





**CINESTAV-IPN**  
Biblioteca de Ingeniería Eléctrica



FB00000963

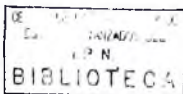
**CENTRO DE INVESTIGACION Y DE  
ESTUDIOS AVANZADOS DEL  
I. P. N.  
BIBLIOTECA  
INGENIERIA ELECTRICA**

CENTRO DE INVESTIGACION Y ESTUDIOS AVANZADOS  
DEL  
INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA  
SECCION DE COMPUTACION

CENTRO DE INVESTIGACION Y DE  
ESTUDIOS AVANZADOS DEL  
I. P. N.  
BIBLIOTECA  
INGENIERIA ELECTRICA

"APRENDIZAJE MEDIANTE INFERENCIA INDUCTIVA"



Tesis que presenta el M. en C. M. Ricardo Vázquez Alvidrez para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERIA ELECTRICA. Trabajo dirigido por el Dr. Zdenek Zdrahal H. y por el Dr. Guillermo Morales Luna.

México, D. F. Julio de 1988.

XM

CLASIF.	88.9
ADQUIS.	21-11057
FECHA	18-1-88
PROF.	DOJ

## C O N T E N I D O

PREFACIO .....	iii
RESUMEN .....	iv
INTRODUCCION .....	1
<b>I NOCIONES GENERALES</b>	
Aprendizaje .....	4
Aprendizaje y Reconocimiento de Patrones .....	6
Aprendizaje de Conceptos vs Constructor de Clasificación .....	7
Buscador de Definición Clasificadora .....	8
<b>II FUNDAMENTOS</b>	
Nociones de Lógica y Notación .....	10
Formalización del Objetivo del Programa .....	12
Fórmulas Tipo "M-de-N" .....	16
Funcionamiento Global del Programa .....	17
Una Base de Conocimientos Como Dato de Entrada .....	21
<b>III IMPLEMENTACION</b>	
Leyes de Unificación Asumidas .....	23
Uso de Generalizaciones en la Búsqueda de una Definición de Subclase .....	24
Leyes de Generalización .....	26
Inconveniencia de Buscar las Definiciones Más Restrictivas .....	27
Búsqueda en la Dirección de la Menor Generalización .....	30
Generalización por Predicados "Ligables" .....	33
Predicados y Funciones Especiales .....	33
CONCLUSIONES .....	36
BIBLIOGRAFIA .....	38
USO DEL FACORRIA, ILUSTRACIONES .....	39
INDICE DE ILUSTRACIONES DE "PROOF-EVER" .....	43
INDICE DE ILUSTRACIONES DE "F-OR-TODO" .....	5

## RESUMEN

Se propone y analiza un programa de cómputo para el aprendizaje automático de conceptos mediante la presentación de ejemplos. El programa busca una fórmula lógica de primer orden que defina al conjunto de ejemplos respecto a uno de contra-ejemplos. A diferencia de trabajos similares [Michalski 1980] aquí: los ejemplos son descritos con fórmulas convencionales de cálculo de predicados de primer orden, se admite como dato una base de conocimientos y se es capaz de tomar en cuenta la frecuencia con que la fórmula se cumple en los ejemplos y los contra ejemplos.



CENTRO DE INVESTIGACION Y DE  
ESTUDIOS AVANZADOS DEL  
I. P. N.  
BIBLIOTECA  
INGENIERIA ELECTRICA

## PREFACIO

Agradezco a todos los integrantes de la sección de computación del CIEA del IPN : profesores, compañeros y empleados por las facilidades otorgadas durante los estudios y durante la preparación de este trabajo.

De manera particular deseo expresar mi agradecimiento a: Dr. Zdenek Zdrahal H. por su valiosa dirección y a mi esposa Elia B. de Vázquez, por sus apreciables comentarios y apoyo en general.

Agradezco también a la UAM azc. por haberme brindado algunas facilidades y al Dr. Guillermo Morales L. por sus valiosos comentarios.

México D.F., Julio de 1983

CENTRO DE INVESTIGACION Y DE  
ESTUDIOS AVANZADOS DEL  
I. P. N.  
BIBLIOTECA  
INGENIERIA ELECTRICA



## INTRODUCCION

El interés por programas que realizan un aprendizaje en forma automática ha prevalecido desde la creación de las primeras computadoras electrónicas.

En la década de los 50 y 60's se estudió intensamente a los perceptrones. Este tipo de programa debía identificar correctamente a una imagen presentada como una matriz de bits, eligiendo una de entre varias posibles respuestas. Pero más aun, en sesiones de entrenamiento debía automáticamente aprender a identificar a estas imágenes ante la presentación repetida de ellas y de la respuesta correcta por parte del "entrenador".

Esta es la idea que se encuentra en el fondo de la frase "aprendizaje a partir de ejemplos".

En el caso de los perceptrones cada ejemplo era una matriz de bits.

Una aportación significativa a esta línea de experimentación e investigación es la de P.H. Winston [Barr, 1982]. El representa a cada ejemplo ya no como una matriz de bits, sino como una gráfica (en el sentido de un conjunto de nodos y de ligas entre ellos).

Además introduce la importante idea de obtener la definición buscada ( en su caso una cierta gráfica ) mediante operaciones de generalización y especialización desencadenados y dirigidos por la comparación entre la definición previa y la presentación de un nuevo ejemplo.

Por su parte Michalski [1980], contribuye con lo siguiente:

\* Propone un lenguaje al estilo de lógica de predicados (..aunque algo confuso [Barr 1982]).

\* Propone un conjunto de leyes de generalización que abarcan algunos conceptos sencillos de números y de conjuntos (..p.ej, las afirmaciones "a<30" y "b<20" se pueden generalizar a: "existe x tal que: x < 30").

La aceptación de la primer proposición nos da la ventaja de representar a los ejemplos de una manera más aproximada a una descripción en lenguaje común (.."lenguaje natural"..).

Es más sencillo para una persona sin conocimientos

de Teoría de Gráficas describir a cada ejemplo mediante un lenguaje de predicados que mediante una "gráfica".

La notación de Michalski no es sin embargo una notación estandar de lenguajes de predicados de primer orden. De hecho, él menciona [ Michalski , 1980] que su lenguaje no se puede reducir a uno de éstos. Esta opinión, por otro lado, no ha sido aceptada en la literatura del tema [ Barr , 1982 ]. Hay algunos otros puntos oscuros en su trabajo, lo cual por supuesto no le resta interés a sus contribuciones.

En este documento, siguiendo la línea de los trabajos que se han comentado, se propone incluir las siguientes ideas:

- \* El uso de bases de conocimiento de fondo.
- \* El uso de criterios estadísticos
- \* El uso de expresiones del tipo: " se cumplen m de las n siguientes propiedades:...".

Por otro lado, se usa además una notación más apegada a la común del cálculo de predicados de primer orden. Esto contribuye a clarificar a la discusión sobre el tema.

Se ha desarrollado un programa de cómputo que lleva a la práctica éstas ideas, capaz de correr en una computadora tipo "PC". Dicho programa realiza un aprendizaje automático ante la presentación de ejemplos.

En el primer capítulo discutimos la noción de " aprendizaje ", su asociación con el reconocimiento de patrones y además algunas cuestiones generales sobre un programa buscador de formas clasificadoras.

En el segundo introducimos algunos conceptos y notación de la teoría matemática de la Lógica y damos una descripción precisa de la función que realiza el programa que se ha desarrollado. Es decir , aclaramos en que sentido se ha realizado el objetivo de : " encontrar una definición de una clase de objetos ".

En el tercero se discuten algunos de los problemas específicos que se tuvieron que resolver para desarrollar el programa. Algunas de las soluciones consistieron en buscar una heurística adecuada.

Se incluye un apéndice: "Ilustraciones de Uso del Programa". En él se revisan los conceptos discutidos en el capítulo de implementación mediante varias ilustraciones del uso de las dos funciones principales del programa , "proof-ever" y "f-or-todo".

Se incluye una sección de referencias bibliográficas.

En la sección de "Conclusiones" se comenta sobre los logros obtenidos , las limitaciones observadas y se sugieren aplicaciones, líneas de investigación y mejoras a futuro .

## CAPITULO I

## N O C I O N E S   G E N E R A L E S

## QUE ES APRENDIZAJE

No siempre resulta necesario o conveniente dar una supuesta definición rigurosa de los conceptos manejados en una rama técnica, de tal forma que esa definición abarque todas las maneras de uso en el ámbito general.

A veces simplemente damos por sentado que el concepto se "entiende" intuitiva o empíricamente hablando. Como ejemplo podemos citar el caso del concepto de "tiempo", en Física.

Otras veces damos una definición más bien restringida, pero que técnicamente es suficientemente precisa como para sernos de utilidad.

Este es el caso de la noción de "aprendizaje".

Nuestro concepto restringido es el siguiente:

Supongamos que hemos definido un objetivo o meta experimentalmente comprobable para un programa de cómputo. Es decir, contamos con una manera experimental y objetiva de decidir si una respuesta del programa de cómputo, dada como consecuencia de la llegada de ciertos "datos" o "estímulos", satisface o no un cierto requerimiento-meta.

Además, supongamos que podemos definir bien a la máquina generadora de los estímulos o datos que entran al programa. Llamemos temporalmente a esta máquina "habitat".

También, supongamos que contamos con alguna medida objetiva del grado de competencia de diferentes programas ante habitat, objetivos y hardware constante.

Por ejemplo, la competencia puede medirse por el inverso del tiempo que tarda un programa en llevar a cabo cierta meta a partir de ciertos datos.

Bajo estas suposiciones, podemos definir "aprender" como la capacidad que tiene el programa de mejorar el promedio de su competencia ante habitat y metas constantes.

Una definición más cuantitativa y por ende más precisa, es definir el grado de aprendizaje como la rapidez del cambio de la competencia de un programa bajo habitat y metas constantes. La medida de tiempo implícita en esta definición es la inducida en forma natural por el habitat. Así por ejemplo, si el habitat es una máquina generadora de matrices de bits, el "tiempo" será el número de matrices presentadas hasta el momento al programa.

Una ventaja de esta definición es su objetividad, es decir, su capacidad de ser experimentalmente mensurable.

Para ilustrar nuestro concepto de aprendizaje, veamos el siguiente caso:

Un programa que simplemente almacena el último par asociativo llave-contenido que se le ha presentado (i.e. un manejador de archivos), no realiza un aprendizaje ante el objetivo:

"dar el último 'contenido' almacenado asociado a una 'llave' arbitrariamente elegida, de entre el conjunto de llaves que se le han presentado al programa "

, aquí suponemos que el habitat es una máquina que elige una llave al azar de entre un conjunto finito fijo de llaves, elige un "contenido" de longitud fija también al azar, y los presenta al "manejador de archivos" para dar de alta o cambiar el contenido de la llave recién generada.

La máquina almacenadora así constituida no realiza un aprendizaje (al menos no uno de signo positivo...), puesto que no mejora su competencia en cada corrida. El tiempo medio de "recuperación" del contenido aumenta con el tamaño del archivo, el cual está creciendo monótonamente por hipótesis. . Programas de este tipo realizan una memorización, pero no un aprendizaje.

En nuestro trabajo el programa realizará un aprendizaje, pues queremos que "clasifique" correctamente a un "objeto" arbitrario (de cierta clase fija) que se le presente y que su competencia para clasificarlo (medida por el porcentaje de aciertos y el inverso del tiempo que se tarda en hacer esta clasificación) mejore con el tiempo. Esta mejora se puede

alcanzar debido a que el programa tendrá una descripción clasificadora cada vez más perfeccionada, supuestamente.

La experiencia a lo largo de años de la comunidad de Inteligencia Artificial ha demostrado la necesidad de tener diferentes programas para diferentes objetivos de aprendizaje.

Nuestro programa no es una excepción y su ámbito de competencia está dirigido a situaciones en las que los objetos por clasificar pueden ser descritos por expresiones "razonablemente cortas" en un lenguaje de lógica de primer orden.

Por supuesto, el "ámbito" de aplicación del programa se ampliará a medida que el hardware y el software (algún intérprete o compilador Lisp) sobre los que se asienta mejoren.

## APRENDIZAJE Y RECONOCIMIENTO DE PATRONES

Como ejemplo de reconocimiento de patrones podemos indicar el reconocimiento de letras en el sentido de:

" Dada una imagen discretizada de una supuesta letra que ocupa una pantalla dada, decidir cual letra del alfabeto latino es ".

Así pues, reconocer es clasificar. En situaciones más complejas, podemos afirmar que al menos una función importante interna de un programa de reconocimiento de patrones es la de clasificar.

El diseñador del programa de reconocimiento puede encontrar sin la ayuda directa de un programa de cómputo las leyes de clasificación. Luego puede introducirlas y hacerlas operables en el programa de reconocimiento deseado.

Pero resulta interesante investigar también las posibles arquitecturas de programas que encuentren por sí mismos las leyes de clasificación.

Así planteado vemos que en cierto sentido, la salida de un programa que aprenda a clasificar objetos puede ser un programa de reconocimiento de patrones.

