CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS

DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION COMPUTACION

"ESQUELETO DE UN SISTEMA EXPERTO EN PROLOG"

TESIS QUE PRESENTA:

JUAN JOSE FLORES ROMERO

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS

ESPECIALIDAD EN INGENIERIA ELECTRICA

MEXICO

Febrero de 1986.

A MIS GINAS:

AGRADECIMIENTOS.

Detrás de esta tesis, no solo están un gran número de noches de desvelo. El mayor esfuerzo lo realizamos juntos con mi esposa y mi hija al estar separados todo este tiempo. A ellas les agradezco toda su paciencia, comprensión y el apoyo moral que siempre me brindaron.

La ayuda que por parte de mi familia he recibido ha sido de gran valor. Mis Padres y mi Tía Prieta siempre me alentaron a seguir hasta el final, a superar los obstáculos.

Al llegar a México, encontré a un buen amigo: Manuel González. A él le agradezco todo el tiempo que dedicó a mi proyecto, sus consejos y su amistad. Otra persona que me brindó su ayuda desinteresada en los momentos que más la necesitaba fue Sergio Chapa. A ellos dos: Mil gracias.

A mis profesores agradesco sus enseñanzas y la ayuda que mediante su experiencia me brindaron. Gracias Dr. Adolfo Guzmán Arenas, Dr. Felipe de J. Contla y Dr. Pablo Noriega.

Especialmente quiero hacer un reconocimiento al Dr. Felipe de J. Contla por su interés que mostro en la revisión y critica que hizo a este trabajo.

Esta tesis fue realizada gracias al apoyo económico que recibí del CONACYT y del CINVESTAV.

RESUMEN.

Los Sistemas Expertos son programas de computadora que emulan el conocimiento humano. Este tipo de programas es aplicable a la solución de problemas que no siguen un modelo matemático exacto. Esto es a problemas que no tienen un algoritmo que describa paso a paso el proceso de solución.

En esta tesis, presento el desarrollo de un esqueleto para el desarrollo de Sistemas Expertos. Este esqueleto (de nombre MYPRO) se puede vestir de cualquier base de conocimientos en algún área del saber humano. De este modo se pueden desarrollar Sistemas Expertos en diferentes áreas.

Este esqueleto, tiene varias características deseables para un esqueleto de Sistemas Expertos. Puede recibir conocimiento de casi cualquier área del saber. También puede en un momento dado explicar como fue que llegó a cierta conclusión. Esto es, mostrar las líneas de razonamiento que siguió para alcanzar una meta. Además tiene la capacidad de trabajar con incertidumbre en las respuestas del usuario y en el conocimiento de la base.

El esqueleto fué desarrollado en el lenguaje de programación PROLOG. Este lenguaje ofrece algunas ventajas para el desarrollo de Sistemas Expertos. PROLOG consta de dos algoritmos que nos ayudan para el desarrollo de Sistemas Expertos: Uno es un algoritmo de casamiento de patrones y el otro es el algoritmo de resolución.

INDICE

CONTENIDO PAG.

CONTENIDO PAG.

CAPITULO 1.

1. INTRODUCCION.

Durante los últimos años en las áreas de Inteligencia Artificial se han obtenido importantes resultados. Una de estas áreas versa sobre la programación de sistemas conocidos como **Sistemas Expertos (SE)** o Sistemas Basados en Conocimiento. Estos programas son diseñados para representar y aplicar conocimiento de hechos o experiencias de áreas específicas. Estos SE emplean conocimiento mecanizado a resolver problemas que ordinariamente requieren de una acción inteligente o humana. Sin embargo, estos sistemas se representan y aplican electrónicamente.

En el área de SE se investigan métodos y técnicas para construir sistemas hombre-máquina que hacen uso de la experiencia en la solución de problemas especializados. La experiencia consiste del conocimiento de problemas y de la habilidad para resolver algunos de ellos.

Usualmente se pueden considerar dos tipos de conocimiento: público y privado. El conocimiento público incluye definiciones, hechos y teorías que comunmente contienen los libros, ademas de referencias en el dominio de estudios. El conocimiento privado consiste de una gran cantidad de reglas muy simples y es a lo que se llama heurística. Consecuentemente la experiencia incluye

no solamente conocimiento pûblico, sino ademàs, conocimiento privado.

Recientemente los SE han mostrado creciente importancia tanto econômica como cientifica. Esto se debe por un lado a la demanda en la consulta de expertos humanos y a la reducciôn de costos por medio de la mecanizaciôn de SE. Por otro lado, a los logros y avances obtenidos con ellos.

El êxito de las tècnicas de SE es su capacidad de comunicaciôn. Los SE se comunican con Ingenieros de Conocimiento, con Expertos, con Usuarios, con Bases de Datos y con otros sistemas de computadoras, justamente como los humanos lo hacen e interactûan con todas esas fuentes. Esto es, el êxito de un SE estriba en que se comunica con cada uno de estos sistemas o personas en su propio lenguaje.

Los SE se comunican con los Ingenieros de Conocimiento a travês de una estructura de editores, que les permiten accesar y modificar fàcilmente componentes de la base de conocimiento. Los SE se comunican con los expertos a travês de diâlogos simples con explicaciones que elucidan sus lîneas de razonamiento y que dirigen a los expertos hacia el lugar donde se deberàn hacer cambios en la base de conocimientos.

Para el caso de los usuarios, los SE pueden transformar su lenguaje en lenguaje natural para generar preguntas y respuestas para interpretar las respuestas de los usuarios. Finalmente los

SE también pueden interactuar con otros sistemas de cómputo.

