=(x,f(z))      θ2=(x/f(z))
Eθ2 = q(a,f(z),f(g(a)))
Fθ2 = q(a,f(z),f(z))


D2=(g(a),z)      θ3=(z/g(a))
Eθ3 = q(a,f(g(a)),f(g(a)))
Fθ3 = q(a,f(g(a)),f(g(a)))
```

Para unificar E y F se aplica la sustitución θ1=(y/a) a E y F obteniendo Eθ1 y Fθ2, a continuación se aplica la sustitución θ2=(x/f(z) y por último la sustitución θ3 para obtener como resultado las dos expresiones unificadas:

$$q(a,f(g(a)),f(g(a))) \quad y \quad q(a,f(g(a)),f(g(a)))$$

con lo que termina el proceso y $\theta = \theta_1\theta_2\theta_3 = (y/a,x/f(g(a)),z/g(a))$. A manera de ejercicio verifica con este programa si los siguientes pares de expresiones son o no unificables.

```
E = p(x,y)&q(x,h(z)).
F = p(y,x)&q(y,h(u)).

E = p(f(x),a).
F = p(y,f(u)).

E = p(y,f(u)).
F = p(v,x).
```

Una vez hechos estos ejercicios, selecciona la opción para regresar al Menú Principal.

4.2.3 Modulo Inferencias

El Módulo Inferencias es la instrumentación de Método de Arboles de Verdad. (2.4.2)
Se dan como entrada el conjunto de premisas y la conclusión, el sistema las transforma y permite al usuario construir tantos árboles como quiera a partir de éstas, indicando si el árbol se cerró o no, o lo que es lo mismo, si las inferencias son válidas. Con el mismo sistema se pueden también construir árboles para verificar si las fórmulas son tautologías.

Utilizaremos este módulo para la siguiente inferencia:

```
a -> ¬b
¬c -> a
-------
b -> c
```

(Por razones de representación interna de la máquina se utilizarán letras minúsculas para las letras sentenciales).

Desde el Menú principal del sistema de lógica II se accesa esta opción. Selecciónala y enseguida se desplegará la siguiente pantalla:

```
F5 : v    F6 : &    F7 : ¬    F8 : ->    F9 : <->

premisa1 :
```

Escribe cada Premisa, terminando en <return>

Para escribir la Conclusión primero presiona <return>

En la parte superior de la pantalla, aparece la indicación de como están programadas las teclas de función para los operadores lógicos.

A continuación, el sistema espera a que escribas la primera premisa. Escribe **a -> ¬b** seguido de <return> (no olvides separar con espacios en blanco entre los operadores -> y <->). Enseguida, el sistema esperará a que escribas la segunda premisa. Escribe **¬c -> a** seguido de <return>. Por ultimo, después del mensaje **premisa3:** presiona <return> para que aparezca una línea divisoria y el mensaje **Conclusión:** . Escribe **b -> c** terminando en <return>. Para este ejemplo debe quedar como resultado la siguiente pantalla:

```
premisa1  :  a -> ¬b
premisa2  :  ¬c -> a
             ----------
Conclusión  :  b -> c
```