Al menos, el primero puede ayudar a un usuario en forma interactiva a encontrar al segundo.

Uno de los objetivos de este trabajo fue iniciar un primer prototipo de una serie de programas tendientes a ayudar a generar programas reconocedores de patrones, en una gama amplia de aplicaciones.

Pienso que una manera de abordar una gama amplia es diseñando programas que aprendan a clasificar a objetos de muy diversos tipos.

La entrada a estos programas no pueden ser matrices  $m \times n$  de unos y ceros. Tampoco pueden ser vectores de números. Esto restringiría demasiado (en la práctica, aunque no en principio) al conjunto de tipos de objetos que serían manipulables por el programa. De las pocas posibilidades que restan está la de describir a los objetos con un lenguaje formal pero muy flexible, como el de lógica de primer orden o al menos algo aproximado a ésto.

Con un lenguaje de predicados y usando sólo conjunciones de afirmaciones simples (conjunciones de fórmulas atómicas, ver "Nociones Fundamentales") podemos describir a objetos de una gama muy amplia, por ejemplo:

Letras discretizadas : indicando "como se dibujan" en un lenguaje tipo Logo.

Frijoles de soya : expresando localizaciones y formas relevantes en un lenguaje de tipo geométrico.

Pacientes clínicos : casi listando simplemente los síntomas e indicando fragmentos de historia clínica.

#### APRENDIZAJE DE CONCEPTOS VS CONSTRUCTOR DE CLASIFICACION

En este trabajo, consideramos equivalente el hecho de aprender un concepto al hecho de encontrar una definición que clasifique correctamente a objetos ejemplos de este concepto.

Cuando un niño aprende el "concepto" de "vehículo", lo observable externamente (considerando al niño como una 'caja negra') es que termina siendo eficiente en seleccionar los que en efecto son vehículos, ante la presentación de un cúmulo de diferentes objetos.

Una crítica, sin embargo, a la expresión "Aprendizaje de Conceptos" es que esta frase es demasiado ambiciosa respecto a lo que realmente logran los programas que supuestamente realizan este objetivo y que pertenecen a la corriente general de Inteligencia Artificial.

Tal vez no es el único efecto interno en un sistema vivo, el de llegar a contar en su cerebro con un clasificador de objetos.

Es probable que algo más suceda, por ejemplo que la máquina misma aprendedora (léase "encontradora") de definiciones clasificadoras sufra una **variación**.

En el ser humano podemos afirmar que el efecto de aprender un concepto se integra además en un sistema de lenguaje humano muy complejo y que simplemente por este hecho las consecuencias de un tal aprendizaje pueden ser grandes. No se restringen las

consecuencias a haber aumentado la competencia en clasificar objetos.

Por ejemplo, una persona con inclinaciones literarias manejará después diestramente la palabra aprendida o un técnico podrá incluso seleccionarla para denotar con ella un concepto asociado pero no idéntico.

Es por estas razones que a pesar de que de acuerdo con la costumbre en otros trabajos de IA podríamos incluir éste en la categoría de "Aprendizaje de Conceptos", hemos preferido usar en el resto de este escrito frases como "Búsqueda de Definición".

En el texto de Winston [1984] se usa la frase: '...búsqueda de una descripción de una clase de objetos...', la cual es muy adecuada. Nosotros empleamos la palabra 'definición' en vez de 'descripción' para enfatizar el hecho de estar distinguiendo a la clase de interés de un conjunto de objetos que no pertenecen a esta clase ('contra-ejemplos').

## BUSCADOR DE DEFINICION CLASIFICADORA

En el medio de IA, el término "búsqueda" tiene una conotación técnica bien definida.

Así por ejemplo, sabemos que estará implícito un árbol de búsqueda, operadores de generación de "nodos hijos", búsqueda en profundidad o amplitud, alguna función de evaluación y forma de memorización incluida para hacer "back-tracking's" etc..

En esta sección, discutiremos someramente algunas de las características generales encontradas en diferentes programas buscadores de definición, desde el punto de vista del espacio de búsqueda.

Hay varias formas para representar a los objetos que se pretende clasificar, y a la definición clasificadora.

Por ejemplo [WINSTON,1984], los objetos y las definiciones pueden estar representados como una 'gráfica', en el sentido de Teoría de Gráficas.

Independientemente de esto, el objetivo siempre es encontrar un conjunto B que tiene como subconjunto al A de ejemplos presentados al programa y que no incluye a los contraejemplos presentados.

Asociada con B está una gráfica ó una fórmula lógica ó un conjunto de números que representan conexiones or, una máquina neural, etc. Esta es la "definición" buscada.

Se tiene que decidir [Barr,1982] |



- . Qué representación tienen las definiciones.
- . Qué representación tienen los objetos.
- . Cómo se hace variar  $B$  hasta englobar a  $A$ .
- . Si se buscará al  $B$  más pequeño ó más grande posible, según la representación elegida, que excluye a los contraejemplos  $C$  y que incluye a  $A$ .
- . Si  $B$  puede o no **excluir** a algunos elementos de  $A$ .
- . Si  $B$  puede o no **incluir** a algunos elementos de  $C$ .
- . Si **todo** el conjunto  $A$  es dado al inicio, o bien , se dará gradualmente o por partes.

El tercer punto puede ser desglosado en:

- . Cuáles son los operadores de generalización, es decir, aquellos que generan diferentes  $B'$  a partir de cierto  $B$  inicial.
- . Cómo se selecciona la activación de un operador de generalización; por ejemplo: con un orden fijo o bien dirigidos por un objeto-ejemplo ó dirigidos por un objeto-contraejemplo.
- . Cómo se viaja en el árbol de los  $B$ 's generados y cómo se hace el back-tracking.

Así mismo, rápidamente se da uno cuenta que todo programa buscador de clasificación tropieza con la dificultad nada simple de evitar una explosión combinatoria.

El espacio de generalización posible es siempre demasiado grande.

Esta es, por supuesto la dificultad que hace técnicamente relevante el estudio de estos programas. El objetivo: 'buscar una definición' es sencillo de plantear ,pero requiere cuidado el conseguirlo en una aceptable medida y a la vez tener una **velocidad aceptable**.

En diferentes partes a lo largo de este escrito se encuentran descritas las opciones que se han elegido respecto a los puntos comentados en esta sección. Esta sección nos ayuda pues, a entender el porqué y el sentido de algunas de las siguientes secciones.

## CAPITULO II

## FUNDAMENTOS

## NOCIONES FUNDAMENTALES DE LOGICA Y NOTACION

En este artículo nuestra discusión se enfocará en conceptos de Lógica. Es por esto que conviene establecer aquí una definición de 'lenguaje de primer orden'. Se ha seleccionado la definición que da Enderton [1972].

## Lenguajes de Primer Orden.

Están constituidos por un número infinito de objetos a los que llamaremos símbolos, distribuidos como sigue:

## A Símbolos Lógicos.

0. Paréntesis: (, )
1. Símbolos sentenciales conectivos:  $\rightarrow$ ,  $\neg$
2. Variables.
3. Símbolo de igualdad (opcional).

## B Parámetros.

0. Símbolo cuantificador:  $\forall$
1. Símbolos de predicados (conjunto posiblemente vacío).
2. Símbolos de constantes.
3. Símbolos de funciones (conjunto posiblemente vacío).

En este trabajo denotaremos a las variables como aquellos símbolos que inicien con las siguientes: letras minúsculas: u, v, w, x, y, z.

Tenemos además las definiciones siguientes:

**Definición de término :**

- . los símbolos de constantes y de variables son términos.
- .  $(f t_1 t_2 t_3 \dots t_n)$  es término, si  $f$  es símbolo de función de  $n$ -lugares y  $t_1, t_2, \dots, t_n$  son términos.

**Definición de fórmula atómica :**

- .  $(P t_1 t_2 \dots t_n)$  es fórmula atómica, si  $P$  es símbolo de predicado de  $n$ -lugares y  $t_1, t_2, \dots$  son términos.

**Definición de fórmula bien formada , "fbf" :**

- . una fórmula atómica es una fórmula bien formada.
- . si  $\alpha, \beta, \alpha_1, \dots, \alpha_N$  son fbf entonces:

$$\begin{aligned} & (\text{AND } \alpha_1 \alpha_2 \dots \alpha_N ) \\ & (\text{OR } \alpha_1 \alpha_2 \dots \alpha_N ) \\ & \neg \alpha \\ & \alpha \rightarrow \beta \\ & \forall x \alpha \\ & (\alpha) \end{aligned}$$

son fbf.

A lo largo del programa, y en el texto, hemos denotado :

La fórmula " $\alpha \leftarrow \beta$ " denota a la fórmula " $\beta \rightarrow \alpha$ ".

'fatom', 'f-atom' o 'PIL' significa "fórmula atómica".

'expresión' o 'fórmula and' o 'f-and': palabras por las que entenderemos a una conjunción de fórmulas atómicas precedidas virtualmente ( no explícitamente ) por el cuantificador existencial 'EXISTE', excepto cuando formen parte de una cláusula de Horn (cláusulas que se comentan adelante), en cuyo caso el cuantificador universal 'PARA-TODO(A)(S)' se sobreentenderá actuando sobre los símbolos variables comunes al antecedente y al consecuente.

'fórmula OR' ó 'f-or' : estas palabras denotarán a una disyunción de f-and's.

### Cláusulas de Horn.

Una clase especial de fbf son las cláusulas de Horn.

Estas tienen la ventaja de ser de más fácil manejo en

programas manejadores de bases de datos de conocimientos, que son arbitrarias, como lo comenta Kowalski [1979]. Y además, según este mismo autor: "todo problema que puede ser expresado en lógica puede ser re-expresado por medio de cláusulas de Horn".

Una cláusula de Horn es una expresión de la forma

$$( \text{AND } A_1 A_2 \dots A_n ) \rightarrow A_0$$

donde las  $A_i$  son fórmulas atómicas. En este trabajo por conveniencia las representaremos por la lista:

$$( A_0 ( A_1 A_2 \dots A_n ) )$$

En este trabajo la parte que llamamos **teoría** es una base de conocimientos que está constituida por un conjunto de cláusulas de Horn.

Llamamos **ejemplo** a una fórmula-and. Suponemos que representa a una 'ocurrencia' o 'muestra' de una clase de objetos.

Llamamos **clase** a un conjunto de ejemplos.

Suponemos que hay algún significado externo o alguna causa externa en la agrupación de los ejemplos en las clases que se dan. Por ilustración: cada clase puede corresponder a un conjunto de pacientes que se sabe presentan la misma enfermedad.

A veces nos referimos al 'universo de clases' para denotar al conjunto de clases dadas.

Respecto cierta clase, al conjunto de todos los ejemplos de las otras clases le llamamos **contra ejemplos**.

## FORMALIZACION DEL OBJETIVO DEL PROGRAMA

En este artículo trataremos de explicar más claramente el objetivo o fin que perseguirá el programa de cómputo.

En forma todavía imprecisa podemos decir que dicho fin es el de, dado un conjunto de clases, encontrar en qué se parecen entre sí los objetos de una clase y en qué se distinguen de los objetos de las otras clases.

En otras palabras, buscamos una definición de cada una de las clases.

Tanto los ejemplos como las leyes que forman la teoría, se describirán por el usuario del programa (el cual puede ser otro programa, por supuesto) usando una notación bastante estándar de lenguaje de primer orden.

Esto es muy ventajoso dado lo bien definido y conocido de

este tipo de lenguajes y la abundancia de bibliografía.

De esta manera evitamos en gran parte la necesidad de que el usuario aprenda un lenguaje especial.

Además, así este trabajo queda de acuerdo con la filosofía de ir construyendo gradualmente una mesa de programas que tengan un potencial de fácil intercomunicación.

Se ha tenido que incluir un conjunto de símbolos reservados de predicados y de funciones. Sin embargo, este conjunto es relativamente pequeño y su uso no requiere prácticamente del desarrollo de ninguna habilidad adicional.

Los símbolos especiales son:

DIFTES	ES-ELE-DE ALGO	ALGO-DE-ESTO:	TODO
TODO-DE-ESTO:	C- > < I->	I-< I-><	I-<>

Estos símbolos pueden ser considerados como símbolos predicados constantes de un lenguaje de primer orden.

En la sección de 'Predicados y Funciones Especiales' se describen y definen su uso y su significado. Por el momento sólo se requiere saber que estos símbolos permiten el manejo de nociones de conjuntos, por un lado, y de intervalos de números reales, por otro.

Antes de poder precisar el fin u objetivo del programa, necesitamos introducir algunas pocas definiciones, las cuales se han propuesto con la intención de ordenar y facilitar la discusión.

Sea  $A$  subconjunto de ejemplos y sea  $f$  una fórmula-and.

Decimos que  $f$  es una generalización de  $A$  si a partir de la teoría subyacente, de cualquier ejemplo en  $A$  podemos demostrar la validez de  $f$ .