Por tanto el objetivo principal de esta tesis es el desarrollo de SE que permita mecanizar las actividades de un experto y la representación de campos del saber en forma dócil para la manipulación mecánica.

Este trabajo se enfoca a la fabricación del esqueleto de un SE, que he llamado MYPRO. A este esqueleto se le considera del tipo EMYCIN [1] porque utiliza los modelos de manejo de incertidumbre propuestos por Shortliffe [2]. El esqueleto tiene una estructura muy flexible, así que es relativamente fácil aproximarse a las formas o estructuras de otros esqueletos. El nombre de MYPRO se debe a que el diseño se baso fundamentalmente en el tipo de arquitectura de MYCIN. De tal forma, que MYPRO significa MYCIN en PROLOG.

Básicamente el trabajo consta de varios módulos que componen el esqueleto. Un módulo maneja las reglas para efectuar inferencias. Otro módulo se encarga de la propagación de la incertidumbre en la red de conocimiento. Otro módulo lleva un registro de las reglas empleadas en una consulta, para mostrar las líneas de razonamiento y responder a preguntas COMO? y POR QUE?. Se tienen además algunos módulos adicionales que permiten: listar las reglas de la base de conocimiento, manejar el diálogo de manera conveniente, e incluso guardarlo en disco para consultas posteriores, etc.

Lo importante de este trabajo es que es posible obtener SE específicos al cambiar la base de conocimiento. Dicha base es estructurada a través de reglas y hechos (con o sin incertidumbre), usando el lenguaje PROLOG. Por su naturaleza declarativa este lenguaje de programación nos permite mucha flexibilidad en la fabricación de la base de conocimiento. PROLOG es un lenguaje apropiado para este tipo de proyectos de inteligencia artificial.

Como se sabe, el poder de un Sistema Experto radica en su base de conocimiento, más que en los procesos lógicos y de control con que pueda contar. Sin embargo, los Sistemas Expertos se distinguen por dos cualidades muy significativas:

i) el poder explicativo de un resultado del problema o consejo
ii) la capacidad de razonamiento en condiciones de evidencias inciertas o incompletas.

El esqueleto que presento en esta tesis cuenta con ellas.

El documento está estructurado en 7 capítulos, haciendo énfasis en dos aspectos principales: la importancia de los SE en los capítulos 2 y 3 y la fabricación del esqueleto en el capítulo 5.

El capítulo 4 trata de algunos temas que forman el núcleo básico en la construcción de cualquier SE. En el capítulo 6 se incluyen los resultados obtenidos al usar dos pequeñas bases de

conocimiento para así mostrar la flexibilidad del esqueleto. Finalmente en el capítulo 7 se presentan conclusiones a este trabajo, dando algunas ideas para futuras investigaciones de SE con estructura mas compleja.

En el apéndice A se dan las principales características y comandos del lenguaje PROLOG y en el apéndice B un listado de las bases de conocimiento.

REFERENCIAS.

[1] Van Melle, W., E.H. Shortliffe y B.G Buchanan. 1981 EMYCIN: A domain-independent system that aids in constructing Knowlege base consultation programs. Machine Intelligence, Infotech State of art Report 9, No. 3.

[2] Shortliffe, E.H. 1976. Computer-based medical consultation: MYCIN New York: American Elsevier.

CAPITULO 2.

2. ANTECEDENTES.

En este capítulo hago un bosquejo histórico de los SE precursores al SE MYCIN. De estos SE menciono los problemas que resuelven, el año en que fueron desarrollados y las instituciones o empresas donde fueron hechos.

Se trata también, la importancia económica y científica que han alcanzado los SE. Del punto de vista económico, los SE son programas que resuelven problemas a un costo muy bajo. En el aspecto científico son importantes por dos razones: han impulsado el desarrollo de nuevas técnicas computacionales y han contribuído en la investigación en diferentes áreas del conocimiento.

Cabe mencionar que el conocimiento del humano que lo distingue como experto es la experiencia. Esta experiencia consiste de un gran número de reglas intuitivas, conocidas como heurísticas.

La heurística le da al humano la habilidad de hacer supuestos correctos (cuando sea necesario), para reconocer aproximaciones prometedoras a problemas y tratar efectivamente

con datos incompletos o incorrectos.

2.1. BOSQUEJO HISTORICO.

Los SE tienen su origen en un proyecto de investigación en el área de inteligencia artificial. La investigación estaba dirigida a la producción de sistemas con la idea central de manejar conocimiento. Este proyecto, **DENDRAL [1]** fué iniciado en 1965 con un grupo interdisciplinario encabezado por Edward A. Feigenbaum, Joshua Ledevherg y Carl Djerassi en la Universidad de Stanford. El objetivo de este sistema era el de analizar espectogramas de masa, resonancia nuclear magnética y otra información físico-química, para inferir la posible estructura de un compuesto químico desconocido. Poco después al inicio de los años 70's, se construyó **METADENDRAL [2]**. En este sistema el objetivo era adicionar análisis de conocimiento a DENDRAL, para así poder seleccionar fragmentación de reglas de estructuras orgánicas.

Otro precursor de los SE fue el sistema **SAINT [3]**, elaborado por Slagle en 1961, que culminó en **MACSYMA [4]**, elaborado por Carol Engleman, William Martin y Joel Moses en 1968. Este sistema experto fué desarrollado en MIT para manipulación simbólica. MACSYMA es tan potente que un grar número de estudiosos de las matemáticas quedan sorprendidos con sus resultados. Básicamente

hace diferenciación e integración de expresiones simbólicas. Este sistema tiene una gran demanda por investigadores de física y matemáticas en todo el mundo.