```
[a] Transformación Premisas        [b] Hacer construcción árbol
[d] Archivo Arboles                [r] Regresa
```

En la parte inferior están las opciones a escoger. Con la opción [a] se consultan la transformacion de las premisas, esto es para ver cómo el sistema transformó las premisas originales.
Con la opción [b] se contruyen los árboles de verdad referentes a esta inferencia.
Con la opción [d] se consultan los árboles existentes en el archivo.
Por último, la opción [r] es para regresar al Menú Principal.

Transformación de Premisas

Para consultar las transformaciones de las premisas, selecciona la opción [a]. A continuación se desplegarán las premisas originales. Para consultar alguna en particular, escribe su número correspondiente o la letra c para consultar la transformación de la conclusión. Veamos en que se transformó la premisa 1. Al escribir 1 se desplegará:

a -> ¬ b se transforma en ¬ a v ¬ b

que es la transformación correspondiente por la regla de inferencia de la implicación. Oprime cualquier tecla para regresar y consulta las transformaciones de la premisa 2 y de la conclusión. Estas son:

¬ c -> a se transforma en ¬ ¬ c v a
¬ ¬ c v a se transforma en c v a

y

¬ (b -> c) se transforma en b & ¬ c

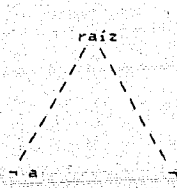
Para construir el árbol se trabaja con las premisas transformadas a conjunciones o disyunciones. En esta opción, se consultan las transformaciones. Las premisas que al transformarlas resultan ser conjunciones (de átomos o negaciones de átomos), dan lugar a nuevas premisas y se presentan como premisas separadas.
Ahora selecciona la opción de regreso para seguir adelante.

Construcción del Arbol

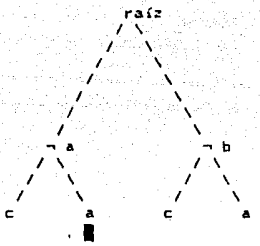
Para construir el árbol a partir de las premisas transformadas, selecciona la opción [b]. A continuación se desplegarán en la parte inferior de la pantalla las premisas transformadas. En este caso están numeradas del 1 al 4:

1) ¬ a v ¬ b 2) c v a 3) b 4) ¬ c

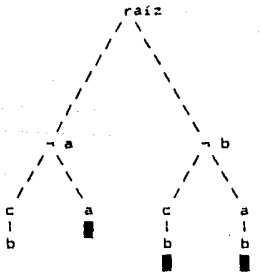
Esta numeración nos permite referirnos a ellas en el proceso de construcción del árbol. Empecemos la contrucción por la primera. Escribe 1 y a continuación se desplegará el siguiente árbol:



para continuar la construcción, selecciona la premisa 2 escribiendo 2. Esta segunda premisa se agregará al árbol y se obtendrá como resultado el siguiente:

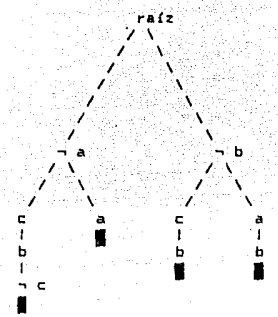


Como se ilustra, en este paso ya se cerró una rama, lo cual se indica con el símbolo ■. A continuación, selecciona la premisa 3 para obtener el siguiente árbol:



En este paso ya se han cerrado 3 ramas quedando como unica posible para seguir ramificándose la rama izquierda. Por último, selecciona la premisa 4 y como resultado final se despliega el siguiente árbol:

Arbol Cerrado

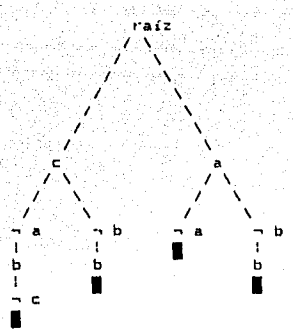


Como se indica, este árbol es cerrado, por lo que concluimos que la inferencia es válida.
A continuación trabajaremos con las opciones indicadas en la parte superior derecha de la pantalla.

Desde la construcción de arboles se pueden consultar las premisas originales a través de la opción **[p] premisas**. Selecciónala para verlas. Al regresar, verás que ya no aparece el árbol que construimos. Para desplegarlo de nuevo, selecciona la opción **[a] árbol** y verás que se pinta otra vez.

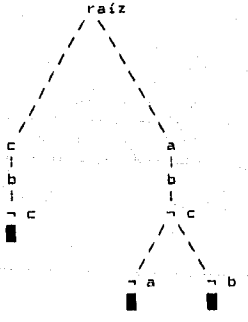
La siguiente opcion: **[g] guarda árbol** es para guardar un árbol en un archivo de árboles. Selecciónala y el sistema te pedirá le des el nombre bajo el que quieres guardar el árbol. Escribe **arbol1**, terminando en <return>. En este momento, el sistema pasó el árbol al archivo y puedes entonces construir otro.
En la parte de teoría (2.4.2) se mostraron para este ejemplo construcciones de árboles tomando en distinto orden las premisas, veamoslas en el sistema.
Construyo ahora otro árbol tomado las premisas en el siguiente orden: 2,1,3,4. Como resultado se desplegará el siguiente árbol:

Arbol Cerrado



El dibujo del árbol varió al anterior, aunque resulta también ser un árbol cerrado. A continuacion guarda este árbol bajo el nombre de **arbol2**. Ahora construye un tercer árbol con las premisas en el orden: 2,3,4,1. De lo que resultará:

Arbol Cerrado



Guarda este árbol bajo el nombre de **arbol3**. Una vez guardados en el archivo estos tres árboles, selecciona la opcion para regresar al menú.

Archivo Arboles

Para consultar los árboles existentes, selecciona la opción [d]. A continuación se desplegarán los nombres de los arboles que guardaste:

1) arbol1 2) arbol2 3) arbol3

La seleción de 1, 2 o 3 ocasionará el despliege del árbol que guardaste bajo el nombre indicado. En la parte inferior de la pantalla se muestran las premisas, y en la superior, el orden en que se seleccionaron y la profundidad que alcanzó el árbol presentado.

Con esta opción, puedes consultar cada uno de los árboles que hayas guardado y compararlos entre si. Fijándose por ejemplo, en que tipos de premisas abren mas un árbol, o en las que aumentan el nivel de profundidad.

En caso de querer remover un árbol del archivo, está disponible la opción [b] **borra árbol**. El sistema lo borra pidiendo el nombre que se le dio.

Una vez hecho este recorrido de las opciones a través de un ejemplo, regresa al menú principal (con las opciones de regreso) y prueba el sistema para las siguientes inferencias:

a) a
   a -> b
   ----
   b

b) a -> b
   b -> c
   ----
   a -> c

c) (a -> b) -> c

   ¬c -> a

d) a <-> b
   ----
   a -> b

Tautologías

Para verificar si una fórmula es o no una tautología, al entrar al módulo de Inferencias, escribe la fórmula como conclusión. Esto es, en respuesta al mensaje **Premisa1:** presiona <return> y escribe la fórmula. Después utiliza las opciones de igual forma que se hizo en la parte de Inferencias.
Como ejercicio, prueba con el sistema sin las siguientes formulas son o no tautologías:

a) a -> a

b) a -> (b -> a)

c) a -> (¬a -> a)

d) (a -> ¬a) -> a

Una vez probadas esta fórmulas, selecciona la opción para regresar al menú principal.

PARTE III

INSTRUMENTACION DE LOS SISTEMAS

5.1 GENERALIDADES

5.1.1 Lenguaje de Programación

La instrumentación se hizo en el lenguaje de programación PROLOG, en su implementación de Arity/Prolog versión 4.0, para microcomputadoras tipo PC.
Prolog es un lenguaje de programacion útil en la solución de problemas que comprenden objetos y relaciones entre ellos. PROLOG quiere decir **PROgramación en LOGica** - una idea que tuvo su origen al principio de la década de los 70's de usar la lógica como lenguaje de programación. Prolog puede verse como un subconjunto de la lógica de primer orden. Su sintaxis es la de fórmulas de la lógica de predicados de 1_{er} orden escritas en forma de **claúsulas de Horn** (claúsulas que tienen a lo más una literal no negada). Un programa en Prolog se define como un conjunto finito y ordenado de claúsulas de Horn representando hechos y reglas. El problema a resolver se plantea como una meta. Prolog utiliza un méetodo de refutación para demostrar que la negación de la meta, se deduce del conjunto de claúsulas. Este procedimiento está basado en el Principio de Resolución para demostración automática de teoremas introducido por Robinson (1965).

A diferencia de los lenguajes de programación imperativos, donde los elementos de un programa son instrucciones a ejecutar, Prolog es un lenguaje declarativo que describe la estructura lógica del programa. Sus estructuras de datos son las listas y su estructura de control fundamental es el retroceso (backtracking), lo que lo hace un lenguaje inherentemente recursivo.

Dadas las caracteristicas de este lenguaje, la instrumentación de los algoritmos que conforman estos sistemas de lógica se hizo de manera casi directa.

5.1.2 Diseño y Organización de Archivos

Los sistemas están diseñados con base en Menús. Cada Módulo es ejecutado mediante un predicado principal; que pinta el menú correspondiente. A partir de la opción que elija el usuario, llama a la ejecucion del sub-módulo seleccionado, mediante otro predicado.
Cada sub-módulo está en un archivo separado y el conjunto de todos ellos es leido al consultar el archivo principal de cada sistema (Lógica1.ari o Logica2.ari).
Cada sistema tiene un archivo de utilerías donde estan definidos los predicados que comparten todos los programas; estos son los predicados de manipulación de listas (agrega, borra, miembro, concatena, invierte, sustituye, etc.), los predicados para pintar los menús y los de pantalla (limpiar lineas, pintar líneas en otro tono, mensajes para pasar a otra pantalla, etc.).
Además, cada módulo tiene un archivo donde están definidos todos los mensajes de error y otro donde se guardan los predicados referentes a los datos que son modificados (como lo son: nuevos símbolos del alfabeto, terminos, fórmulas, ...).

Esta organización modular se hizo para dar la posibilidad de agregar en un futuro nuevas opciones o para hacer versiones de los sistemas que solo manejen algunos módulos.

5.2 SISTEMAS

A continuación se presenta una explicación de la instrumentación de cada sistema, exponiendo con mas detalle aquellos procesos o algoritmos que son los principales en cada módulo.

5.2.1 Sistema de Lógica I

Alfabeto

+ representación y manipulación de los símbolos

Los símbolos del alfabeto están representados como cadenas de caracteres, cada categoría definida mediante un predicado que tiene como argumento una lista. Así por ejemplo, se define a las variables como: **variables([u,w,x,y,z]).**

Los símbolos que son susceptibles de ser modificados durante la ejecución, se guardan en un archivo aparte; de esta forma, las funciones de agregar y borrar se hacen a través de los predicados assert y retract.

Términos

+ representación de las expresiones

Toda expresión se lee como cadena de caracteres (c_1,...,c_n) y se transforma en una lista ($[a_1,...a_n]$). Para poder distinguir en la expresión aquellos símbolos que se componen de mas de un caracter (por ejemplo, x1 se compone de dos caracteres), se aplica un predicado que recibe como entrada la lista ($[a_1,...a_n]$) y entrega como resultado la expresión en forma de lista donde cada símbolo utiliza una posición de la lista. Esta ultima representación es la que se manipula.

ejemplo:

```
Expresión original (Cadena) : $+(0,x1)$
list_text(Lista,Cadena)     : [43,40,48,44,120,49,41]
Expresión resultado (Exp)   : [+,'(',0,x1,')']
```

En la implementación de Prolog que se utilizó, hay un predicado que transforma una cadena en término (string_term(Cadena,Termino)), pero éste no se puede utilizar ya que en este módulo, al agregar o verificar una expresión, ésta no es necesariamente un término.

+ Algoritmo de Verificacion

Para verificar si una expresión es o no un término, se instrumenta de manera casi directa la definición de término. Los pasos del algoritmo son los siguientes:

1.- Si la expresión es una variable o una constante del alfabeto, el algoritmo termina indicando que es un término. En caso contrario, si es un solo símbolo, indica que no es término. Si es un conjunto de símbolos pasa al paso 2.

2.- Parentización.
predicado: checa_par(Exp,Valor).

Se verifica la parentización a través de este predicado que recibe como entrada la lista Exp y entrega como resultado en Valor una constante del conjunto: (vacio, exceso_cerrados, exceso_abiertos). La verificación se controla con una pila que inicialmente es vacía (pila([])). El proceso consiste en recorrer la lista Exp e ir metiendo a la pila los paréntesis abiertos y cada vez que aparezca un paréntesis cerrado, sacar un elemento de la pila. El proceso llega a su fin al terminar de recorrer la lista o cuando se intenta sacar un elemento y la pila es vacía. En el primer caso, si la pila es vacía, la expresión está bien parentizada y Valor regresa el valor vacío, si la pila no es vacía, regresa el valor exceso_abiertos. En el segundo caso, Valor regresa el valor exceso_cerrados.

3.- Símbolos en el Alfabeto
predicado: en_alfa(Exp,Val).

Se verifica si todos los símbolos están en el alfabeto recorriendo la lista Exp y viendo si sus componentes son variables, constantes, símbolos de puntuación u operadores. Si se encuentra un símbolo que no sea ninguno de los anteriores, se interrumpe el proceso y el valor que se regresa en Val es el símbolo desconocido. Si se termina de recorrer la lista exitosamente, se regresa en Val la constante v.

4.- Construccion Sintactica.
predicado: sintax_exp(Exp,Vals,Salida).

Se verifica si la expresión está bien construida recorriendo la lista Exp y verificando que tipo de símbolo debe venir después del leído. El primer símbolo debe ser un operador seguido de un parentesis. Si el símbolo leído es una variable o una constante, debe seguir una coma o un paréntesis que cierra. Si el símbolo es un operador, se llama recursivamente al predicado. El valor que se regresa en Vals es la constante v o f, indicando respectivamente si la construcción fue exitosa o fallida.
El último argumento (Salida), se utiliza para controlar desde la llamada al predicado si salen en pantalla los mensajes de error en la construcción. Lo anterior se hace porque este programa se utiliza tanto en la opción de verificar como en la de agregar un término (para proteger al sistema de que no agregue expresiones que no lo sean). Cuando en la llamada está instanciado a salida, presenta los mensajes.

5.- Grado de los Operadores
predicado: functor(Term,Oper,Aridad).

Se transforma la expresión a termino (Term) y se verifica si el número de argumentos (Aridad) es igual al grado con el que se definió el operador (Oper). Se hace este paso para todos los operadores que aparecen en la expresión.

Si el proceso que involucra a todos los pasos anteriores termina exitosamente, la expresión es término. Si alguno de los pasos anteriores falla, el proceso se detiene. Así por ejemplo, si una expresión esta mal parentizada, el algoritmo termina y ya no ejecuta los siguientes pasos.

Fórmulas

+ Teclas de Funciones

Para hacer accesible al usuario la escritura de los conectivos y cuantificadores, están programadas las teclas de funciones como sigue:

F3 : ∀ F4 : ∃ F5 : v F6 : & F7 : ¬ F8 : -> F9 : <->

Se leen las fórmulas caracter por caracter mediante el predicado de prolog **keyb(Ascii,Scan)** y se va transformando cada caracter leido (tecla presionada) en un caracter de tipo cadena.
Por ejemplo, si el predicado regresa los valores keyb(0,61), esto quiere decir que el usuario presionó la tecla de función F3. Esta lectura se transforma en la cadena $∀$. Despues de esta transformación de los caracteres leidos, la fórmula queda representada como una cadena.

+ representación de conectivos y cuantificadores

Los conectivos y cuantificadores se definen como operadores:

```
:- op(100,fy,#).        % negacion
:- op(110,fy,∀).        % cuantificador universal
:- op(110,fy,∃).        % cuantificador existencial
:- op(120,xfy,&).       % conjunción
:- op(120,xfy,v).       % disyunción
:- op(130,xfy,->).      % implicación
:- op(130,xfy,<->).     % doble implicación
```

Para cada operador lógico se definen su precedencia, su sintaxis y el símbolo que lo representa. El símbolo que representa a la negación #, no es el mismo que aparece en el sistema ¬. La implementación de Prolog no permitio usarlo como operador por lo que la representación interna se hace con el símbolo # y para despliege aparece ¬.
Con estas definiciones, las fórmulas se manipulan como functores lo que hace muy sencilla su verificacion.

+ Algoritmo de Verificación

Para verificar si una expresión es o no una fórmula, se instrumenta de manera casi directa la definición de fórmula. Los pasos del algoritmo son los siguientes:

1.- Si la expresión no contiene cuantificadores o conectivos (caso en que si es fórmula, es atómica), se verifica si es fórmula en forma análoga al verificador de términos, con la única diferencia que el primer símbolo debe ser un predicado. Esto es, se verifica parentización, pertenencia al alfabeto y construcción sintactica. El paso 5 (Grado de Operadores) no se incluye ya que en este módulo se le da mas libertad al usuario para definir sus fórmulas.

2.- Si la expresión tiene cuantificadores o conectivos, se transforma de cadena a término (en el sentido de Prolog) y se verifica si cada componente es una fórmula como en el paso 1. Si por ejemplo, la fórmula es del tipo A -> B, se verifica mediante la siguiente clausula del predicado es_formula:

```
es_formula(X -> Y) :- es_formula(X), es_formula(Y).
```

+ Representación y Manipulación de Fórmulas

Las fórmulas se manipulan como cadenas o como functores. Para su verificación conviene la representación de functores ya que, gracias a que los conectivos y cuantificadores están definidos como operadores, la verificación de la sintaxis se hace directa en Prolog. Para leer la fórmula, conviene tenerla como cadena ya que, como se va leyendo caracter por caracter y haciendo su transformación, resulta mas facil entregar una cadena. Además, para la opción que construye fórmulas a partir de otras, la construcción de conjunciones, disyunciones o implicaciones se hace directamente concatenando cadenas.

5.2.2 Sistema de Lógica II

Alcance

+ representación y manipulación de las fórmulas

Las fórmulas en este caso se representan como cadenas de caracteres. La fórmula puede ser escrita por el usuario o leída de la base de datos de fórmulas. En cualquiera de los dos casos, la fórmula se transforma a una lista especial para poder localizar las posiciones de las variables y cuantificadores.

ejemplo:

```
Expresión original (SFormu) : ∀x1R(x1,x2)->R(x1,x2)$

list_text(Lista,SFormu)     : [227,120,49,82,40,120,49,44,120,50,
                               41,45,62,82,41,120,49,44,120,50,41]

Expresión resultado (Form)  :

['∀',x1,'R','(',x1,',x2,')','->','R','(',x1,',',x2,')']
```

+ representación de variables y cuantificadores

Una vez representada la fórmula como una lista (**Form**), se identifican en ella las variables y los cuantificadores y se crean las listas de ocurrencias y de alcances. En la lista de ocurrencias cada elemento es una lista con dos elementos: la variable y la posición de su ocurrencia en la fórmula ($[[Var_1,Pos_1],\ldots,[Var_n,Pos_n]]$). En la lista de alcances, cada elemento es una lista con cuatro elementos: el cuantificador, la variable que cuantifica y las posiciones de su alcance, el inicial y el final ($[[Cuant_1,Var_1,Posi_1,Posf_1],\ldots,[Cuant_n,Var_n,Posi_n,Posf_n]]$).
Tomando como ejemplo la fórmula ∀x1R(x1,x2)->R(x1,x2), se crean las listas:

```
Ocurrencias = [[x1,1],[x1,4],[x2,6],[x1,11],[x2,13]]

Alcances    = [['∀',x1,1,7]]
```

+ ocurrencias de variables

Con la representación anterior, para saber si la ocurrencia de una variable es libre o acotada, se aplica un predicado: **ocur(Ocurrencia,Alcances,Resultado)**, donde Ocurrencia es la lista de la variable a analizar ([x1,1]), Alcances es la lista de alcances ([['∀',x1,1,7]]) y en Resultado el predicado regresa los elementos de la lista de Alcances que acotan a esa variable, o bien, la lista vacía si ningún cuantificador acota a esa variable. La definición de alcance está dada por posiciones de la siguiente forma:

La ocurrencia de una variable [Variable,Posicion] es acotada si existe un elemento en la lista de alcances [$Cuant_n$,Var_n,$Posi_n$,$Posf_n$] tal que Variable = Var_n y $Posi_n$ <= Posicion <= $Posf_n$

Siguiendo el ejemplo, el resultado de las llamadas al predicado ocur son:

```
ocur([x1,1],[['∀',x1,1,7]],['∀',x1,1,7])

ocur([x1,4],[['∀',x1,1,7]],['∀',x1,1,7])

ocur([x2,6],[['∀',x1,1,7]],[])

ocur([x1,11],[['∀',x1,1,7]],[])

ocur([x2,13],[['∀',x1,1,7]],[])
```

Con lo que se obtenga en el argumento Resultado, si es la lista de todos los cuantificadores que acotan a una variable, se aplica un predicado **marca_ocur([Var,Pos],Resultado,Linea)** para pintar en tono más fuerte la variable y el alcance del cuantificador que la acota.

Si al llamar al predicado **ocur**, el argumento Alcances es la lista vacía, esto significa que es una fórmula sin cuantificadores por lo que se deduce que todas las ocurrencias de variables que aparezcan, son libres.

Unificación

+ representación de las expresiones

Las expresiones que son leidas para aplicarles el algoritmo de unificacion, se leen y se representan directamente como terminos. Con esta representación, se aprovechan los predicados ya implementados en Prolog de manipulación de terminos como los son:

```
functor(Termino,Nombre,Aridad)  % predicado para obtener en Nombre el
                                  nombre del Termino y en Aridad el
                                  numero de argumentos.
arg(N,Termino,ArgumentoN)       % predicado para obtener en ArgumentoN
                                  el enésimo (N) argumento de Término.
```

+ Algoritmo de Unificación

Para verificar si dos expresiones E y F son unificables, se instrumenta de manera casi directa el algoritmo de unificación. Los pasos del algoritmo son los siguientes:

1.- Si las expresiones E y F son iguales, el algoritmo termina indicando que son unificables. En caso contrario, pasa al paso 2.

2.- Unificación.
predicado: unify(Term1,Term2,NuevoT1,NuevoT2).

En los primeros dos argumentos: Term1 y Term2 están las expresiones a unificar y el predicado debe regresar en NuevoT1 y NuevoT2 el resultado de haberlas unificado.

El predicado unify está definido mediante cinco cláusulas cubriendo los siguientes casos:

a) Cuando Term1 y Term2 son variables
b) Cuando Term1 es variable y Term2 no lo es
c) Cuando Term2 es variable y Term1 no lo es
d) Cuando Term1 y Term2 no son variables
e) Cuando Term1 y Term2 son términos

En los primeros cuatro casos se hace una sustitución, siempre y cuando la variable a sustituir no ocurra en lo que va a sustituirla. Por ejemplo, en el caso a) se sustituyen todas las ocurrencias de Term2 por Term1. La llamada a este predicado es recursiva.
Para el caso e) (cuando las dos expresiones son términos), se llama a un predicado **term_unify**, que verifica primero que los dos términos tengan el mismo operador y mismo número de argumentos para seguir con la llamada al predicado **unify_args** que intenta unificar argumento por argumento estos dos términos. Estas unificaciones se realizan llamando recursivamente al predicado principal **unify**. (paso 1)

El algoritmo termina cuando las dos expresiones a unificar son iguales, en cuyo caso son unificables o cuando durante el proceso de unificación se encontró que argumentos no son unificables.

Durante todo el proceso de este algoritmo se van pintando las sustituciones que se van realizando y como van quedando las expresiones.

Inferencias

+ <u>representación y tranformación de las premises</u>

Las premisas se leen como cadenas de caracteres utilizando las teclas de funciones para los conectivos lógicos como en el caso de las fórmulas. Las premisas leidas y la conclusión se van guardando en una lista que es el argumento del predicado **premisas([a->b,b->c,a->c])**. A cada premisa se le aplica el predicado **procesa(Premisa)** para transformarla en conjunción o disyunción. En esta transformación, se va creando la lista de premisas transformadas: **claus([#avb,#bvc,a,#b])**. Con estas dos listas, se tienen siempre presentes las premisas originales y sus transformaciones.

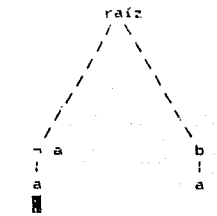
+ representación y manejo de los árboles

Inicialmente se tiene el árbol vacío representando en el predicado **arbol(nil)**. En cuanto se comienza la construcción del árbol, esto es, cuando se añade la primera premisa al árbol vacío, la representación del árbol se sustituye por el predicado:

arbol(Subarbol_Izquierdo,Raiz,Subarbol_Derecho).

Todo árbol diferente de vacío, es un árbol binario donde Raiz es siempre una letra sentencial, la negación de ésta o la constante *raiz* que representa a la raíz principal del árbol. Los sub-arboles izquierdo y derecho son: otro predicado árbol con la misma representación, la constante **nil** o la constante **hoja**. La constante nil, indica que es el final de una de las ramas del árbol de donde se pueden seguir añadiendo nuevas premisas. La constante hoja, indica alguna de las siguientes condiciones: 1) que es el final de una rama cerrada del árbol, ó 2) que es el subárbol izquierdo de una premisa de donde cuelga una premisa que no es del tipo a v b. En estos dos casos, ya no se pueden seguir añadiendo nuevas premisas de estos nodos llamados hojas.

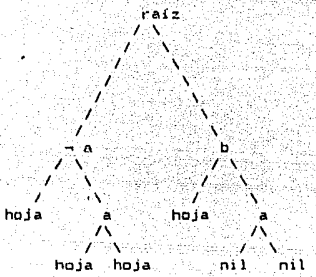
Por ejemplo, el árbol que en pantalla se ve como:



se representa en el predicado :