En notación simbólica estilo la de Enderton[1972] :

$$\forall e \in A : T_e \vdash f$$

La idea simplemente es que la fórmula lógica  $f$  "abarca", "describe" o "es común" a todos los ejemplos del subconjunto  $A$ .

Como ilustración:

Sea  $A$  el conjunto

$$\left[ \begin{array}{l} (x \text{ es animal con pelo, garras y caninos}), \\ (x \text{ es animal con pelo, pezuñas y molares}) \end{array} \right]$$

,entonces, una generalización de  $A$  es:

'  $x$  es animal con pelo '.

Adelantándonos un poco, el hecho de aprender el concepto implícito al dar la clase  $C_1$  en el contexto de otras clases  $C_2, C_3, \dots, C_n$  y de un conjunto de definiciones y conocimiento  $T$ , estará asociado con el hecho de encontrar una generalización de  $C_1$  que no sea válida en ningún ejemplo de  $C_2, C_3, \dots, C_n$ .

Pero este planteamiento no es suficiente para aclarar bien el proceso de "conceptualización" a partir de la base de conocimientos en el caso de la presentación de un número finito de elementos de una clase que contiene un infinito número de ejemplos distintos.

Debemos pues, tomar en cuenta lo siguiente:

**Las clases  $C_1, C_2, \dots, C_n$  no siempre son dadas en forma completa.**

Precisamente en este hecho radica el carácter objetivo, es decir, el carácter experimentalmente comprobable, del proceso de adquisición de una definición de una cierta clase o conjunto dado.

En otras palabras, el haber llegado a una definición o conceptualización aceptable de cierta clase está objetivamente bien definido en los siguientes términos:

La definición de la clase  $C$  es correcta, si (casi) nunca hay un error al serle presentado al sujeto un ejemplo nuevo de esta clase.

Así pues, uno de los sentidos y usos más importantes de un sistema-conceptualizador radicará en el acierto que tenga al hacer predicciones.

Además vemos que una característica de un sistema encontrador de definiciones es el empleo de un método de inferencia inductivo, en contraposición a uno deductivo. El carácter inductivo está asociado al carácter predictivo y estriba en que a partir únicamente de la presentación parcial de una clase de objetos no podemos asegurar en rigor la validez de ninguna definición que proponamos. Es decir, no podemos deducir que

cierta definición sea la correcta.

Tal vez la única manera de "aprender" en el sentido de "descubrir", sea aplicando un razonamiento o método inductivo. Es interesante, creo, plantear el siguiente ejemplo: A partir de ningún experimento se puede en Física demostrar ninguna ley. A lo más que podemos llegar en rigor, es a demostrar que el resultado experimental no contradice la ley de nuestro interés. De hecho, en más de una ocasión, leyes físicas antes aceptadas basándose en abundantísimos experimentos y experiencias han sido sustituidas por otras, que son una generalización de ellas.

Como ejemplo del carácter predictivo del hecho de "encontrar una definición", podemos mencionar la necesidad médica de acertar con un alto porcentaje, en el diagnóstico de una enfermedad.

Al tomar en cuenta los "porcentajes" o fracciones de clasificación acertada de un ejemplo dado, estamos "haciendo estadística".

Así pues, también debemos emplear criterios estadísticos al tratar de extraer una definición para una clase dada.

Es este argumento el que nos impide exigir que una fórmula  $f$  no sea válida en ningún contraejemplo, como una condición para ser una definición de la clase dada.

Otro aspecto relevante es el de tomar en cuenta que una clase bien puede estar formada por un conjunto de 'sub clases', en el sentido de que que es la unión de éstas y de que cada sub-clase satisface una fórmula-and distinta.

Por todo esto, nuestra definición de "candidato aceptable para la definición de una sub-clase", queda así:

$w$  fórmula-and es un candidato aceptable para definir al conjunto  $A$ , subconjunto de la clase  $C$ , sólo si según la teoría subyacente  $T$ ,  $w$  es válida con diferente frecuencia en  $A$ , que en el conjunto de contra-ejemplos de la clase  $C$  respecto al universo de clases dado.

Se pide diferente frecuencia y no mayor, para no rechazar a fórmulas atómicas que bien podrían formar parte de la definición de la clase. Hay un ejemplo en la sección de Ilustraciones de Uso del programa que aclara este caso.

En notación simbólica:

$w$  es aceptable para definir  $A$  solo si  $w$  es  $f$ -and  $y$

$$| p(w/A, T) - p(w/CEJS, T) | > \text{EPSILON}$$

,donde  $p(w/Z, T)$  es la probabilidad de que  $w$  sea válida en el conjunto de ejemplos  $Z$ , dada la teoría  $T$ .

De esta manera ya se está en la posibilidad de definir la meta del programa que se presenta como tesis. En realidad estamos definiendo la "función" o "procesamiento" a la que se deberá ajustar este programa:

Dadas, una clase  $C$  de ejemplos, una clase CEJS de contraejemplos y una Teoría cualesquiera, encontrar:

\* una división de  $C$  en subconjuntos  $S_1, S_2 \dots S_M$

tal que:

\* de cada  $S_i$  se tenga:

- o bien una definición aceptable, en el sentido arriba señalado
- o bien una indicación de falla en este subconjunto .

Así pues, nuestro programa deberá ser capaz de dar una definición de la clase  $C$  como una fórmula-OR de las f-and's asociadas a  $S_1, S_2 \dots S_M$ .

Debemos aclarar que en el programa, como un método heurístico, la implementación se hizo exigiendo fórmulas atómicas que se cumplan con frecuencia inferior en los contra-ejemplos.

#### FORMULAS TIPO "M-de-N"

A veces para clasificar a algunos objetos naturales, se requieren contar las características que resultan válidas de entre cierto conjunto previamente dado de ellas. Si el número pasa cierto umbral el objeto pertenece a cierta clase.

En los "sistemas-expertos" se toman en cuenta "nodos" o "reglas" que también operan con esta idea del "conteo-umbral".

Es por esto que creemos sumamente conveniente que un programa encontrador de definiciones clasificadoras sea capaz de encontrar también a definiciones del tipo mencionado.



Se debe hacer notar que en las matemáticas familiares en las diversas ingenierías no aparecen definiciones de este tipo. De hecho las definiciones de tipo umbral son un tanto "difusas".

Entenderemos por fórmula "m-de-n" a una de la forma:

(m válidas-de: <LISTA-DE-FORMULAS-ATOMICAS> )

,mediante la cual deseamos abreviar la afirmación:

son válidas al menos m de entre las siguientes aserciones: <LISTA-DE-FORMULAS-ATOMICAS>

Podemos considerar a una fórmula "m-de-n" como una generalización del concepto de conjunción de fórmulas atómicas. La generalización consiste en que en vez de pedir la validez de todas las fa's de una cierta lista pedimos ahora la validez de solo un cierto número de ellas.

Para mencionar un ejemplo de un caso en que se podrían utilizar fórmulas tipo m-de-n, imaginemos un problema de clasificación de fósiles. En algunos especímenes de transición se podría hacer un conteo del número de características "nuevas" contra el de "viejas" que están presentes con el fin de emitir un juicio acerca de la clasificación probable.

Una categoría de situaciones en que las fórmulas "m-de-n" son potencialmente útiles es la de tratar de clasificar a elementos a partir de datos algo inadecuados o de información insuficiente.

Por ejemplo, podríamos clasificar a los problemas aplicables para estudiantes de recién ingreso a la Universidad en la categoría de "difícil" si se cumplen al menos dos de las siguientes condiciones:

- 1 Se requiere el uso de 3 o más ecuaciones simultáneas.
- 2 La variable aparece en dos términos trigonométricos.
- 3 Es necesario efectuar una integración.
- 4 El tiempo dado para resolver el problema es de 20 minutos.
- 5 Alguna ecuación de las que intervienen es vectorial.

Observemos que esta definición de "difícil" parece intuitivamente razonable a pesar de que no se cuenta con información detallada acerca de los problemas que deberá resolver el alumno.

#### FUNCIONAMIENTO GLOBAL DEL PROGRAMA

El programa ha sido concebido como un conjunto de "máquinas" procesadoras de datos en una "línea de ensamblaje", de tal manera que cada una de ellas transforma "ligeramente" a lo corriente de

datos que le llegan y su salida sirve para alimentar a otra máquina .

Cada máquina la asociamos a una cierta función en LISP.

A continuación explicamos el funcionamiento de las funciones principales en este programa:

#### PROOF-EVER.

Le llega :

\* Un conjunto A de ejemplos y un conjunto CEJS de contraejemplos.

\* Una f-and , que es la definición inicial o anterior del conjunto A ; así se pueden ir construyendo nuevas definiciones en diferentes sesiones de trabajo.

Sale : \*

\* Una f-and w ,

\* Un subconjunto B de A ( o NIL en caso de fracaso) de tal manera que w es válida en todo B con mayor frecuencia que en los contraejemplos.

#### Comentarios:

\* Frecuentemente la w de entrada a proof-ever es el primer ejemplo de la clase, o bien , un ejemplo especialmente adecuado a criterio del usuario o 'entrenador'.

\* El conjunto A debe estar formado por ejemplos que muy probablemente tengan algo en común.

#### GENERALIZA.

Es usada por PROOF-EVER. Su misión es generalizar una determinada f-atómica sistemáticamente, hasta lograr que sea válida en cierto ejemplo.

#### FRON.

Es usada por PROOF-EVER. Su nombre deriva de la idea de su función: trata de generalizar a una fórmula-and 'w' de entrada de tal manera que el conjunto asociado en principio a w casi toca la "frontera" de los contra-ejemplos.

#### WS-CONJS.

Usa a PROOF-EVER. Le llega un conjunto de ejemplos que

parecen tener algo en común y obtiene una lista formada por elementos del siguiente estilo:

(f-and , subconjunto en que f-and es válida ).

En otras palabras, obtiene una f-or descriptiva del conjunto que le llega y que no es válida en los contraejemplos.

#### F-OR-TODO.

Le llega una clase de ejemplos. O bien, en casos en que la clase de ejemplos sea muy grande, le debe llegar una selección de ejemplos (tal vez al azar ) para evitar una explosión combinatoria. También le llega una selección de contraejemplos.

Divide a la clase en subconjuntos, a cada uno de los cuales lo hace pasar por la máquina WS-CONJS y obtiene así finalmente una fórmula-or-n-de-m que describe total o parcialmente a la clase de entrada.

En el caso de una descripción parcial ,sale de ella información suficiente como para saber en qué subconjunto de ejemplos ha fallado.

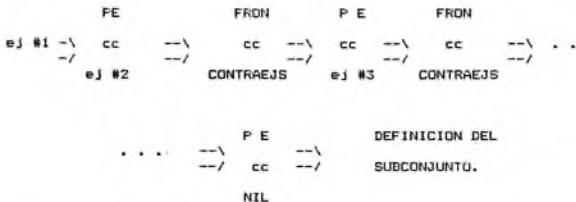
#### Comentarios:

‡ No se puede asegurar que la máquina F-OR-TODO tenga éxito en todo el conjunto de ejemplos de entrada, pues puede ocurrir que de acuerdo con los 'conocimientos' dados , los ejemplos no sean distinguibles respecto a los contra-ejemplos.

A continuación presentamos unos diagramas ilustrativos del modo de operar de las funciones PROOF-EVER y de GENERALIZA:

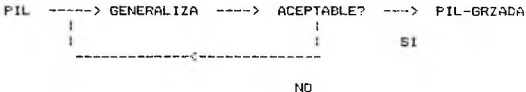
A continuación presentamos unos diagramas ilustrativos del modo de operar de las funciones PROOF-EVER y de GENERALIZA:

PROOF-EVER:



,donde hemos denotado: "PE" por "PROOF-EVER" , "cc" denota a la frase "compara a los datos de entrada contra :", y las flechas indican flujo de datos.

GENERALIZA:



## UNA BASE DE CONOCIMIENTOS COMO DATO DE ENTRADA

Todo programa usa implícitamente al menos, una base de conocimientos.

Esta Base de Conocimientos, podría estar distribuida a lo largo de todo el programa y ser por consiguiente difícil sustituirla o cambiarla.

Sería difícil adecuar al programa para un uso distinto al inicial.

Por esta y otras razones es sumamente conveniente el hecho de usar un conocimiento **explícito**.

Por ejemplo, si se incluye un conocimiento suficientemente amplio ( teorías de conjuntos, números, álgebra, etc. ) podemos atacar un cúmulo grande de problemas de clasificación, tales como :

- . Reconocimiento de letras
- . Reconocimiento de texturas
- . Reconocimiento del patrón que sigue una secuencia de números
- . Reconocimiento de reglas de manipulación algebraica (por ejemplo, aprender a usar al binomio de Newton)
- . Reconocimiento de algunos dibujos planos simples

Requiere por supuesto, una considerable cantidad de labor el hecho de incluir una base de conocimientos muy extensa. Por otro lado, para usarla sin gasto excesivo de tiempo de máquina, se requeriría un hardware más poderoso y afinar detalles de software tales como incluir técnicas de indización especiales al explorar la Base de Conocimientos.