Posteriormente a DENDRAL [1] (en la Universidad de Carnegie Mellon), se desarrollaron los sistemas **HEARSAY** [5] y **HARPY** [6], para el entendimiento de discursos continuos en lenguaje natural. Este SE evolucionó al HEARSAY II (Herman et al 1980), llegando a manejar un vocabulario de 1000 palabras. Las principales características de HEARSAY II incluyen múltiple cooperación de especialistas en "problem-solving" en diferentes niveles de abstracción. Se deben señalar dos proyectos a raíz de HEARSAY II: **AGE** [7] en Stanford hechos por Nii y Aiello en 1979 y **HEARSAY III** [8] en ISI por Balzer, Erman y London en 1980.

Al principio de los años 1970's, dos importantes sistemas hicieron su aparición:

Uno en la Universidad de Carnegie-Mellon en Pittsburgh, llamado **CADUCEUS** [9] escrito por Pople, Myers y Miller en 1975 y posteriormente revisado en 1981 por Pople llamado **INTERNIS-I** [10]. Este sistema consiste de una red semántica grande que relaciona síntomas y enfermedades en medicina interna. En 1982 este sistema contaba con cerca de 100,000 relaciones representando cerca del 85% de todo el conocimiento relevante en esta área.

El otro sistema es **MYCIN** [11]. MYCIN fué desarrollado en la

Universidad de Stanford en 1976 por Shortliffe para el tratamiento y diagnòstico de enfermedades infecciosas en la sangre. Su conocimiento comprende aproximandamente 400 reglas, relacionando posibles condiciones y sus interpretaciones asociadas.

MYCIN es un sistema que aceleròla era de los SE. Una de las razones es que debido al uso de reglas independientes de la forma IF-THEN, propiciò la construcciòn de varios sistemas. Por ejemplo **PROSPECTOR [12]**, escrito por Duda, Gaschning y Hart en 1979 y revisado en 1981 por Duda y Gaschning en SRI.

Prospector usa una representaciòn de conocimiento para relaciones de depòsitos de minerales. Actualmente hay una variedad de bases de conocimiento de diferentes depòsitos que han ayudado al descubrimiento de importantes yacimientos de molibdeno.

Otro resultado de MYCIN [11] fuè el programa **TEIRESIAS [13]**, que asiste en la construcciòn de grandes bases de conocimiento, ayudando a transferir la experiencia de expertos humanos a la base de conocimiento. Este sistema fuè escrito por Davis, Buchanan y Shortliffe en 1977 y revisado por Lenat y Davis en 1980. Su caracterìstica es que permite un diàlogo (en lenguaje natural reducido) entre el experto y TEIRESIAS.

Tambièn **EMYCIN [14]** es un producto de MYCIN, escrito en la Universidad de Stanford por Ivan Melle en 1979. EMYCIN facilitò el desarrollo de aplicaciones relacionadas con diagnòstico. Este sistema es realmente MYCIN, pero sin su base de conocimiento. Se

puede decir que a partir de esta separación de la base de conocimientos de MYCIN se marca otra importante era, la de la construcción de esqueletos de SE.

Uno de los primeros sistemas que surgieron como aplicación de EMYCIN fué PUFF [15], escrito por Freiherr en 1980. Puff es uno de los SE que han trabajado bastante bien con pocas reglas (aproximadamente 250).

En este punto es conveniente hacer una lista de las personas y Universidades que participaron en el desarrollo e investigación de SE en las diferentes áreas de aplicación.

AREAS DE APLICACION	INVESTIGADORES
Prospección mineral.	R. O. Duda et al 1979. Gaschning 1980.
Configuración de computadoras.	McDermott 1980.
Búsqueda de estructuras químicas.	Buchanan & Feigenbaum 1978. Feigenbaum, Buchanan & Lederberg 1971. Buchanan, Feigenbaum y Sutherland 1969 Feigenbaum 1977. Lindsay et al 1980.
Matemáticas Simbólicas.	Martin & Fateman 1971. Slagle J.R. 1961.
Ajedrez.	Wilkins 1979,1980.

Diagnóstico Médico y Terapia.	Clancey 1979. Clancey, Shortliffe & Buchanan 1979. Fagan et al 1979. Kulikowsku 1980. Pauker et al 1976. Pople, Myers & Miller 1975. Shortliffe 1976. Weiss, Kulikowski & Safir 1978. Weiss & Kulikowski 1979.
Análisis Electrónico.	Staullman & Sussman 1977. Sussman 1977.

Para la fabricación comercial de SE se construyó **ROSIE** [16] desarrollado por Fain et al en 1981, 1982 y F. Hayes Roth et al en 1981 para la compañia Rand. ROSIE suministra un sistema de propósito general a partir del cual se pueden construir SE. Este sistema surgió de uno menos poderoso llamado **RITA** [17], escrito por R. Anderson & Gillogly en 1976. Ambos ROSIE y RITA fueron consecuencia del éxito de MYCIN [11] en el estilo de tener representación de conocimiento en reglas. ROSIE extendió sus cualidades e incorporó numerosas facilidades adicionales. Algunas de estas facilidades incluyen técnicas de representación del conocimiento, comunicación interactiva usuario-sistema, programación estilizada en inglés y programación interactiva de medio ambiente. Además ROSIE fué el primer sistema diseñado para soportar una gran clase de aplicaciones de nuevos SE.