```
arbol(arbol(hoja,¬a,arbol(hoja,a,hoja)),
      raiz,
      arbol(hoja,b,arbol(nil,a,nil))
     ).
```

esta representación se ilustra mejor en el siguiente árbol:



De esta forma, cuando un árbol tiene como subárboles izquierdo y derecho la constante hoja, se sabe que es una rama cerrada. Cuando solamente el subárbol izquierdo es hoja, significa que se añadió una letra sentencial. Para añadir nuevas premisas, se modifica el predicado árbol solamente en el subárbol que tiene como subárboles a las constantes nil. Las modificaciones al árbol se hacen borrando (con retract) el árbol actual y añadiendo el árbol actualizado (con assert).

+ <u>despliegue de árboles en pantalla</u>

Para pintar graficamente un árbol en la pantalla, se pasa a una lista la estructura del árbol y se pinta en el siguiente orden:

- pinta la raíz del árbol
- pinta el subárbol izquierdo en el mismo orden
- pinta el subárbol derecho en el mismo orden

En la transformación a lista de la estructura del árbol, se calcula su nivel de profundidad, lo que sirve para que al construir su representación gráfica, ésta quepa en la pantalla.

+ Algoritmo de Arboles de Verdad

Para verificar si de un conjunto de premisas $p_1,\ldots,p_n$ se deduce una conclusión c, se instrumenta de manera casi directa el método de árboles de verdad. Los pasos del algoritmo son los siguientes:

1.- entrada de premisas.
Se leen las premisas y la conclusion. Se transforman las premisas y la negación de la conclusion a disyunciones o conjunciones. Las premisas que resultan ser conjunciones, se parten en premisas mas simples, una por cada conjugando.

2.- selección de premisa.
Se presenta la lista de las premisas transformadas, para que el usuario seleccione la que quiere agregar al árbol.

3.- agregar una premisa.
La premisa seleccionada puede ser de dos formas: una disyunción del tipo avb o una letra sentencial (o su negación). Si es del tipo disyunción, se sustituye al subárbol izquierdo la letra a y al subárbol derecho la letra b. Si es una letra sentencial, se sustituye tanto en el subarbol izquierdo como en el derecho.

4.- verificar si se ha cerrado el arbol.
Para cada letra sentencial (o su negación), se recorre hacia arriba la trayectoria a que pertenece, revisando si en ese camino se encuentra su negación, en cuyo caso se cierra la rama, cambiando las dos ocurrencias correspondientes de las constante nil por la constante hoja.

5.- Se presenta en pantalla el árbol resultado.

6.- Si faltan premisas por agregar y el arbol NO es cerrado en todas sus ramas, se va al paso 2 para seleccionar la siguiente premisa.
Si ya se añadieron todas las premisas existentes, se verifica si el árbol es o no cerrado en todas sus ramas y se indica el resultado.

CONCLUSIONES

Antes de concluir, haremos un breve resumen de lo que ofrece el sistema automatizado que se presenta en esta tesis, haciendo un análisis de sus alcances y limitaciones con respecto a dos puntos principales: su utilidad como sistema de apoyo para la enseñanza de la lógica y su potencial como programa de computación.

A continuación se presenta un cuadro donde se muestran, en la columna izquierda, los conceptos para los que se instrumento un módulo en el sistema, indicando en la columna derecha las funciones que se realizan en cada uno de ellos.

Conceptos	Apoyo del Sistema
Alfabeto	- En el Módulo Alfabeto se presenta un alfabeto inicial, dándole la posibilidad al usuario de manipular los símbolos, esto es, de agregar nuevos símbolos o borrar existentes.
Términos	- En el Módulo Terminos, se le da posibilidad al usuario de agregar o borrar expresiones que sean términos y de verificar expresiones indicando paso a paso si la expresión que se da como entrada, es o no un término.
Fórmulas	- En el Módulo Fórmulas, se le da posibilidad al usuario de agregar o borrar fórmulas del sistema, así como de construir nuevas fórmulas a partir de las existentes. Además, tiene un programa que verifica si una expresión es o no una Fórmula.
Variables y Cuantificadores	- En el Modulo Alcance, por un lado, a la entrada de una fórmula, el sistema presenta las presencias de las variables, indicando si son libres o acotadas. Por otro lado, tiene la opción de recibir como entrada, junto con la Fórmula, los tipos de presencias y verificar si se han indicado correctamente.

Sustituciones y Unificación	- En el Módulo Unificación, a la entrada de dos expresiones, el sistema va presentado las sustituciones para intentar unificarlas. Si no son unificables, se interrumpe el proceso indicando los argumentos que impidieron la unificación.
Inferencias	- En el Modulo Inferencias, a la entrada del conjunto de premisas y la conclusión, el sistema le da posibilidad al usuario de que verifique si la conclusión se deduce de las premisas, a través de la construcción del arbol de verdad. Cada arbol construido puede guardarse, para después hacer comparaciones de las estructuras de los arboles existentes. Ademas, el usuario puede consultar las transformaciones de las premisas originales a formas disyuntivas o conjuntivas.

Los tres primeros modulos, que son lo que conforman el sistema de lógica I, proporcionan de manera completa una automatización de todo lo referente a la sintaxis de los lenguajes de primer orden. Con estos módulos, se pueden hacer ejemplos para alfabetos en diferentes teorías, definiendo en el sistema los símbolos y construyendo terminos y fórmulas a partir de ellos. Algunas de las limitaciones que tiene el sistema en esta parte son, que está diseñado para manipular términos y fórmulas de no más de 80 caracteres (un renglón). No se pueden definir mas de 100 fórmulas, pues la numeracion es de dos digitos. Además, los programas que verifican, tanto términos como fórmulas, no dan la posibilidad de reeditar la expresion sobre la que se está trabajando, si el usuario quiere volver a verificarla haciendo algunos cambios, tiene que volver a escribirla. A pesar de estas restricciones, para efectos de utilizar esta parte como una introduccion a la lógica, se puede trabajar en el sistema en paralelo a una exposición teorica de la sintaxis de los lenguajes de primer orden probando suficientes ejemplos.

Con respecto a los otros módulos, en el referente a variables y cuantificadores (llamado en el sistema: Alcance), se instrumentó la definicion 2 (2.2.1), la cual presenta a las fórmulas como sucesiones de símbolos e introduce el concepto de presencia de una variable. Se escogió esta, por ser la que mejor se podía representar en la maquina y con la que se podia dar una presentación mas gráfica.