Si bién por estas cuestiones este programa todavía no puede usar eficientemente una base de conocimientos muy extensa, creemos que el hecho de poder incluir como dato a bases pequeñas ya lo hace interesante en el cúmulo de trabajos similares y además, permite ya una cierta flexibilidad y capacidad de adaptación .

Pensamos que la experiencia ganada al desarrollar y usar este trabajo, permitirá realizar mejoras sustanciales en el uso de bases de conocimientos para encontrar definiciones clasificadoras.

Y probablemente también nos de experiencia de valor en el uso de estas definiciones clasificadoras para ampliar, a su vez, a la base de conocimientos.

Después de todo, la base de conocimiento y las definiciones están representadas en este programa en el mismo lenguaje.

Además, una definición aprendida es realmente nuevo conocimiento.

## CAPITULO III

## IMPLEMENTACION

## LEYES DE UNIFICACION ASUMIDAS

Aceptaré las leyes de unificación que vienen en el texto de Wilensky [ 1984 ] pp 296 ,salvo que consideraré la posible conmutatividad de los argumentos de los predicados o funciones :

\* Una variable , tanto en la expresión patrón como en la expresión sustrato , unifica con cualquier cosa si no hay 'circularidades'.

Si las hay , fracasa la unificación.

\* Un simbolo constante unifica sólo con otro simbolo igual

\* Una lista unifica con otra lista sólo si

- Los primeros elementos son simbolos constantes y unifican. Es decir , el simbolo predicado o función es idéntico.

- El resto de elementos unifican con conmutatividad o no, según la propiedad de conmutatividad del primer elemento.

Como manejar a las variables

Las variables que aparecen en dos fórmulas atómicas por unificar son esencialmente 'mudas'. Esto es, sus nombres pueden ser sustituidos por cualquier otro.

Es necesario evitar el problema de encontrarse con símbolos variables que son el mismo, tanto en la primera fórmula como en la segunda.

Tal como están enunciadas arriba las leyes de unificación, estas variables serían tratadas como denotando al mismo objeto. Pero sabemos que no denotan realmente al mismo objeto, pues pertenecen a fórmulas distintas.

Lo más fácil para resolver este problema, y es la solución que he adoptado, es modificar los nombres de las variables en la segunda fórmula de tal manera que automáticamente sean distintos a los de la primera.

Una forma de lograrlo es agregarles a sus nombres un índice al final. Este índice refleja el nodo en el árbol de demostración en que aparece esta fórmula.

Obsérvese que aparece un árbol de 'evaluación' (léase 'demostración' o 'prueba') debido a que estamos tomando en cuenta una base de cláusulas de Horn subyacentes (la cual es la 'teoría' de fondo) para poder decidir si cierta fórmula entra en PROOF-EVER se evalúa en 'TRUE' o 'FALSE' para cierto ejemplo de una clase dada.

#### USO DE GENERALIZACIONES EN LA BÚSQUEDA DE UNA DEFINICIÓN DE UNA SUB-CLASE

Estamos buscando, dado un subconjunto  $A$  de la clase  $C$ , una fórmula  $w$  tal que:

$$\forall e \in A: e, T \models w \quad (1)$$

y tal que  $w$  no sea válida con la misma frecuencia en un conjunto de contraejemplos dado.

Por la fórmula (1) anterior observamos que esta  $w$  es simplemente una generalización de los ejemplos en  $A$ .

El problema al buscar  $w$  es el de dirigir de alguna manera adecuada la búsqueda según las restricciones de cómputo que se tengan.



Esto es crítico dado el conjunto tan grande de posibles generalizaciones de  $A$ , pues cualquier  $w$  tal que

$$\forall e \in A : T_e \vdash w$$

es una generalización de  $A$ .

Por otro lado resulta conveniente enfatizar que esta definición de generalización es muy clara, comparada con la que se da en otros trabajos [NICHALSKI 1980].

En este trabajo consideramos de manera directa solo a las generalizaciones  $w$  de  $A$  producidas por un conjunto restringido de operadores de generalización.

#### OPERADORES DE GENERALIZACION DE UNA FORMULA ATOMICA

Los operadores usados sobre una fórmula atómica  $fa$  que aparece en la fórmula  $w$ , producen los cambios:

- 1 Un símbolo constante cambia a un símbolo variable no ocurrente en  $w$ .
- 2 Un símbolo variable cambia a algún símbolo variable no ocurrente en  $w$ .
- 3 Generalizaciones especiales para fórmulas atómicas construidas con símbolos predicados y funcionales reservados ( DIFTES , ES-ELE-DE , etc., ).
- 4 Eliminación de la fórmula  $fa$  en  $w$ .
- 5 Substitución de un símbolo predicado por un símbolo predicado "más general".

Hemos dirigido la búsqueda mediante el hecho de

"generalizar sólo aquellas  $f$ -atómicas de  $w$  que fallan de demostrarse"

PROOF-EFVER trata de demostrar o probar (de aquí el nombre que se le dio a esta función: "proof") cada  $f$ -atómica de una  $w$

dada, en base a la teoría y a un ejemplo. Si esta prueba falla entonces aplicamos un operador de generalización que se elige de tal manera que se trata de abarcar a una de las fa del ejemplo ("fa modelo"). Si continúa la falla, se prosigue con la siguiente fa del ejemplo.

En la siguiente sección, "Leyes de Generalización", se ven con detalle las reglas que se aplican para generalizar a una fa dada dirigiéndose por una fa "modelo".

## LEYES DE GENERALIZACION

La siguiente definición es de utilidad:

El predicado A es una generalización de B si siempre que B es válida 'evaluada' en ciertos términos  $t_1 t_2 \dots t_n$ , entonces A también lo es evaluada en los mismos términos

, o sea:

A es generalización de B si:

$$(A \ t_1 \ t_2 \ \dots \ t_n ) \leftarrow (B \ t_1 \ t_2 \ \dots \ t_n ) .$$

Las fórmulas construidas a partir de las fórmulas atómicas se generalizan exclusivamente mediante la aplicación a estas últimas de algún operador de generalización.

## REGLAS PARA APLICAR LOS OPERADORES DE GENERALIZACION

- 0 Se aplican al ir haciendo la unificación de una fátom de la conjunción candidato ("patrón") contra alguna fátom de un ejemplo ("guía").
- 1 Si a un símbolo constante del patrón le corresponde un objeto distinto de la guía, cambiar el símbolo del patrón por una variable.

Esta variable es nueva si no hay variable nueva del patrón ligada al objeto de la guía.

En caso contrario se elige alguna al azar, de las variables nuevas ligadas al objeto.

- 2 Si a una variable del patrón, que ya tenga ligadura previa  $l_1$ , le corresponde una estructura  $l_2$  y resulta que éstas no son unificables, entonces cambiamos el nombre de esta variable:

- \* A uno nuevo, si  $l_1$  no está ligada a ninguna variable nueva del patrón.
  - \* En caso contrario, a uno de los ligados a  $l_2$ .
- 3 Si el predicado del patrón no corresponde al de la guía, se cambia el nombre del predicado del patrón por alguno, escogido al azar, del conjunto de predicados "más cercanos al patrón" que son generalizaciones tanto del predicado del patrón como del predicado del guía.

"Más cercano al patrón", significa aplicar en orden:

- \* más cercano al patrón en número de pasos de generalización.
  - \* más cercano al guía, en número de pasos de generalización.
- 4 Las reglas para generalizar a fórmulas construidas usando en parte símbolos de predicados o funciones especiales, abarcan las anteriores y algunas más, específicas de cada símbolo especial. El principio director para el diseño de estas reglas es el mismo que el usado en las reglas 1 a 3: simplemente encontrar una forma nueva que sea válida cuando las dos formas patrón y guía, son válidas. Para no distraerse en detalles, se remiten éstos a la sección de "Funciones y Predicados Especiales".

#### INCONVENIENCIA DE BUSCAR LA DEFINICION MAS RESTRICTIVA

Muchas veces se busca "la definición más breve" o "más esencial" de una clase dada de objetos. Podemos concretizar la idea intuitiva de "esencial" y traducirla por "corta". Es decir, una buena definición será aquella que sin "muchas palabras" logre describir a la clase y distinguirla de las contra-clases.

En este artículo proponemos una medida de longitud de una fórmula de las que maneja el programa, y discutimos el compromiso entre la "brevedad" y el "grado de restricción".

Definimos así longitud:

1. La longitud de un átomo es la unidad.

2. La longitud de la lista  $L$  es la suma de las longitudes de los elementos de  $L$ , entendiendo ésto en sentido recursivo.

Dadas  $P_1$  y  $P_2$  expresiones, siendo ambas definiciones de la clase  $C_1$ , puede ocurrir que  $P_2$  sea de longitud mayor que  $P_1$  y que además sea más restrictiva que  $P_1$ .

Es por esto que no necesariamente las definiciones de longitud mayor son las "peor".

La longitud de una expresión será relevante al comparar dos expresiones  $P_1$  y  $P_2$  que sean equivalentes en el sentido lógico:

$$P_1 \leftrightarrow P_2$$

En este caso debemos elegir a la expresión de menor longitud (a veces se le refiere como la "expresión más elegante").

Tenemos por lo tanto un compromiso: debemos buscar las definiciones más cortas, por un lado, y por otro debemos encontrar las definiciones más restringidas (i.e., menos generalizadas) tomando en cuenta que existe una tendencia en general, experimentalmente observada, a que las definiciones más cortas sean las más generalizadas.

Este compromiso nos obligará a tomar criterios empíricos en el diseño del programa.

Hay varios sentidos posibles de la idea de dar la definición "más restringida" posible de una clase.

Un sentido importante que se le puede dar a la frase "definición más restringida" es el siguiente:

Una fórmula  $P$  finita tal que para toda fórmula  $F'$  finita que sea válida en cada uno de los ejemplos de la clase  $C$ , se tenga que: (0)

$$P \rightarrow F'$$

No es evidente que esta expresión  $P$  finita exista. De hecho la mencionada  $P$  es equivalente a:

"La conjunción de todas las definiciones finitas de la clase  $C$ " (1)

donde "las definiciones de  $C$ " significa el conjunto de las expresiones definiciones finitas de  $C$ , que son construibles de la

Cierta teoría, es decir, dado "cierto conjunto de símbolos y reglas lógicas que definen a cierta 'gramática'".

Si es infinito el conjunto de definiciones de C, entonces no es evidente que P construable exista. Por supuesto, la expresión (1) es una meta-expresión y no es la expresión que estamos buscando. Estamos buscando expresiones de lenguaje lógico de primer orden.

Si el conjunto de ejemplos  $C = \{e_i\}$  es finito, entonces el P referido en (0) existe y es equivalente a:

$$P = (O R e_1 e_2 \dots) = \sum_i e_i$$

, puesto que para toda P' definición de C, como  $e_i \rightarrow P'$  para todo i, entonces si P es válida, alguno de los  $e_i$  será válido y por lo tanto P' será válida, satisfaciendo P por consiguiente a la definición (0).

Fero entonces una definición que sea de las las más "restringidas" posible en el sentido de (0), no es interesante para clases C finitas, pues resulta ser sólo un listado de todos los ejemplos dados.

Debido a esto y a que al programa que será entrenado no se le presentarán un número infinito de objetos de la clase de interés (tampoco a un ser vivo se le presentan un número infinito de elementos durante su aprendizaje, observándose también en ellos la restricción de buscar una definición que más que rigurosa, sea eficiente), debemos buscar una definición con los criterios:

1. entre más corta mejor.
2. entre "más restringida" mejor, según alguna medida empírica de grado de restricción.

Lo que hace interesante a este problema es que estos dos requisitos son teóricamente independientes y que por otro lado es frecuente que una expresión que es cambiada para hacerse más corta se vuelva menos restringida.

Obsérvese que hemos renunciado a buscar la definición mas restrictiva. En vez de ello ahora aceptamos alguna medida, incluso sólo empíricamente fundamentada, de grado de restricción.

Es difícil establecer una medida absoluta del grado de restricción que cierta expresión establece.

Medir es comparar y dadas  $F_1$  y  $F_2$  tales que no se tiene

$$F_1 \rightarrow F_2 \quad \text{ni} \quad F_2 \rightarrow F_1$$

¿Cómo saber cuál de  $F_1$  o  $F_2$  es más restrictiva?

No es posible saber esto en el caso general, mismo que es el que nos interesa. Es posible cuando a cada  $F$  se le puede asociar unívocamente un conjunto finito (o 'medible') de elementos, pero esta situación se da más bien en algunos campos de matemáticas que en el campo "general" para el que está dirigido nuestro programa.

Por estas razones, en el diseño del programa se adoptó el criterio de buscar las definiciones con menor número de símbolos y en vez de buscar las definiciones "más restrictivas", lo que hice fué continuar en la dirección de la fórmula atómica a la que se la hayan cambiado el menor número de símbolos durante la generalización a que estuvo sujeta.