PSG es otro lenguaje de sistemas de producción escrito en la Universidad de Carnegie-Mellon [18] por Newell en 1973 y Newell y

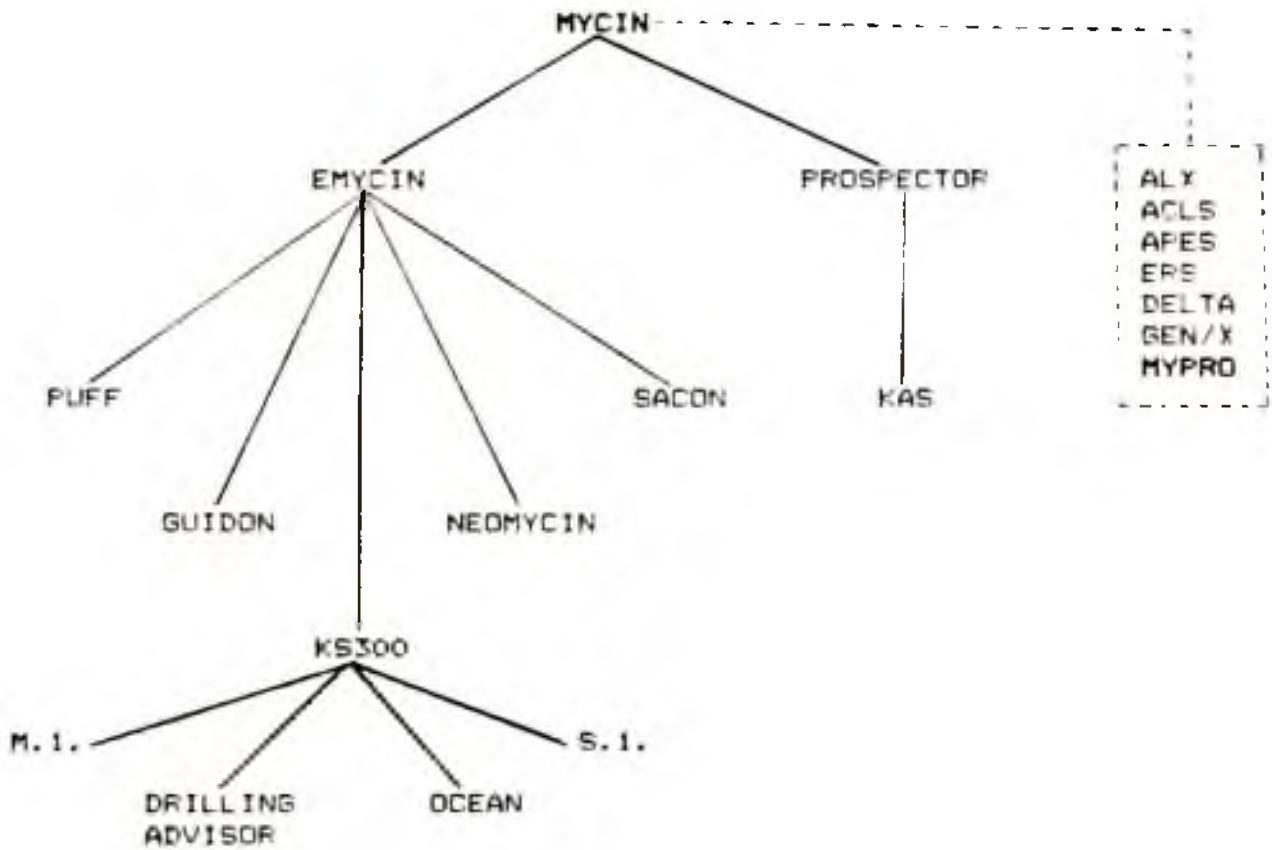
McDermott en 1976. PSG fuè diseñado para el estudio de modelaciòn de conocimiento humano. Este sistema condujo al desarrollo de OPS [19], lenguaje de sistemas de producciòn escrito por Forgy y McDermott en 1977 y Forgy en 1981. Este poderoso lenguaje permitiò escribir R1 [20], actualmente llamado XCON [20]. XCON fuè escrito por McDermott para configurar sistemas de computaciòn DEC/VAX. Este sistema XCON representa la màs notable aplicaciòn del lenguaje OPS.

La figura (1) muestra un àrbol de los Sistemas Expertos que surgieron a partir de MYCIN [11]. Esta figura fue presentada en la Sèptima Conferencia Internacional sobre Reconocimiento de Patrones en Montreal, Canadà por Shortliffe y Buchanan.

2.2. IMPORTANCIA DE LOS SISTEMAS EXPERTOS.

Los sistemas basados en conocimiento, tambièn llamados SE, han tenido recientemente una gran importancia tanto econòmica como cientìfica. La importancia en el aspecto econòmico se debe a la reducciòn de costos en la reproducciòn y consulta de SE. De no tener estos sistemas, se tendrìa que pagar grandes sumas a personal altamente calificado para la soluciòn de problemas complejos. Por ejemplo la claridad de diseño obtenida con los SE XCON [20] para reconfigurar equipo de còmputo DEC/VAX. O bien la

exactitud lograda con PROSPECTOR [12], para determinar yacimientos minerales, han redituado cientos de miles de dólares a los propietarios.