En un curso de lógica, una vez presentados los conceptos de ocurrencias libres y acotadas de las variables en las fórmulas y de alcances de cuantificadores, este módulo puede servir de apoyo en dos sentidos: por un lado, tomando como ejemplos la lista de fórmulas que ya vienen en el sistema o escribiendo otras, puede utilizarse la parte de despliegue automático para que el alumno vaya reconociendo el tipo de ocurrencia de cada variable en las fórmulas. Por otro lado, de manera interactiva, el alumno puede indicar el tipo de presencias de las variables y evaluar con el sistema si lo que escribió fue o no correcto.

Con respecto a las limitaciones del sistema en este módulo, para evitar que el programa creciera mucho y resultara muy lento, cuando el usuario escribe la fórmula a analizar, no se incluyó el verificador de fórmulas (para verificar si efectivamente la expresión que se da como entrada, es una fórmula). Esto ocasiona que el sistema no funcione efectivamente en el análisis de los alcances de cuantificadores cuando una fórmula esta mal construida, sobre todo cuando esta mal parentizada. Por esta razón, se da una lista de fórmulas para trabajar en este módulo, por si se quiere usar para usuarios inexpertos en la construcción correcta de fórmulas. Aunque de los errores de 'dedo' no se salva nadie.

Con respecto al módulo de unificación, se instrumento el algoritmo de unificación para dos expresiones. Este módulo puede utilizarse una vez que se hayan presentado los conceptos de sustitución, sustituciones compuestas y unificador mas general. Lo que más distingue a este módulo con respecto a los otros, es que no es un módulo interactivo, en el sentido de que solo va desplegando como va intentando unificar las dos expresiones, sin dar opción al usuario de proponer las sustituciones.

Por último, referente al módulo inferencias, que es quizá el más pretensioso y por tanto, el más limitado, podemos decir que es el módulo del sistema que requiere de un mayor soporte en la parte de teoría, el usuario debe conocer el concepto de inferencia y en particular, el método de arboles de verdad que es lo que esta instrumentado en este módulo. Es un modulo muy interactivo que permite presentar de una manera didáctica y grafica las pruebas de inferencias y tautologías. Con respecto a sus limitaciones, una de ellas es que las letras sentenciales de las premisas deben escribirse en minúsculas, (ya que Prolog interpreta las mayúsculas como variables), cuestión que puede ocasionar confusiones. La limitación mas evidente es que solo se pueden probar ejemplos con pocas premisas, debido a la limitación de espacio en la pantalla para desplegar el arbol correspondiente.

Una vez hecho un análisis de los módulos del sistema, a continuación presentaremos algunas ventajas y desventajas con respecto a la implementación de Prolog utilizada para esta instrumentación.

Se escogió Arity/Prolog versión 4.0 por ser una implementación que tiene intérprete y compilador, además de que tiene estructuras de control que no son propias de prolog puro como lo son el **ifthen** y el **case**, las cuales permiten no depender totalmente del backtracking. En cuanto a la experiencia de haber trabajado en Prolog para la automatización de este sistema, pueden señalarse fundamentalmente dos cuestiones. La primera es que el hecho de que la sintaxis de prolog y la del lenguaje de la lógica sean tan similares, permitió traducir las definiciones y algoritmos de manera casi directa, cuestión que resultó ser una gran ventaja. La segunda es que, sin embargo, el control general del programa, esto es, manejo de errores, mensajes y control de opciones escogidas para llamadas a predicados, resultó muy pesada y poco facil de programar ya que, a pesar de contar con las estructuras de control extra-prolog ya mencionadas, el hecho de que prolog sea inherentemente recursivo lo hace ineficiente y muy lento en ejecución en las máquinas convencionales. Esto se hace muy evidente sobre todo en el modulo de inferencias, donde el sistema se alenta muchísimo al ir construyendo nuevos arboles.

Para concluir, mencionaremos los aspectos más importantes que caracterizan al sistema automatizado así como la utilidad que tiene el haber instrumentado un sistema de apoyo a los cursos de lógica.

- La modularidad del sistema permite trabajar con los módulos por separado, así como integrar nuevos, sin romper la estructura general del programa.

- El sistema ofrece una notación muy parecida a la usual en el lenguaje lógico, lo que permite tener de fácil manejo los conectivos y cuantificadores y con esto, la presentación de las fórmulas se hace muy clara.

- El hecho de que muchos de los programas sean interactivos, permite que el aprendizaje sea didáctico y con participación del usuario.

- Los programas de verificación, proporcionan una opción para que el usuario visualice y corriga sus errores.

- los programas de despliegue, presentan paso a paso lo que van haciendo.

- En general el sistema, como apoyo a los cursos de lógica en su parte práctica, propone una forma de trabajo para que el alumno practique los conceptos aprendidos a través de ejemplos muy variados.

Por último, presenteramos los puntos más relevantes que quedaron por resolver con respecto a mejoras que se le pueden hacer al sistema e instrumentación de nuevos módulos:

1) Añadir un módulo que ejemplifique las tablas de verdad de los conectivos lógicos.

2) Añadir una opción interactiva en el modulo de unificación.

3) Ampliar el módulo de inferencias permitiendo trabajar con ejemplos tan largos como se quiera y, con respecto al despliegue de los arboles, cuando estos no quepan en la pantalla, permitir que el usuario pueda visualizar parcialmente el arbol. En la opción donde se pueden consultar los arbolés guardados, hacer un programa que permita desplegarlos todos juntos pintando solo su estructura en cuadros pequeños y así poder comparar sus estructuras teniéndolos todos en pantalla.

4) Añadir un módulo que permita hacer la construcción de las cerraduras universales y existenciales de fórmulas.

5) Añadir un módulo interactivo que permita aplicar sustituciones simples y compuestas a las fórmulas.

En particular, para lo presentado en los puntos 1 y 5 se comenzo la instrumentación, pero por no quedar totalmente terminada, no se incluyó en este trabajo.

Estos puntos que quedaron pendientes pueden ser motivo de nuevos trabajos que enriquezcan y complementen la tarea de enseñar lógica con apoyo en un sistema automatizado.

APENDICE

MANEJO DE ERRORES Y DAÑOS AL SISTEMA

Dado que los sistemas de lógica I y lógica II están orientados a la enseñanza, están diseñados para que puedan ser utilizados por cualquier usuario principiante en el uso de las computadoras. Los sistemas están protegidos para que en la mayoría de los casos, cuando el usuario comete algún error, se le indique a través de un mensaje y no se le permita hacer cosas inválidas. Sin embargo, existen daños al sistema que están fuera de toda protección (como cuando se va la luz) y que es importante guiar al usuario en que hacer en casos de este tipo.