Por estas razones he adoptado una definición "puramente basada en la forma". Además solo comparo a  $f$ 's derivadas de una misma  $f$ :

Defino el grado de generalización que ha sufrido una  $f$  como el número de símbolos que han sido cambiados en ella.

En cuanto a la 'pérdida de información' resultante de seguir este criterio, podemos afirmar que variará de acuerdo con que clases se estén estudiando y que el mismo concepto de 'cantidad de información' resulta oscuro cuando se admite un conjunto muy diverso de 'clases de objetos' posibles, tales como: pacientes con cierta enfermedad, letras, etc..

Para nosotros resulta satisfactorio el hecho de que aplicando este criterio el programa ha salido a vante en casos razonablemente complicados y similares a los encontrados en la práctica.

#### BUSQUEDA EN LA DIRECCION DE LA MENOR GENERALIZACION

Hemos definido un conjunto finito de operaciones

sobre una fórmula atómica. De esta manera todo cambio grande o 'global' de cierta  $f$ -and será en nuestro programa producido por una sucesión de cambios elementales.

Así podemos establecer la búsqueda de una definición a partir de cierta expresión, en forma tal que la expresión se va cambiando "gradualmente y en forma mínima", es decir siguiendo primero la dirección del "menor número de cambios elementales".

Este método es similar a tratar de resolver un problema numérico siguiendo preferentemente el camino opuesto del gradiente.

El método del menor cambio que hemos usado consiste en:

Se hacen cambios elementales en una  $f$ a fallida, dirigiéndose por una  $f$ a del ejemplo en cuestión, hasta que la primera se transforma en una fórmula válida en el ejemplo o se alcanza la fórmula nula.

La fórmula transformada se desecha si es válida con mayor frecuencia en el conjunto de contra-ejemplos que en el de ejemplos. Este criterio se interrumpe temporalmente cuando el programa todavía no ha filtrado la fórmula a través de un número suficiente de ejemplos.

Si la  $f$ a no es desechada, se añade a una pila ordenada según el número de cambios elementales efectuados.

Como podemos observar, el hecho de elegir operadores de cambios elementales nos permite establecer el número de esos cambios como una medida del 'grado de generalización'. Y esto nos permite ordenar la búsqueda, y dirigirla por lo tanto en la dirección de la menor generalización (...de todas las generadas con nuestros operadores ...recuérdese que no se puede hablar de la definición menos generalizada admitiendo un espacio de búsqueda irrestricto...).

También se hace más pequeño el problema de la búsqueda en el sentido de que variamos una expresión tratando de abarcar a un ejemplo, en vez de primero generar expresiones arbitrarias con algún método de generalización más prolífico, y luego seleccionar las "buenas" de entre ellas.

Hay sin embargo, algunas preguntas relevantes:

1. ¿ es un método adecuado en general ?
2. ¿ cuándo es de esperar que si sea adecuado ?

3. ¿ es posible que siguiendo así la búsqueda se llegue a un estado "de meseta" o sea uno a partir del cual no podemos seguir "avanzando" ?
4. ¿ podemos decir en qué casos este método llegará a una definición buena en el sentido de
  - . muy corta
  - . muy restringida ?

#### Comentarios o Respuestas:

1. Este método "gradual" no es adecuado en general si la definición "buena" no está "casi contenida en cuanto a forma" en los ejemplos  $e_i$  de la clase C.

Una ilustración de este caso es la siguiente:

Sea la Teoría:  $(H x) \leftarrow ( (A x) \& (B x) )$   
 $\quad \quad \quad \vee ( (C x) \& (D x) )$   
 $\quad \quad \quad \vee ( (E x) \& (F x) )$

Sean los Ejemplos:

$e_1: (A x) \& (B x)$

$e_2: (C x) \& (D x)$

$e_3: (E x) \& (F x)$

Entonces ,por pequeñas variaciones de  $e_i$  con  $i = 1, 2$  ó  $3$ , nunca podremos llegar a la definición "buena" :  $(H x)$ .

2. El método resulta adecuado cuando la definición buscada esta "casi contenida" en los ejemplos de la clase, en el sentido de que se puede llegar a ella con una sucesión de cambios elementales a partir de cualquier ejemplo de la clase.
3. No es posible quedarse en una meseta ya que se sigue el camino de la menor generalización solo de manera preferente.
4. En la práctica en los ejercicios experimentados, se ha podido llegar a definiciones muy cortas, las cuales, por otro lado son deseables por ser "elegantes".



## GENERALIZACION POR PREDICADOS LIGABLES

Decimos que dos predicados son ligables si son casos especializados de un predicado superior.

Por ejemplo, "tío" y "padre" son ligables por el predicado "familiar-ascendientes".

En estos casos se busca la generalización viajando a través de una "red" que tiene como nodos a los nombres de los predicados, y no se viajará por lo tanto haciendo una unificación detallada con diferentes fátoms siguiendo a las cláusulas dadas en la base de conocimiento.

Esta manera de abordar el problema nos asegura un funcionamiento más rápido.

Así mismo, para implementar esta idea debemos asegurarnos que las fátoms por unificar generalizando sean unificables en todos sus otros elementos (esto es, todos salvo el primero, que es el símbolo predicado). Eso se toma por supuesto en cuenta en el programa PROOF-EVER.

En el programa se tiene que definir esta red de predicados ligables en forma explícita por el usuario. Para auxiliar hemos incluido una función LISP adecuada ("PUT-CADENA").

## PREDICADOS Y FUNCIONES ESPECIALES

El conjunto de símbolos predicados y funcionales reservados es el siguiente:

DIFTES	ES-ELE-DE	>	<
ALGO-DE-ESTO:	ALGO	TODO-DE-ESTO:	TODO C-
I-><	I-<>	I->	I-<

El primer renglón es de símbolos predicados, los demás de símbolos funcionales. El significado que intentamos otorgarles a los primeramente mencionados es:

(DIFTES a b c . . .)	denota a la frase: "a,b,c,... son diferentes entre si".
(ES-ELE-DE x y)	denota: "x es elemento de y".
(> x y)	denota: "x es mayor que y".
(< x y)	denota: "x es menor que y".

Por otra parte, los símbolos funcionales ALGO, TODO y sus variaciones notacionales adquieren sentido para nosotros en construcciones del tipo:

(CUMPLE árbol (TODO-ESTO: .... ))  
 (CUMPLE x (ALGO-DE-ESTO: ... ))

Al predicado CUMPLE no lo hemos manejado como uno reservado. El usuario podría utilizar SATISFACE, por decir algo.

Los términos formados con los símbolos tipo conjunto se generalizan haciendo intersección para el caso de TODOS y haciendo unión para el caso de ALGO.

Además tenemos a los símbolos funcionales:

$I \rightarrow$   $I \leftarrow$   $I \langle \rangle$   $I \rightarrow \langle \rangle$

estos denotan intervalos de números reales:

fórmula	denota a los números x tal que:
$(I \rightarrow a)$	$x \geq a$
$(I \leftarrow a)$	$x \leq a$
$(I \langle \rangle a b)$	$x \leq a$ o $x \geq b$
$(I \rightarrow \langle \rangle a b)$	$x \geq a$ y $x \leq b$

Las reglas de generalización de los términos o fórmulas atómicas que se pueden construir a partir de los símbolos comentados en esta sección son fácilmente planteables, si se toma en cuenta que la fórmula final debe ser cierta cuando se suponen ciertas las dos fórmulas patrón iniciales.

Por ejemplo,

(ES-ELE-DE x (ALGO 1 7 19 ))

se puede generalizar a:

(ES-ELE-DE x (I  $\rightarrow$  1 ))

Describimos estas reglas de generalización mediante la tabla de la siguiente página.

TABLA DE REGLAS DE GENERALIZACION DE SIMBOLOS FRECUENTES Y FUNCIONALES ESPECIALES.

#	PATRON 1	PATRON 2	PATRON RESULTADO
1	( = x a )	---	(ES-ELE-DE # (ALGO-DE-ESIG: a ) )
2	(ALGO a1 a2..an)	(ALGO b1 b2..bn)	(ALGO unión de (a1) & (b1) )
3	(TODO a1 a2..an)	(TODO b1 b2..bn)	(TODO intersección de (a1)&(b1))
4	( > x a )	( > x b )	( > x el-menor-de a y b )
5	( < x a )	( < x b )	( < x el-mayor-de a y b )
6	( > x a )	( < x b )	(ES-ELE-DE x (I-<) b a ) ,si b < a.
7	( > x a )	---	(ES-ELE-DE x (I-> a ) )
8	( < x a )	---	(ES-ELE-DE x (I-< a ) )
9	... las reglas para I-> y I-< son similares a las reglas 4,5 y 6.		
10	( I-> x A ) ( I-< x A ) } }	---	(I->< C D ) ,donde alguno de C y D es A y el otro es
11	( I->< a b )	(I->< c d)	(I->< el-menor-de a y c el-mayor-de c y d )
12	( I-<> a b )	(I-<> c d)	(I-<> el-mayor-de a y c el-menor-de b y d )
13	(I->< a b)	(I-<> c d)	... el intervalo resultado es la unión de los intervalos. Si esta unión es todos los reales, el resultado es "falla"

## CONCLUSIONES

\* Una aportación de éste trabajo es la proposición de que las bases de conocimientos entren como dato. Así tenemos una gran flexibilidad y más posibilidades de investigación.

\* Otra aportación es la proposición de tomar en cuenta la frecuencia o probabilidad con que cierta fórmula resulta válida en los ejemplos y contra-ejemplos. Gracias a esto la salida de este programa es similar a las reglas de un sistema experto, y por lo tanto puede ayudar a generarlos de manera automática.

Debemos aclarar que han aparecido en la literatura métodos para buscar clasificaciones que toman en cuenta las frecuencias, como p.ej. el de seguir el camino de la menor entropía, pero no sabemos de proposiciones para aplicarlos en programas que usan una notación como la de primer orden para describir a sus objetos.

\* Se propone además el uso de expresiones tipo:

"se cumplen al menos  $m$  de las siguientes  
 $n$  propiedades:..."

, las cuales describen muy adecuadamente a muchas clases de objetos naturales.

\* Es ventajoso que las clases de ejemplos, la base de conocimiento y las definiciones encontradas estén en la misma notación y que esta notación sea la estándar de lenguaje de predicados de primer orden.

\* Las heurísticas empleadas le confieren una velocidad de operación aceptable en el conjunto de pruebas efectuado, sin haber detectado una reducción del grado de alcance del objetivo. No se detectan pérdidas de información relevante durante los procesos de generalización empleados.

\* Las respuestas o salidas del programa, es decir, las definiciones de clase, son bastante inteligibles para un usuario no especializado en computación, aún sin emplear una interface traductora.

\* El rango de aplicación de este programa es bastante amplio, en el sentido de que el hecho de describir a los objetos o ejemplos de una clase en un lenguaje de primer orden permite abordar un conjunto muy diverso de casos.

### Mejoras Posibles a Futuro:

- \* Que las reglas de generalización entren como dato y en el mismo lenguaje de primer orden que los otros datos.

Esto tendría varias consecuencias interesantes. Una es que el investigador podría variar la arquitectura del programa más fácilmente. Otra consiste en que se pueden hacer ensayos en la dirección de crear un programa que modifique sus propias reglas de operación. El autor estima sin embargo, que llevar a cabo este objetivo requiere reescribir completamente el código del programa.

- \* Tener una base de conocimientos con jerarquía y herencia.

Implementar esto en una primera versión no es tan difícil como el punto anterior, estimo que se requerirán unas 160 horas.

- \* Incluir métodos de indexación en el uso de la base de conocimientos. De esta manera se aumentará la velocidad.

También estimo unas 160 horas para este fin. El objetivo es que dado un símbolo predicado se obtenga rápidamente un conjunto de apuntadores a las reglas de la base de conocimientos en las que aparece.

- \* Poder usar eficientemente datos que radiquen en memoria secundaria ( p.ej., disco duro en una IBM-PC ).

Esto es importante para poder usar una base de conocimientos extensa en una microcomputadora relativamente pequeña. Otra posibilidad de uso es tener una tal base que pueda ser usada por varios programas independientes de inteligencia artificial, y de esta manera todo aumento en la base de conocimientos beneficiará a varias investigaciones o usuarios distintos, o bien a varios programas complementarios que estén trabajando concurrentemente formando alguna máquina global. Estimo también unas 160 horas para realizar esta ampliación.

Los últimos puntos van en la dirección de facilitar investigaciones llevadas a cabo con bases de conocimiento amplias.

## BIBLIGRAFIA

- Barr ,A., and Feigenbaum, E. A., (eds), The Handbook of Artificial Intelligence, William Kauffman, Inc., Los Altos, CA, 1982.
- Enderton, A Mathematical Introduction to Logic, Ac.P., 1972.
- Kowalski, Robert, Logic for Problem Solving, North Holland, New-York Oxford, 1979.
- Michalski, Ryszard S., IEEE Transactions on Pattern Analysis and Machine Intelligence, VOL. PAMI-2, no4, July 1980.
- Wilensky, Robert, Lisp , NW Horton & Company, Inc., NY 1984.
- Winston, P.H., Artificial Intelligence, Addison-Wesley, 1984.