AL/X Advice Language/X
ACLS Analog Concept Learning System
APES A Prolog Expert System
ERS Embedded Rule-based System
DELTA Diesel Electric Locomotive Troubleshooting Aid
GEN-X Generic Expert System
MYPRO MYcin en PROlog.

Fig. 1 La familia MYCIN de los SE.

Desde el punto de vista científico los SE, han contribuído en la elucidación de nuevos compuestos químicos (nuevas drogas), diseño de tarjetas para circuitos impresos, diagnóstico y terapia en medicina (enfermedades pulmonares, de la sangre, de la vista, etc).

Tanto es el interés en los SE que las mejores universidades del mundo se han avocado a su desarrollo, no solo a nivel de software, sino también a nivel de hardware. Por ejemplo, en Caltech se inició un proyecto para construir un circuito integrado que contara con el algoritmo de unificación. Esto facilitaría aún más la tarea de casamiento de patrones y así hacer mas rápido el algoritmo de resolución.

Nos preguntamos ?Qué tan importantes son los Sistemas Expertos que han revolucionado parte de las tendencias tradicionales en computación?. A ese respecto observamos que las mejores Universidades e instituciones de investigación del mundo occidental, tienen un grupo especial trabajando en este tema. Además de las Universidades y de los institutos de investigación, algunos países se encuentran interesados en el desarrollo de este tema de investigación. Como es el caso de Japón, con el proyecto ICOT de la quinta generación y USA con el proyecto MCC. También los países del Mercado Común Europeo participan en el desarrollo de la quinta generación, con el proyecto ESPRIT.

2.3. SUMARIO.

En este capitulo hago una revisiòn de los SE màs importantes que se han desarrollado. Tambièn cito algunos de los SE y sus areas de interès como son: manejo simbòlico de matemàticas, hasta prospecciòn mineral, configuraciòn de sistemas de còmputo, etc.

Los SE tienen gran importancia social, econòmica y cientifica. Varios paises e instituciones han enfocado esfuerzos a la producciòn de sistemas inteligentes. Estos sistemas son auxiliares en la soluciòn de problemas, aportando consejos como verdaderos expertos. Los SE son altamente productivos y fàciles de reproducir a bajo costo.

REFERENCIAS.

[1] Buchanan, B.G., G.L. Sutherland y E. A. Feigenbaum. 1969 Heuristic DENDRAL: Aprogram for generating explanatory hypotheses in organic chemistry. En B. Meltzer y D. Michie, eds., Machine intelligence, vol. 4. Edinburh University Press, pp. 209-254.

[2] Buchanan, B.G. y E.A. Feigenbaum.1978. DENDRAL y MetaDENDRAL: Their applications dimension. Artificial Intelligence 11:5-24.

[3] Slagle, J.R. 1961. A heuristic program that solves symbolic integration problems in freshman calculus. Symbolic Automatic Integrator (SAINT). Ph. D. Diss. Rept. 5G-00001. Lincon Laboratory, Massachusetts Institute of Technology.

[4] Martin, W.A. y R.J. Fateman. 1971. The MACSYMA system. In Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation. Los Angeles, pp. 59-75.

[5] Lesser V. R. y L.D. Erman. 1977. A Retrospective View of the HEARSAY-II. Procedings of the Fifth International Joint Conference on Architecture Artificial Intelligence, pp. 790-800.

[6] Lowerre, B.T. 1976. The HARPY speech recognition system. Ph. D. Diss. Computer Science Departament, Carnegie-Mellon University, Pittsburgh, Pa.

[7] Nii, H.P., y N. Aiello. 1979. AGE (Attempt to Generalize): A knowledge-based program for building knowledge-based programs. En IJCAI 6, pp. 645-655.

[8] Erman, L.D., P.E. London y S.F. Fickas. 1981. The design and an example use of HEARSAY-III. En IJCAI 7, pp. 409-415.

[9] Pople, H.E. Jr. 1977. The formation of composite hypoteses in diagnostic problem solving: An exercise in synthetic reasoning. En IJCAI, pp. 1030-1037.

[10] Miller, R.A. , H.E. Pople y J.D. Myers. 1982. INTERNIST-I, an experimental computer-based diagnostic consultant for general internal medicine. New England Journal of Medicine (Agosto 19): 468-476.

[11] Shortlifle, E.H. 1976. Computer-based Medical Consutation: MYCIN, New York: American Elseivier.

[12] Duda, R.O.1980. Th PROSPECTOR system for mineral explotation. Final Rept:, SRI Project 8172, Artificial Intelligence Center, SRI International, Menlo Park, Calif.

[13] Davis, R. y B.G. Buchanan.1977. Meta-level knowledge: Overview and applications. IJCAI 5:920-928.

[14] van Melle, W., E.H. Shortliffe y B.G. Buchanan. 1981. EMYCIN: A domain-independent system that aids in constructing knowledge based consultation programs. Machine Intelligence, Infotech State of the Art Report 9, no. 3.

[15] Feigenbaum. E.A. 1977. The art of artificial intelligence: Themes and case studies of knowledge engeneering, En IJCAI 5. pp. 1014-1029.

[16] Fain, J., D., Gorlin, F. Hayes-Roth, S.J. Rosenschein, H. Sowizral y D. Waterman. 1981. The ROSIE language reference manual Tech. Rept. N-1646-ARPA. Rand Corp., Santa Monica, Calif..

[17] Anderson, R.H. y J.J Gillogly. 1976a. Rand intelligent terminal agent (RITA): Design philosophy. Rand paper R-1809-ARPA, Rand Corp., Santa Monica, Calif.
------ 1976b. The Rand intelligent terminal agent (RITA) as network design acces aid. In Proceeding of the AFIPS National Computer Conference. pp. 501-509.