A continuación expondremos primero las protecciones más importantes con las que cuentan los sistemas así como las protecciones con las que no cuenta para finalizar con la explicación de los errores y daños al sistema más comunes con los que se puede enfrentar el usuario.

Protecciones del sistema

+ Selección de opciones en los menus

Al escoger una opción de cualquier menú, el sistema acepta solamente las letras indicadas a seleccionar, si se escribe otra, no se puede pasar del menú y el sistema espera otra opción, hasta recibir una correcta.

+ Como el sistema guarda información

En los módulos donde el usuario manipula datos, esto es, que agrega o borra, el sistema los actualiza cuando se selecciona la opción de regreso al menú. Esto asegura que si el sistema falla, la información que se modificó no se altere.

+ Manipulación de datos

En los módulos donde el usuario agrega o borra datos existentes, el sistema está protegido para no permitir agregar datos que ya existen o borrar datos inexistentes. Si el usuario comete un error de este tipo, el sistema despliega un mensaje indicando la falla y le permite volver a intentar. En el módulo fórmulas, el sistema no permite definir dos fórmulas con el mismo número de referencia o al construir nuevas fórmulas a partir de otras, verifica que existan las usadas en la construcción, si se intenta usar una que no exista, se despliega un mensaje y se permite volver a capturar un número correctamente. Par datos que se requiere sean numéricos (como el orden y grado en predicados y operadores), el sistema verifica que lo que el usuario escriba sea justamente un número, si no es así, lo indica con un mensaje y da la posibilidad de escribirlo correctamente.

Errores de los que no está protegido el sistema

+ Escritura de fórmulas en lógica I

Cuando el usuario quiere escribir una fórmula a través de la opción **agrega**, no funciona el verificador. Esto ocasiona que se pueda agregar cualquier cadena de caracteres como fórmula.

Recomendación:

Si no has dado <return>, te puedes regresar con la tecla de retroceso (backspace) y reeditarla.
Si ya diste <return>, dale un número de referencia y enseguida bórrala.

+ Escritura de fórmulas en el módulo alcance

Al escribir una fórmula en este módulo, tampoco funciona el verificador por lo que, si está mal escrita, no va a reconocer las variables y los programas no funcionarán.

Recomendación:

Selecciona la opción para regresar y ejecuta de nuevo el módulo.

+ Escritura de expresiones en el módulo unificación

En este módulo, se pide a través de un mensaje que escribas la expresión terminando en punto. Si la expresión esta mal parentizada, Prolog dará un error de uso de operadores o de puntuación inválida y se quedará el cursor tintileando.

Recomendación:

Lo único que se puede hacer para recuperar al sistema de este error es teclar ^c (la tecla de control y la letra c) y enseguida aparecerá la señal del intérprete de prolog: ?- . Esto significa que has interrupido la ejecución del sistema, por lo que tendrás que empezar de nuevo tecleando **logica2.** seguido de <return> para que aparezca el menú principal.

+ Escritura de premisas en el módulo inferencias

En este módulo, es importante separar por un espacio las letras sentenciales de los conectivos -> y <->. Si esto no se hace, el sistema interpreta esa premisa como la constante **vacío** y aunque el programa funciona con normalidad, esa constante se tiene que añadir al árbol como si fuera otra premisa.

Recomendación:

Interrumpe la ejecución del programa (con ^c) y vuelve a llamar al sistema tecleando logica2.

Daños al Sistema

+ Que hacer si se va la luz

Si el usuario está trabajando en el sistema y se interrumpe el suministro de energía electrica, lo único que se puede hacer es reinicializar el sistema. Entendiendo por esto, volver a encender la máquina (cuando haya luz de nuevo, por supuesto), entrar al intérprete de prolog y llamar al sistema correspondiente (logica1 ó logica2). Si se han modificado datos en el sistema (nuevos simbolos, términos o fórmulas), y se ha dado la opción de regreso, éstos no se borrarán del sistema. Si se está trabajando en alguno de los verificadores o en cualquier modulo de lógica II, se perderá la expresión que se está verificando, la fórmula o el conjunto de premisas, por lo que habrá que escribila(s) de nuevo.

+ Como interrumpir la ejecución del sistema y salir de el

Si en algun momento durante la sesión de trabajo, se comete un error de escritura de los que el sistema no cuenta con protección alguna o el sistema cae en un ciclo (loop), la manera de interrumpirlo es tecleando ^c (la tecla de control y la letra c), lo que ocasionará la interrupción del sistema y el despliege de la señal del interprete de Prolog: ?- . A lo que el usuario puede volver a ejecutar el sistema que interrumpió (logica1 ó logica2) o bien, puede salir del intérprete a nivel de sistema operativo con la siguiente instrucción:

?- **halt.** <return>

Con lo anterior, se asegura que ningún archivo quede abierto y con esto evitar daños el sistema.

+ Como NO salirse del sistema

Si desde el sistema o despues de haber interrumpido algún proceso, el usuario se sale a la brava, esto es apagando la máquina o tecleando ^alt-del para inicializarla, puede dañar el sistema, haciendo que algún módulo ya no funcione o se pierda información. Por esto, es caso de desesperación lo mejor es interrumpir el sistema y salirse de el como se indica en el punto anterior.

BIBLIOGRAFIA

[1] ARITY/PROLOG Reference and Technical Manual. Versión 4.0

[2] BRATKO, Ivan. Prolog Programming for Artificial Intelligence. Addison-Wesley, 1986

[3] BUNDY, Alan. The Computer Modelling of Mathematical Reasoning. London, Academic Press, Inc., 1983

[4] ENDERTON, Herbert. A Mathematical Introduction to Logic. Academic Press, 1972.

[5] ENNALS, R. Teaching Logic as a Computer Language in Schools. contenido en: Logic Programming and its Applications. Ablex Series in Artificial Intelligence. 1986.

[6] JEFFREY. Logic : It Scope and Limits. McGraw-Hill

[7] KOWALSKI. R. Logic as a Computer Language in Education, contenido en: Logic Programming. Academic Press, 1982.

[8] LLOYD, J.W. Foundations of Logic Programming. Springer-Verlag, 1984

[9] NILSON, Nils, J. Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, 1971

[10] STERLING L. and SHAPIRO E. The Art of Prolog. MIT Press, 1986

[11] TARSKI, Alfred. Lógica Simbólica. contenido en: SIGMA El Mundo de las Matemáticas. Tomo 5. Grijalbo. 1969

[12] WINOGRAD T. and FLORES F. Understanding Computers and Cognition. Addison Wesley, 1986.