## apéndice: ILUSTRACIONES DE USO DEL PROGRAMA

## ----- USO DEL PROGRAMA -----

Se pretende que sea usado en forma interactiva.

El usuario puede usar el filtro principal F-OR-TODO eligiendo al azar algunos ejemplos y contraejemplos. La salida será una primer definición de la clase elegida.

Puede intentar comprobar que tan satisfactorio es el resultado comprobando la definición candidato con otros ejemplos y contraejemplos, usando para esto al programa PROOF-EVER.

PROOF-EVER dá como respuesta a un conjunto y una fórmula-and. El complemento de dicho conjunto es 'un conjunto rebelde' en el sentido de no ajustarse a la fórmula-and arriba mencionada .

En caso de falla, el usuario puede tratar de definir alguna otra clase o bien, intentar definir a un subconjunto rebelde de otra clase.

Se puede guardar en disco el estado del 'Area de Trabajo', y proseguir semanas después, cuando ya se cuente con nuevos ejemplos o contraejemplos.

Por supuesto, utilizando a F-OR-TODO, PROOF-EVER y otras funciones definidas en el programa como "primitivas", un usuario puede hacer investigación o bien buscar heurísticas que sean particularmente eficientes para la aplicación de su interés.

Precisamente una de las razones para no hacer una máquina que intente substituir al usuario humano en el proceso planteado en los primeros renglones de este artículo , es que considerando que puede haber muy diversas situaciones específicas de aplicación de este programa , resultará probablemente inadecuado en alguna aplicación, todo diseño que se pueda dar en este momento.

## -----\* LISTADOS DE LAS FUNCIONES PRINCIPALES \*-----

A continuación se da una lista de las funciones principales :

F-OR-TODO  
 PROOF-EVER  
 UNIFG-ESPECIALES

GENERALIZA  
FRON  
ACEPTABLE?

; File: PPALES-2.LIB

(PUTD 'DEFUN ' (NLAMBDA (NAM\$ EXP\$) (PUTD NAM\$ EXP\$) NAM\$))

(DEFUN F-OR-TODO (LAMBDA (NU-EJS NCLASE CONTRA-EJS)  
(C (L 'PRINTN ' (F-OR-T-- NU-EJS NCLASE F-OR-T--FIN)))  
(FROG1  
(F-OR-TODOO (CONJS-ASOC (NU-EJS))  
(PRINT NIL) ) ) )

(DEFUN F-OR-TODOO (LAMBDA (CONJS)  
(C (L 'PRINTN ' (F-OR-TO CONJS F-OR-TO---FIN)))  
( (NULL CONJS) NIL)  
(U (F-OR-D-ANDS (POP CONJS)) (F-OR-TODOO CONJS)) ) )

(DEFUN PROOF-EVER (LAMBDA (EXP NU-EJS CONTRA-EJS EJS-W PEJS FCEJS TEXP NEJS  
NO-ACEPTADAS)  
(C (L 'PRINTN ' (PE----- EXP NU-EJS CONTRA-EJS EJS-W -----PE-FIN)))  
(= NGJS-O (LENGTH NU-EJS))  
(= TEXP (TA EXP))  
( ( (NULL PEJS)  
(= PEJS (= FCEJS 100)) ) ) )  
(PROOF-EVER1 EXP NU-EJS CONTRA-EJS EJS-W TEXP NEJS-O PEJS FCEJS) ) )

(DEFUN PROOF-EVER1 (LAMBDA (EXP NU-EJS CONTRA-EJS EJS-W TEXP NEJS-O PEJS  
FCEJS SUDEXP TEORIA+E FIL-SACADA? PILA-SUBEXPS EJ AX FR FE-NEWVARS NUME  
(C (L 'PRINTN ' (PE----- EXP NU-EJS CONTRA-EJS EJS-W -----PE-FIN)))  
(= AX (PROOFAX (NARCAR-EXP EXP) ' 0))  
(LOOP  
( (NULL NU-EJS)  
(= NUMEJ (POP NU-EJS))  
( (NONNULL (= EJ (EJEMPLO NCLASE NUMEJ)))) ) )  
(PUSH NUMEJ NU-EJS)  
( (NULL EJ)  
( (NULL EXP) NIL)  
( (NULL EJS-W) NIL)  
( (= AX (FRON (LIST (LENGTH EJS-W) NEJS-O PEJS FCEJS) CONTRA EJS 2 EXP))  
(LIST AX EJS-W) )  
(PUSH EXP NO-ACEPTADAS)  
NIL )  
(C - SOBRA EXITO EN TODOS EJS --)  
(= TEORIA+E (U EJ TEORIA))  
( (PROOF-EVER (POP AX) AX NIL 'GENERALIZA))  
( (= AX (FRON (LIST (+ (LENGTH NU-EJS) (- 1) (LENGTH EJS-W)) NEJS-O PEJS  
FCEJS) CONTRA-EJS 2 EXP))  
(PROOF-EVER1 AX (CDR NU-EJS) CONTRA-EJS EJS-W TEXP NEJS-O PEJS FCEJS) ) )

(DEFUN UNIFG-ESPECIAL (LAMBDA (P1 P2 LIGAS)  
(UNIG P1 P2 LIGAS) ) )



```

(DEFUN UNGE (LAMBDA (P1 P2 LIGAS A1 A2 A3)
  (C (L 'PRINTH '(UNGE--- P1 P2 LIGAS -----)))
  (= A1 (CAR P1))
  (= A2 (CAR P2))
  ((=? A1 '=)
    ((=? A2 '=)
      (UNGE-== P1 P2 LIGAS) )
    (UNIFG (LIST 'ES-ELE-DE (CADR P1) (CONS 'ALGO (LAST P1))) P2 LIGAS) )
  ((=? A2 '=)
    (UNIFG P1 (LIST 'ES-ELE-DE (CADR P2) (CONS 'ALGO (LAST P2))) LIGAS) )
  ((SUBSET? (LIST A1 A2) '(> <))
    (UNGE->< P1 P2 LIGAS) )
  ((MEMBER A1 '(> <))
    (= A3 (WORD 'I- A1))
    (UNIFG (LIST 'ES-ELE-DE (CADR P1) (CONS A3 (LAST P1))) P2 LIGAS) )
  ((MEMBER A2 '(> <))
    (= A3 (WORD 'I- A2))
    (UNIFG P1 (LIST 'ES-ELE-DE (CADR P2) (CONS A3 (LAST P2))) LIGAS) )
  ((SUBSET? (LIST A1 A2) '(ALGO ALGO-DE-ESTO: TODO TODO-ESTO: C-))
    (UNGE-SETS P1 P2 LIGAS) )
  ((MEMBER A1 '(ALGO ALGO-DE-ESTO:))
    ((NONNULL (= A3 (TO-I (CDR P1))))
      (UNIFG A3 P2 LIGAS) ) )
  ((MEMBER A2 '(ALGO ALGO-DE-ESTO:))
    ((NONNULL (= A3 (TO-I (CDR P2))))
      (UNIFG P1 A3 LIGAS) ) )
  ((SUBSET? (LIST A1 A2) '(I-> I-< I->< I-<>))
    (UNGE-IS P1 P2 LIGAS) )
  ((AND
    (=? A1 'ES-ELE-DE)
    (=? A1 A2)
    (= PILA-UN (U PILA-UN '# ES-ELE-DE)))
    (UNIFGO NIL (CDR P1) NIL (CDR P2) LIGAS) ) ) )

```

```

(DEFUN GENERALIZA (LAMBDA (NFILEJ PR LIGAS F-GRZS)
  (= PIL (CADR PIL))
  ((PROF0-IFAZ T T PIL RESTO LIGAS 'NORMAL))
  ((PROF0-IFAZ T T PIL RESTO LIGAS 'GENERALIZA))
  ((MEMBER (CAR PIL) SET-UN-BREAKS)
    (PROF0-IFAZ2 NIL RESTO LIGAS) )
  (= NFILEJ (- 1))
  (LOOP
    ((GREATERP (ADD1N 'NFILEJ) (LENGTH EJ))
      (PROF0-IFAZ2 F-GRZS RESTO LIGAS) )
    (= F-GRZS (QUADA-F-GRZS P-GRZS (UN-GRZND0 FIL (NTH NFILEJ EJ) LIGAS
      SUDEXP RESTO))) ) ) )

```

```

(DEFUN FRON (LAMBDA (FIRMA CEJS MODO W)
  (C --MODO 1 ES MINIMA DISTINCION ---)
  ((NOT (ACCEPTABLE? W CEJS FIRMA)) NIL)
  (TIME 'T)
  (FRON0 W) ) )

```

```

(DEFUN FRON. (LAMBDA (W INDICE WF DP SGA)

```

```

(C --- W ES FAND REGRESA FAND GRZ ACEPT DE W --)
((GREATERP (TIME) 5000) W)
(LOOP
  (= WP (AF-GRZ MODO W))
  ((NULL WP))
  ((ACCEPTABLE? WP CEJS FIRMA)) )
((NULL WP) W)
(= SHA (INS-SP-F (SELEC-ACEPTABLES FIRMA CEJS (SEPARA WP)) NIL 1))
(LOOP
  ((NULL SHA) OP)
  (= OP (U-FRON OP (FRONO (POP SHA)))) ) ) )

(DEFUN ACEPTABLE? (LAMBDA (W CEJS FIRMA FUMBRAL ACU N NT PE PC NTCEJS)
  (C (L 'PRINTH ' (AC?----- W CEJS FIRMA AC?-----FIN-----)))
  ((NULL W) NIL)
  ((NULL CEJS) W)
  ((IMPLICADA? W NO-ACEPTADAS) NIL)
  (= CEJS (SELEC-RANDOM 10 CEJS))
  (C -- N NT --num. de ejs ya abarcados y de ejs total)
  (C - PE PC = porc de ejs y de cejs exteriores en "10s" de porciento)
  (L= '(N NT PE PC) FIRMA)
  (= FUMBRAL (* (QUOTIENT (* 10 N) NT) PE))
  (= ACU 0)
  (= NTCEJS (LENGTH CEJS))
  (LOOP
    ((NULL CEJS) W)
    ( ((PROOF W (POP CEJS))
      (ADDINA 'ACU) ) )
    ((NOT (> FUMBRAL (* (QUOTIENT (* 10 ACU) NTCEJS) PC))) NIL) ) ) )

(DEFUN PFALES-2# (LAMBDA NIL
  '(F-OR-TODO F-OR-TODO PROOF-EVER PROOF-EVER1 UNIFG-ESPECIAL UNGE
  GENERALIZA FRON FRONO ACEPTABLE? PFALES-2#) ))

(RDS)

```

## USO DEL PROGRAMA , ILUSTRACIONES

Las ilustraciones se dividen en dos partes: las de PROOF-EVER y las de F-OR-TODO. En las ilustraciones de PROOF-EVER, si no se indica explícitamente lo contrario, es vacío el conjunto de CONTRA EJEMPLOS o el de TEORÍA. Por otro lado, el tiempo indicado en cada ilustración ha sido obtenido en una IBM-PC compatible con 256k a 4.77 Mhz con microprocesador 8088.