[18] Newell, A. y H.A. Simon.1963. GPS: A program that simulates human thought. En E.A. Feigenbaum y J.A. Feldman, eds., Computer and thought. New York: McGraw-Hill.
------.1976. Computer Science as empirical enquiry: Symbols and search. [The 1975 ACM Turing Lecture.] Communications of the ACM 19, no. 3:113-126.

[19] Forgy, C., y J. McDermott.1977. OPS: A domain-independent production system language. En IJCAI 5, pp. 933-939.

[20] McDermott, J.1980a. R1: A rule-based configurer of computer systems. Technical Rept. CMU-CS-80-119. Department of Computer Science. Carnegie-Mellon University, Pittsburgh, Pa.
------.1980b. R1: An expert in the computer systems domain. En AAAI 1. pp. 269-271.
------.1980c. R1: An expert configurer. Rept. no. CMU-CS-80-119, Computer Science Department. Carnegie-Mellon University, Pittsburgh, Pa.

CAPITULO 3.

3. ARQUITECTURA DE SE.

Un SE es un programa de computadora que capta el conocimiento de un experto y lo aplica para resolver problemas prâcticos. Un SE debe tener ciertas caracterîsticas:

i) inferir a partir del conocimiento,
ii) hacer estas inferencias bajo incertidumbre y
iii) explicar como llega a los resultados.

Este capîtulo habla de la estructura y arquitectura de un SE [1,2]. Un SE consta de dos parte medulares: la màquina inferencial y la base de conocimientos. Alrededor de estas partes, trabajan otros mòdulos complementarios: mòdulo de adquisiciòn de conocimiento, interfases usuario-màquina, mòdulo de explicaciòn, interpretador de evidencias, etc.

3.1. LAS GENERACIONES DE SE.

Se consideran SE de primera generación a todos aquellos códigos de computadora que tenían conocimiento en una forma parecida a las bases de datos. Es decir, no había un procedimiento automático de inferencia ni de explicación. Un ejemplo es el SE DENDRAL [3].

En la segunda generación se hicieron sistemas con el conocimiento aún programado. Sin embargo, estos SE ya contaban con módulos explicativos, de modo que ya podían mostrar las líneas de razonamiento. Tal es el caso de MYCIN [4].

En la tercera generación, se tienen códigos de computadora en donde se distinguen los procesos de inferencia y el conocimiento en una forma independiente. Es decir, se puede utilizar tales códigos con diferentes bases de conocimiento. A estos sistemas se les llama esqueletos. Dentro de este tipo caen **EMYCIN** y **MYPRO**, este último presentado en esta tesis.

3.2. CARACTERISTICAS DE UN SISTEMA EXPERTO.

Debido a la variedad de aplicaciones en muchas disciplinas de la ciencia e ingeniería, los sistemas expertos han alcanzado un lugar preponderante dentro del área de inteligencia artifi-

cial. Un SE es un programa que se viste de experiencia de uno o más expertos en algún dominio de conocimiento y lo aplica haciendo útiles inferencias para el usuario del sistema. Un SE tiene características básicas que lo distinguen de los programas convencionales y de las cuales podemos citar tres de las más importantes:

- Capacidad de inferencia.
- Razonamiento con datos inciertos.
- Capacidad explicativa.

La figura 3.1 es un diagrama reminiscente de Feigenbaum [1], representando un SE o bien un sistema basado en conocimiento.

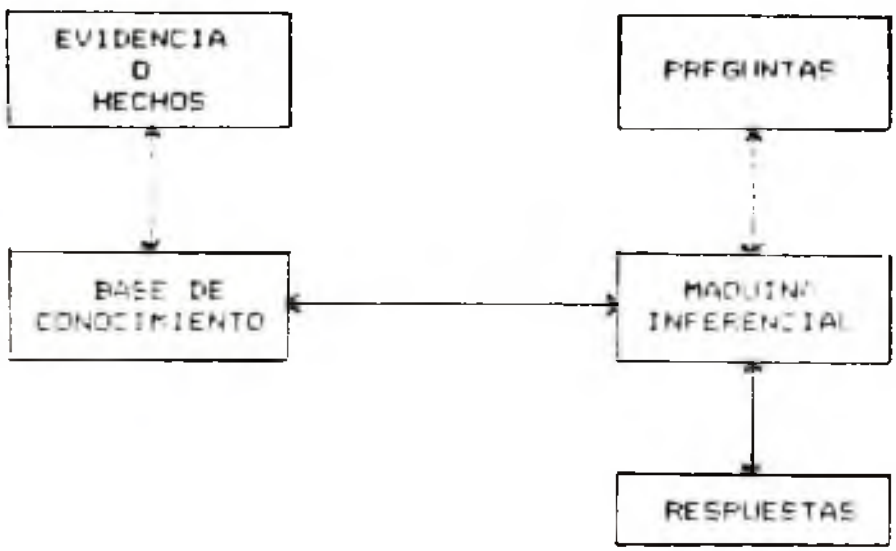


FIGURA 3.1 DIAGRAMA DE UN SE.

Un SE está basado sobre un extensivo cuerpo de conocimiento, acerca de un problema específico del área en cuestión. Característicamente, este conocimiento está organizado como una colección de reglas que permiten al sistema obtener conclusiones de datos o premisas dadas que son llamados hechos.

Se puede ver que el diseño de SE representa un cambio evolutivo. Sus consecuencias se marcan en el remplazo parcial del software tradicional por una nueva arquitectura centrada alrededor de una base de conocimiento y una máquina inferencial. Estos cambios han revolucionado tanto las áreas técnicas como las áreas comerciales. Notándose en los últimos cinco años un gran interés alrededor de este tema en todos los centros de investigación.

Un SE se puede considerar como la personificación dentro de una computadora de la habilidad y del conocimiento de un experto humano. Así que este sistema puede ofrecer consejos inteligentes y toma de decisiones acerca de un área específica de conocimiento. Una característica importante es que el sistema tiene la capacidad para justificar su propia línea de razonamiento en una forma inteligible, que resulta de la consulta del usuario. Esto es posible debido a que la programación del sistema para aceptar el conocimiento esta basado en reglas. La base de conocimiento constituída de esas reglas se cimenta sobre la arquitectura o estructura de este SE.

En resumen un SE tiene las siguientes caracteristicas:

- responde con datos inciertos o incompletos.
- puede explicar su linea de razonamiento en una forma comprensible.
- el conocimiento y el mecanismo de inferencia se distinguen claramente.
- se diseña para crecer en forma incremental.
- es màs declarativo que procedural.
- està tipicamente basado en reglas.
- diseñado para un dominio especifico de experiencia.
- puede emitir consejos como resultados de un diagnòstico o del tratamiento de un problema.

3.3. ESTRUCTURA BASICA DE SE.

En la figura 3.2. se muestra la estructura bàsica de un SE y de sus componentes. Sus componentes serán explicados en el orden siguiente:

- Base de Conocimientos.
- Máquina Inferencial.
- Módulos de Adquisición de Conocimiento.
- Módulos de Explicación.
- Interfase con el Usuario.

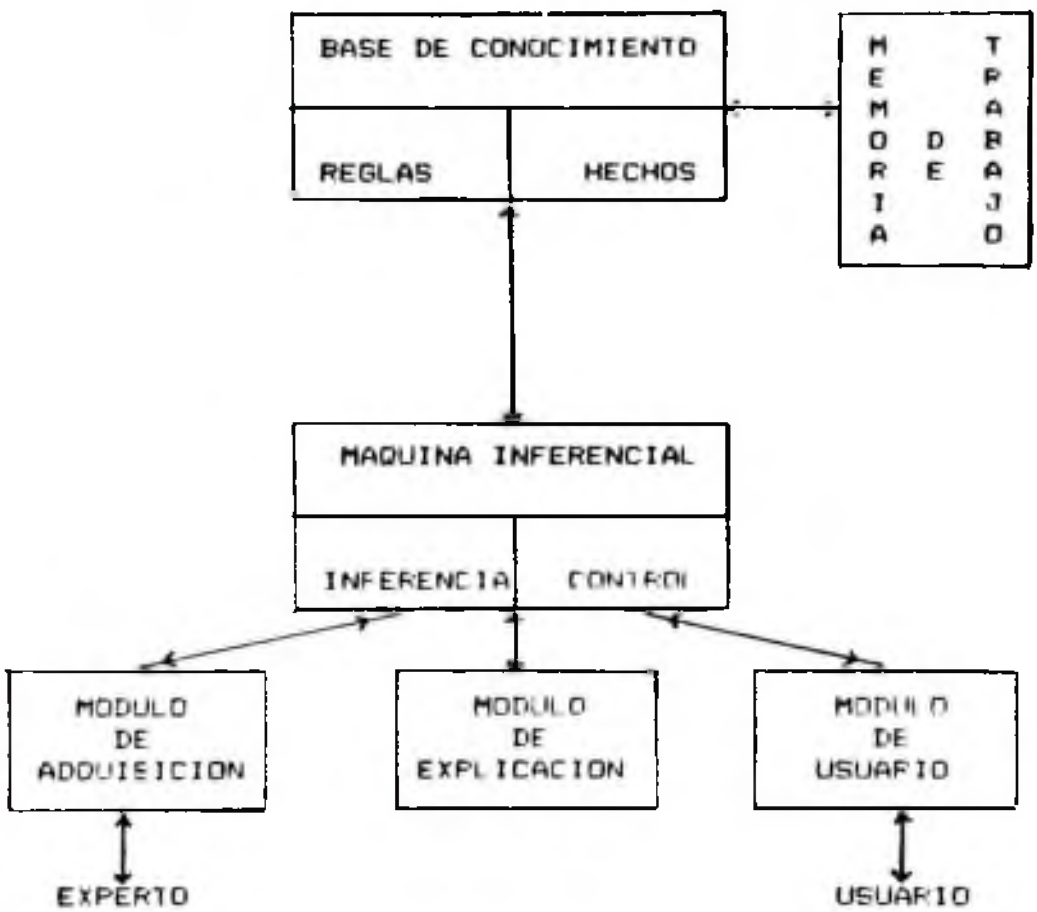


FIG. 3.2. ESTRUCTURA BASICA DE UN SE.

3.3.1. LA BASE DE CONOCIMIENTO.

Una base de conocimiento es un conjunto de reglas y hechos que se obtienen de la experiencia y el conocimiento de un experto humano. Los hechos son la información determinista y puede cambiar durante una consulta. Por ejemplo: Es un hecho "el agua es un líquido". Las reglas también son información, pero permiten generalizar un concepto. Por ejemplo: "todo líquido es un fluído", esta regla nos permite decir que el vino también es un fluído, ya que el vino es un líquido, por lo cual tendríamos una regla y un hecho de acuerdo a nuestra consideración.