## INDICE DE ILUSTRACIONES DEL USO DE "PROOF-EVER"

Ilust. #	ILUSTRACION:	
1	DIFTES cierto.....	44
2	DIFTES falso.....	44
3	Generalización por cambio de símbolo variable...	44
4	Uso de predicados conmutativos.....	45
MOTOR	.....	45
5	Generalización por cambio de símbolo constante a variable.....	46
6	Back-tracking.....	47
7	Necesidad de varias vías de generalización.....	47
8	Similar a # 7 .....	47
9	Estilo los "trenes" de Michalski.....	49
10	Clases , Contra-Clases en ejemplos estilo trenes	49
12	Aceptación por frecuencia inferior en la contra-clase.....	49
13	Complementario de 12.....	50
14	Frecuencia en clases , contra-clases.....	51
15	Cambio de ámbito prohibido.....	51
16	Generalizaciones especiales tipo numérico.....	52
17	Generalizaciones especiales tipo conjuntos.....	52
20	Uso de cláusulas con antecedente "and" & "or" en una teoría de Fondo.....	54

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : 1

-----NOTA-----:

-- AQUI TRATAÑOS DE VER QUE EN EFECTO SI SE CUMPLE LA EXPRESION

-- -- LA CLASE ES :

EJ NUM 1 := ( AND (A a) (A b) (B b) )

EJ NUM 2 := ( AND (B d) (A d) (A e) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (A x) (B y) (DIFTES x y) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (A x) (B y) (DIFTES x y) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :

EJS-W = ( 1 2)

----- EL TIEMPO TRANSCURRIDO ES: 5 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : 2

-----NOTA-----:

-- AQUI NO SE DEBE CUMPLIR LA EXPR EN EJ 1 --

-- -- LA CLASE ES :

EJ NUM 1 := ( AND (A a) (B a) )

EJ NUM 2 := ( AND (A d) (B e) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (A x) (B y) (DIFTES x y) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (A x) (B y) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :

EJS-W = ( 1 2)

----- EL TIEMPO TRANSCURRIDO ES: 6 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : 3

-----NOTA-----:

--AQUI DEBE ELIMINARSE:(C x) PUES NO ES VALIDA :(C a)

-- -- -- LA CLASE ES :  
 EJ NUM 1 := ( AND (A a) (B a) (A b) (C b) (B c) (C c)  
 )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (A x) (B x) (C x) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :  
 ( AND (A x) (B x) (C xUG) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :  
 EJS-W = (1)

----- EL TIEMPO TRANSCURRIDO ES: 6 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : 4

-----NOTA-----:  
 --OBSERVESE COMO SE MANEJA A LAS VARIABLES

-- -- -- LA CLASE ES :  
 EJ NUM 1 := ( AND (C d h) (C c h) (ã d (B c (D h) a)) (D c  
 )  
 (D c a) )

EJ NUM 2 := ( AND (D b a) (A c (B d (D h) a)) (D c a) (C c  
 )  
 )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (C z h) (D x a) (A x (B z a (D h))) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :  
 ( AND (C z h) (D x a) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :  
 EJS-W = (1 2)

----- EL TIEMPO TRANSCURRIDO ES: 6 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : NOTOR

-----NOTA-----:  
 --- TRATARA DE ENCONTRAR DEF DE NOTOR ---

-- -- - LA CLASE ES :

EJ NUM 1 := ( AND (TIENE M Bobinas) (FUENTE M Voltaje)  
 (TIENE M Rotor) (TIENE M Armadura) (CONVIERTE x y)  
 (ES x ENERGIA Electrica) (ES y ENERGIA Mecanica) )

EJ NUM. 2 := ( AND (TIENE M Pistones) (FUENTE M Gasolina)  
 (TIENE M Bujias) (CONVIERTE x y) (ES x ENERGIA Quimica)  
 (ES x ENERGIA Mecanica) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (TIENE M x) (TIENE M y) (DIFTES x y)  
 (FUENTE M z) (TIENE M y2) (DIFTES x y y2) (CONVIERTE u v)  
 (ES u ENERGIA w) (ES v ENERGIA Mecanica) (DIFTES w Mecanica) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (TIENE M x) (TIENE M y) (FUENTE M z)  
 (TIENE M y2) (CONVIERTE u v) (ES u ENERGIA w)  
 (ES v ENERGIA Mecanica) (DIFTES w Mecanica) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :

EJS-W = ( 2 )

----- EL TIEMPO TRANSCURRIDO ES: 33 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER - - NUMERO O IDENTIF : 5

-----NOTA-----;

-- DEBE CAMBIAR CONS POR VARS Y TENER FINALMENTE: (A x)(B y)---

-- -- - LA CLASE ES :

EJ NUM 1 := ( AND (A a) (B a) )  
 EJ NUM 2 := ( AND (A b) (B c) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (A a) (B a) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (A xakb) (B xaun) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :

EJS-W = ( 2 )

----- EL TIEMPO TRANSCURRIDO ES: 5 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : 6

-----NOTA-----:

--- AQUI DEBE ENCONTRAR :  $A \times B \times$

-- - - - LA CLASE ES :

EJ NUM 1 := ( AND (A a) (A b) (B b) )

EJ NUM 2 := ( AND (B d) (A d) (A e) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (A a) (A b) (B b) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (A xaPA) (A xaPA) (B xaPA) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :

EJS-W = ( 1 2 )

----- EL TIEMPO TRANSCURRIDO ES: 6 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : 7

-----NOTA-----:

ES NECESARIO ENSAYAR VARIAS VIAS DE GENERALIZACION ?

ESTA ILUSTRACION NOS MUESTRA QUE ESTE ES EL CASO :

-- - - - LA CLASE ES :

EJ NUM 1 := ( AND (A c c c) )

EJ NUM 2 := ( AND (A a a b) (A b a a) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (A c c c) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (A xcra xcra xcra) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :

EJS-W = ( 1 2 )

----- EL TIEMPO TRANSCURRIDO ES: 5 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : 8

-----NOTA-----:

-- TRATAMOS DE VER SI PUEDE SELECCIONAR LAS PILS DE MENOR GENERALIZACION ---

-- - - - LA CLASE ES :

EJ NUM 1 := ( AND (V 1 (C- CH A)) (V 2 (C- CH T)) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (V 4 (C- CH T)) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (V x4Ew (C- CH T)) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON  
EJS-W = (1)

----- EL TIEMPO TRANSCURRIDO ES: 5 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : 9

-----NOTA-----:

-- ESTA ILUSTRACION E ACERCA DE CONVOYES DE TRENES

-- V SIGNIFICA VAGON C- ES NOTACION PARA CONJUNTO DE PROPIEDADES

-- REDO ES REDONDO RO = ROJO GR = GRANDE CUA = CUADRADO ETC

--- QUEREMOS VER SI ATACA ESTE PROBLEMA CON EFICIENCIA

-- - - - LA CLASE ES :

EJ NUM 1 := ( AND  
(V 1 (C- REDO GR ROJO 3-RUEDAS))  
(V 2 (C- CUA CH T BLANCO 2-RUEDAS))  
(V 3 (C- REDO AZUL 3-RUEDAS GR)) )

EJ NUM 2 := ( AND (V 1 (C- AZUL 2-RUEDAS GR)) (V 2 (C- ROJO  
)  
(V 3 (C- BLANCO REDO 3-RUEDAS GR)) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND  
(V 1 (C- REDO GR ROJO 3-RUEDAS))  
(V 2 (C- CUA CH T BLANCO 2-RUEDAS))  
(V 3 (C- REDO AZUL 3-RUEDAS GR)) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (V x1dQ (C- REDO GR 3-RUEDAS)) (V 2 (C- CH T))  
(V 3 (C- REDO 3-RUEDAS GR)) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON  
EJS-W = (1 2)

----- EL TIEMPO TRANSCURRIDO ES: 27 /60 SEGUNDOS -----



```

*****
---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : 10

-----NOTA-----:
-- LA NOTACION ES COMO EN LA ILUSTRACION 9
-- EN ESTE CASO QUEREMOS HACER OBSERVAR COMO MANEJA PROOF-EVER
--EL CASO EN QUE TENEMOS CONTRA-EJEMPLOS NO NIL
-- NOS DEBE DAR UNA EXPRESION O FORMULA NO VALIDA O CASI-NO-VALIDA EN
CONTAREJEMPLOS

-- - - - LA CLASE ES :
EJ NUM 1 := ( AND
  (V 1 (C- REDO GR ROJO 3-RUEDAS))
  (V 2 (C- CUA CH T BLANCO 2-RUEDAS))
  (V 3 (C- REDO AZUL 3-RUEDAS GR)) )

EJ NUM 2 := ( AND (V 1 (C- AZUL 2-RUEDAS GR)) (V 2 (C- ROJO CH
))
  (V 3 (C- BLANCO REDO 3-RUEDAS GR)) )

----- LOS CONTRA EJEMPLOS SON :
EJ NUM 1 := ( AND (V 1 (C- REDO GR 2-RUEDAS))
  (V 2 (C- CUA ROJO 3-RUEDAS)) (V 3 (C- GR)) (V 4 (C- BLANCO CH
2-RUEDAS))
)

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND
  (V 1 (C- REDO GR ROJO 3-RUEDAS))
  (V 2 (C- CUA CH T BLANCO 2-RUEDAS))
  (V 3 (C- REDO AZUL 3-RUEDAS GR)) )

-----RESULTADOS-----
----- LA DEFINICION ENCONTRADA ES :
( AND (V 2 (C- CH T)) (V 3 (C- REDO 3-RUEDAS)) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ADARCADOS POR LA ANTERIOR FORMULA SON :
EJS-W = (1 2)
----- EL TIEMPO TRANSCURRIDO ES: 138 /60 SEGUNDOS -----

*****
---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : 12

-----NOTA-----:
-- En esta ilustracion vemos el efecto de la seleccion
-- de una formula basada en las FRECUENCIAS
-- De no usarse este algoritmo no se podria tener (A x) como
-- resultado

```

--Obsérvese que A x es valida con frecuencias 80/100 y 60/100  
 -- en los ejemplos y contra-ejrs respectivamente

-- - - - LA CLASE ES :

EJ NUM 1 := ( AND (A a) )  
 EJ NUM 2 := ( AND (A b) )  
 EJ NUM 3 := ( AND (A c) )  
 EJ NUM 4 := ( AND (A d) )  
 EJ NUM 5 := ( AND (B b) )

----- LOS CONTRA EJEMPLOS SON :

EJ NUM 1 := ( AND (A a) )  
 EJ NUM 2 := ( AND (A b) )  
 EJ NUM 3 := ( AND (A 3) )  
 EJ NUM 4 := ( AND (C 1) )  
 EJ NUM 5 := ( AND (C 2) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (A a) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (A xaeb) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :  
 EJS-W = ( 1 2 3 4 )

----- EL TIEMPO TRANSCURRIDO ES: 22 /60 SEGUNDOS -----

#####

---- ILUSTRACION DEL USO DE PROOF-EVER -- NUMERO O IDENTIF : 13

-----NOTA-----:

-- Aqui invertimos los papeles de clase-1 y 2 respecto  
 -- la ilustracion # 12

-- - - - LA CLASE ES :

EJ NUM 1 := ( AND (A a) )  
 EJ NUM 2 := ( AND (A b) )  
 EJ NUM 3 := ( AND (A 3) )  
 EJ NUM 4 := ( AND (C 1) )  
 EJ NUM 5 := ( AND (C 2) )

----- LOS CONTRA EJEMPLOS SON :

EJ NUM 1 := ( AND (A a) )  
 EJ NUM 2 := ( AND (A b) )  
 EJ NUM 3 := ( AND (A c) )  
 EJ NUM 4 := ( AND (A d) )  
 EJ NUM 5 := ( AND (B b) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (A a) )

## -----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :

EJS-W = NIL

----- EL TIEMPO TRANSCURRIDO ES: 368 /60 SEGUNDOS -----

\*\*\*\*\*

----- ILUSTRACION DEL USO DE PROOF-EVER - - NUMERO O IDENTIF : 14

-----NOTA-----:

-- Debido a contar con un criterio estadistico

-- aqui se debe poder obtener como definicion de clase-1 a :

----- B x -----

-- - - - LA CLASE ES :

EJ NUM 1 := ( AND (A a) (B 1) )

EJ NUM 2 := ( AND (A b) (B 2) )

EJ NUM 3 := ( AND (A 3) (B 3) )

EJ NUM 4 := ( AND (C 1) )

EJ NUM 5 := ( AND (C 2) )

----- LOS CONTRA EJEMPLOS SON :

EJ NUM 1 := ( AND (A a) )

EJ NUM 2 := ( AND (A b) )

EJ NUM 3 := ( AND (A c) )

EJ NUM 4 := ( AND (A d) )

EJ NUM 5 := ( AND (B b) )

----- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (A a) (B 1) )

## -----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (B x1VF) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :

EJS-W = (1 2 3)

----- EL TIEMPO TRANSCURRIDO ES: 22 /60 SEGUNDOS -----

\*\*\*\*\*

----- ILUSTRACION DEL USO DE PROOF-EVER - - NUMERO O IDENTIF : 15

-----NOTA-----:

--Aquí deseamos ilustrar el rechazo a una disminucion drastica

-- del tamaño cuando ya se ha filtrado a la expresion por muchos ejes

-- - - - LA CLASE ES :

```

EJ NUM 1 := ( AND (A x) (B x) )
EJ NUM 2 := ( AND (A x) (B x) )
EJ NUM 3 := ( AND (A x) (B x) )
EJ NUM 4 := ( AND (A x) (B x) )
EJ NUM 5 := ( AND (A x) (B x) )
EJ NUM 6 := ( AND (A x) (B x) )
EJ NUM 7 := ( AND (A x) (B x) )
EJ NUM 8 := ( AND (A x) (B x) )
EJ NUM 9 := ( AND (A x) (B x) )
EJ NUM 10 := ( AND (A x) (B x) )
EJ NUM 11 := ( AND (A x) )
    
```

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

```
( AND (A x) (B x) )
```

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

```
( AND (A x) (B x) )
```

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :

EJS-W = ( 1 2 3 4 5 6 7 8 9 10)

----- EL TIEMPO TRANSCURRIDO ES: 5 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER - - NUMERO O IDENTIF : 16

-----NOTA-----:

-- AQUI VENOS EL USO DE GENERALIZACIONES DE TIPO NUMERICO

-- Y DE TIPO DE CONJUNTOS Y DE INTERVALOS

-- - - LA CLASE ES :

```

EJ NUM 1 := ( AND (> (TIO JUAN) 30) (= (EDAD JUAN) 20)
              (= (PAIS JUAN) MEXICO)
              (ES-ELE-DE (ALTURA JUAN) (I->< 160 170)) )
    
```

```

EJ NUM 2 := ( AND (> (ABUELO PACO) 100) (> (ALTURA PACO) 165)
              (< (TIO PACO) 40)
              (ES-ELE-DE (PAIS PACO) (ALGO-DE-ESTO: USA JAPON)) (= (EDAD PACO) 30)
              )
    
```

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

```
( AND (> (TIO JUAN) 30) (= (EDAD JUAN) 20)
      (= (PAIS JUAN) MEXICO)
      (ES-ELE-DE (ALTURA JUAN) (I->< 160 170)) )
```

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND

(ES-ELE-DE (EDAD xJUANWH) (ALGO-DE-ESTO: 20 30))  
 (ES-ELE-DE (PAIS xJUANWH) (ALGO MEXICO USA JAPON))  
 (ES-ELE-DE (ALTURA xJUANWH) (I-> 160)) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :  
 EJS-W = (1 2)

----- EL TIEMPO TRANSCURRIDO ES: 10 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER - - NUMERO O IDENTIF : 17

-----NOTA-----:

-- VEREMOS ALGUNAS REGLAS DE GENERALIZACION QUE USAN NOCIONES DE CONJUNTOS

-- - - - LA CLASE ES :

EJ NUM 1 := ( AND  
 (SATISFACE FLOR-2 (ALGO-DE-ESTO: 3-PETALOS ROJA AROMATICA))  
 (CUMPLE FLOR-3 (TODO-ESTO: 100 RAICES-AEREAS VOLUMINOSA ESPINOSA))  
 )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND

(SATISFACE FLOR-4 (ALGO-DE-ESTO: 3-PETALOS AMARILLA))  
 (CUMPLE FLOR-5 (TODO-ESTO: RAICES-AEREAS ACEITOSA)) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND

(SATISFACE xFLOR-4)c (ALGO-DE-ESTO: 3-PETALOS AMARILLA ROJA AROMATICA  
 (CUMPLE xFLOR-5md (TODO-ESTO: RAICES-AEREAS)) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :  
 EJS-W = (1)

----- EL TIEMPO TRANSCURRIDO ES: 5 /60 SEGUNDOS -----

\*\*\*\*\*

---- ILUSTRACION DEL USO DE PROOF-EVER - - NUMERO O IDENTIF : 18

-----NOTA-----:

---ES SIMILAR A LA ILUSTRACION # 7---

---ES NECESARIO ENSAYAR VARIAS VIAS DE GENERALIZACION ?

---ESTA ILUSTRACION NOS MUESTRA QUE ESTE ES EL CASO :

--- VENOS QUE DE ESTA MANERA SE PUEDE TOMAR EN CUENTA QUE EN

-- LOS CONTRA-EJS ES VALIDA LA FORMULA: ( A x x y )

-- Y ENTONCES SE ENSAYA LA FORMULA: ( A x y y )

-- LO CUAL FUE POSIBLE GRACIAS A UN "BACK-TRACKING "

-- - - - LA CLASE ES :

EJ NUM 1 := ( AND (A c c c) )  
 EJ NUM 2 := ( AND (A a a b) (A d f f) )

----- LOS CONTRA EJEMPLOS SON :

EJ NUM 1 := ( AND (A m m k) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (A c c c) )

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (A xccS xcoe xcoe) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :  
 EJS-W = (1 2)

----- EL TIEMPO TRANSCURRIDO ES: 11 /60 SEGUNDOS -----

----- ILUSTRACION DEL USO DE PROOF-EVER - - NUMERO O IDENTIF : 20

-----NOTA-----:

-- AQUI TRATAMOS DE ILUSTRAR COMO SE PUEDE USAR UNA TEORIA DE  
 -- FONDO

-- - - - LA CLASE ES :

EJ NUM 1 := ( AND (A b) )  
 EJ NUM 2 := ( AND (C c) (C d) (D d) )  
 EJ NUM 3 := ( AND (F a) )  
 EJ NUM 4 := ( AND (G g) )

---- LA EXPRESION DE ENTRADA A PROOF EVER ES :

( AND (A a) (B b) )

---- LA TEORIA DE FONDO ES :

((A x) (AND (C x) (D x))) ((A x) (OR (F x) (G x)))

-----RESULTADOS-----

----- LA DEFINICION ENCONTRADA ES :

( AND (A x&Ev) )

-- LOS NUMEROS DE EJS DE LA CLASE-1 ABARCADOS POR LA ANTERIOR FORMULA SON :  
 EJS-W = (1 2 3 4)

----- EL TIEMPO TRANSCURRIDO ES: 6 /60 SEGUNDOS -----

**ADICE DE ILUSTRACIONES DEL USO DE "F-OR-TODO"**

# Ilustración	SE OBTIENE
1	F-OR de friend's
2	F-OR de friend's
3	fórmula tipo m-de-n
4	F-OR de m-de-n's
5	tipo m-de-n

**NOTA PARA LEER LAS ILUSTRACIONES DE "F-OR-TODO":**

En cada una de las ilustraciones de la función principal, "F-OR-TODO", hemos usado como auxiliar para generar a la clase de ejemplos a una matriz que hemos llamado "matriz-patrón".

La clase se genera así a partir de la matriz-patrón:

- \* La columna  $j$  se hace corresponder al PREDICADO  $A_j$
- \* El renglón  $i$  corresponde al ejemplo  $i$ .
- \* Un "1" en la columna  $j$  renglón  $i$  afirma que en el ejemplo  $i$  la proposición  $A_j$  es verdadera.
- \* Un "0" en la columna  $j$  renglón  $i$  afirma que en el ejemplo  $i$  la proposición  $A_j$  es falsa.
- \* Un "1" en la columna  $j$  renglón  $i$  afirma que en el ejemplo  $i$  la proposición  $A_j$  es verdadera.
- \* Un "0" en la columna  $j$  renglón  $i$  afirma que en el ejemplo  $i$  la proposición  $A_j$  es falsa.

**ILUSTRACIONES DE "F-OR-TODO"**

-----  
 --- LA MATRIZ PATRON DE LA CUAL SE VAN A GENERAR LOS EJEMPLOS  
 -DE LA CLASE-I ES:

```
(1 0 1 0)
(0 1 0 1)
(1 0 1 0)
(0 1 0 1)
```

```
-----
--- LA CLASE-I ES:
(1 0 1 0)
(0 1 0 1)
(1 0 1 0)
(0 1 0 1)
```

((A1 x) (A3 x))  
 ((A2 x) (A4 x))

---- LOS CONTRA-EJEMPLOS SON:

--- SE APLICARÁ LA FUNCIÓN F POR TODO  
 EN EL CASO IDEAL SE DEBERÍA RECUPERAR LA MATRIZ PATRON

NIL

----- EL RESULTADO ES: -----

(W1 VALIDAS-EN (1 3) 0)  
 DONDE-LAS-WS-SON:  
 (W1= ((A1 x) (A2 x)))

(W1 VALIDAS-EN (2 4) 0)  
 DONDE-LAS-WS-SON:  
 (W1= ((A2 x) (A3 x)))

----- EL TIEMPO TRANSCURRIDO ES: 16 700 SEGUNDOS -----

\*\*\*\*\*

---- LA MATRIZ PATRON DE LA CUAL SE VAN A GENERAR LOS EJEMPLOS  
 --DE LA CLASE-1 ES :

(1 1 0 0 0)  
 (0 0 1 1 1)  
 (1 1 0 0 0)  
 (0 0 1 0 1)  
 (0 0 1 1 0)  
 (0 0 0 1 1)

----- EN (1 3 3 4) -----

----ASI PUES TENEMOS :  
 ---- LA CLASE 1 ES :  
 ((A1 x) (A2 x))  
 ((A3 x) (A4 x) (A5 x))  
 ((A1 x) (A2 x))  
 ((A3 x) (A5 x))  
 ((A3 x) (A4 x))  
 ((A4 x) (A5 x))

---- LOS CONTRA-EJEMPLOS SON:

--- SE APLICARÁ LA FUNCIÓN F POR TODO  
 EN EL CASO IDEAL SE DEBERÍA RECUPERAR LA MATRIZ PATRON

NIL

----- EL RESULTADO ES: -----

(W1 VALIDAS-EN (1 3) 0)  
 DONDE-LAS-WS-SON:  
 (W1= ((A1 x) (A2 x)))

(2 DE W1 W2 W3 VALIDAS-EN (2 4 5 4) 0)  
 DONDE-LAS-WS-SON:  
 (W1= ((A3 x))),  
 (W2= ((A5 x)))



W3= ((A4 x)))

----- EL TIEMPO TRANSCURRIDO ES: 27 /60 SEGUNDOS -----

\*\*\*\*\*

---- LA MATRIZ PATRON DE LA CUAL SE VAN A GENERAR LOS EJEMPLOS  
 --DE LA CLASE-1 ES :

(1 1 1 0)  
 (1 1 0 1)  
 (1 0 1 1)  
 (0 1 1 1)

----ASI PUES TENEMOS :

---- LA CLASE 1 ES :

((A1 x) (A2 x) (A7 x))  
 ((A1 x) (A2 x) (A4 x))  
 ((A1 x) (A3 x) (A4 x))  
 ((A2 x) (A3 x) (A1 x))

---- LOS CONTRA-EJEMPLOS SON:

--- -- SE APLICAR LA FUNCION : F-OR-YODO

--- EN EL CASO IDEAL SE DEBERIA RECUPERAR LA MATRIZ PATRON

NIL

----- EL RESULTADO ES: -----

(3 DE W1 W2 W3 W4 VALIDAS-EN (1 2 3 4) )  
 DONDE-LAS-WG SON:  
 W1= ((A4 x)))  
 W2= ((A3 x)))  
 W3= ((A2 x)))  
 W4= ((A1 x)))

----- EL TIEMPO TRANSCURRIDO ES: 29 /60 SEGUNDOS -----

\*\*\*\*\*

---- LA MATRIZ PATRON DE LA CUAL SE VAN A GENERAR LOS EJEMPLOS  
 --DE LA CLASE-1 ES :

(0 0 0 1 1)  
 (1 1 0 0 0)  
 (0 1 1 0 0)  
 (0 0 0 1 1)  
 (1 0 1 0 0)

----ASI PUES TENEMOS :

---- LA CLASE 1 ES :

((A7 x) (A5 x))  
 ((A1 x) (A2 x))  
 ((A2 x) (A3 x))  
 ((A4 x) (A5 x))  
 ((A1 x) (A3 x))

---- LOS CONTRA-EJEMPLOS SON:

--- SE APLICARA LA FUNCION : F-OR-TODO  
 --- EN EL CASO IDEAL SE DEBERIA RECUPERAR LA MATRIZ PATRON

NIL

----- EL RESULTADO ES: -----

(W1 VALIDAS EN (1 4) 0)  
 DONDE-LAS-WS-SON:  
 (W1= ((A1 1) (A2 1))

(2 DE W1 W2 W3 VALIDAS EN (2 7 5) 0)  
 DONDE-LAS-WS-SON:  
 (W1= ((A3 1))  
 (W2= ((A1 1))  
 (W3= ((A3 1))

----- EL TIEMPO TRANSCURRIDO ES: 22 /60 SEGUNDOS -----

\*\*\*\*\*

--- LA MATRIZ PATRON DE LA CUAL SE VAN A GENERAR LOS EJEMPLOS  
 --DE LA CLASE-1 ES :

(0 1 1 1)  
 (1 1 0 1)  
 (1 0 1 1)  
 (1 1 1 0)

---ASI PUES TENEMOS :

--- LA CLASE 1 ES :

((A2 x) (A3 x) (A4 x))  
 ((A1 x) (A2 1) (A4 x))  
 ((A1 x) (A3 x) (A4 x))  
 ((A1 x) (A2 x) (A3 x))

--- LOS CONTRA-EJEMPLOS SON:

--- SE APLICARA LA FUNCION : F-OR-TODO

--- EN EL CASO IDEAL SE DEBERIA RECUPERAR LA MATRIZ PATRON


NIL

----- EL RESULTADO ES: -----

(3 DE W1 W2 W3 W4 VALIDAS EN (1 0 0 1) 0)  
 DONDE-LAS-WS-SON:  
 (W1= ((A1 x))  
 (W2= ((A3 1))  
 (W3= ((A2 1))  
 (W4= ((A4 x))

----- EL TIEMPO TRANSCURRIDO ES: 27 /60 SEGUNDOS -----

El jurado designado por la Sección de Computación del Departamento -  
de Ingeniería Eléctrica del Centro de Investigación y de Estudios -  
Avanzados del IPN., aprobó esta tesis el 25 de julio de 1988.



---

Dr. Josef Kolar Sabor.



---

Dr. Renato Barrera Rivera.



---

Dr. Guillermo Morales Luna.

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL  
INSTITUTO POLITECNICO NACIONAL

**BIBLIOTECA DE INGENIERIA ELECTRICA**  
FECHA DE DEVOLUCION

El lector está obligado a devolver este libro  
antes del vencimiento de préstamo señalado  
por el último sello.

- 3 MAR. 1989

DEVOLUCION