REGLA: Todo líquido es un fluído.

HECHO: El agua es un líquido.

De esta forma se considera que dado un problema, este requerirá de cuando menos cuatro componentes. Estos componentes son la construcción de reglas, hechos, heurística, hipótesis y todo aquello que realmente no es una aplicación directa.

Para la adquisición de este conocimiento o de esta experiencia de un experto humano, es indispensable contar con técnicas que nos permitan representar conocimiento [5]. Algunas de estas técnicas son:

- Redes Semànticas.
- Objeto Atributo Valor.
- Reglas de Inferencia.
- Marcos Jeràrquicos.
- Lògica de Predicados.
- Sistemas de Producciòn.

Todas la tècnicas anteriores han sido utilizadas de una u otra forma, dependiendo del conocimiento que se quiso representar. Sin embargo, aûn cuando todas ellas sirven para representar conocimiento, no todo conocimiento puede ser representado por cada una de ellas. Asi que dentro de la identificaciòn del problema y la conceptualizaciòn del mismo, tambièn se investigarà que tècnica es la màs apropiada en la representaciòn del conocimiento para el problema en cuestiòn.

3.3.2. LA MAQUINA INFERENCIAL.

Uno de los puntos màs relevantes en el desarrollo de SE ha sido el de construir una màquina inferencial [6]. Esta màquina permite efectuar el razonamiento acerca de una pregunta de tal manera que pueda decidir sobre la afirmaciòn o negaciòn de la misma.

La màquina inferencial es la parte del sistema experto en donde toma lugar el razonamiento. Este razonamiento es de una forma estilizada con respecto a la pregunta que se le hace al sistema o de las respuestas que salen de èl. La màquina inferencial toma en consideraciòn la forma simple del problema particular que serà resuelto. La forma es simple con respecto a la pregunta que deberà ser respondida o bien aquello que sea relevante en la base de conocimiento. Ademas, la forma es simple màs que nada por virtud de su forma y no de su contenido.

A la màquina inferencial no le interesa de que se habla, sòlamente lo que se ha dicho o supuesto en esta base de conocimiento. Asi, la construcciòn de la base de conocimiento deberà ser estructurada de tal forma, que cada pieza de conocimiento estè dispuesta en una forma adecuada. De este modo, el usuario va desde la pregunta hasta la respuesta atravès de la màquina inferencial .

A continuaciòn se muestra un ejemplo con algunas piezas de conocimiento con hechos y reglas simples. La facilidad de este ejemplo es el de mostrar las lìneas de razonamiento que una màquina inferencial deberà seguir a la utilizaciòn de un algoritmo particular. En este ejemplo se observa còmo el razonamiento trabaja utilizando reglas de inferencia. Aòn cuando se tienen pocas piezas de conocimiento es necesario contar con un proceso automàtico, ya que, el problema crece a medida que aumentan las piezas de concimiento. Sin embargo, si el procedimiento funciona para una lista pequeña con un patròn de razonamiento, tambièn

trabajarà para una lista grande de estas piezas. Esta lista se mantiene actualizada en tal forma que permite obtener una respuesta adecuada. Ejemplo, considere las siguientes piezas de conocimiento:

1. Juan debe tomar aspirina si Juan sufre de dolor, la aspirina quita el dolor y no le hace mal a Juan.

2. La aspirina no le hace mal a Juan si la aspirina agrava úlcera y Juan no tiene úlcera.

De las dos proposiciones anteriores, podemos formar reglas que nos permitan hacer màs general el tratamiento de las situaciones anteriores. Asi podemos formar una pequeña base de conocimiento con las siguientes reglas y hechos:

REGLAS

R1. X debe-tomar Y if
X sufre-de Z &
Y quita Z &
Y no-hace-mal-a X

R2. X no-hace-mal-a Y if
X agrava Z &
not Y tiene Z

HECHOS

H1. aspirina quita dolor

H2. aspirina agrava úlcera

Si quisièramos resolver el problema: ?quê medicamento debe tomar Juan para suprimir el dolor?, iniciariamos un proceso de prueba (instanciamiento) para saber que regla(s) nos conduce(n) a una posible soluciòn.

El proceso de soluciòn [7] se llevaria a cabo de la siguiente manera. De R1 tenemos que para determinar què debe tomar Juan, debemos resolver tres condicionales o antecedentes. Sabemos que Juan sufre de dolor y es un hecho que la aspirina quita el dolor. Solo nos queda determinar si la aspirina no le hace mal a Juan.

El proceso se repite recursivamente. Para determinar si la aspirina no le hace mal a Juan, utilizamos R2. Es un hecho que la aspirina agrava la ùlcera. Sin embargo como Juan no tiene ùlcera, podemos concluir que la aspirina no le hace mal a Juan. Del razonamiento anterior, si Juan sufre de dolor, y no tiene ùlcera, podemos entonces inferir que debe tomar aspirina.

Bàsicamente el problema es encontrar todos los patrones posibles de razonamiento y lo que seria llamado el problema de bùsqueda. Asi para dar soluciòn al problema, se deberia considerar el nùmero de movimientos que hay que hacer ya que de ellos depende el pronto èxito de la bùsqueda. Esto es, entre menos movimientos se tengan, màs fàcil serà examinar todas las posibles lineas de razonamiento en el problema. Por ejemplo el juego del ajedrez es màs complicado que el juego del gato, ya que en èl se involucran màs movimientos [8]. Este proceso, Tècnicamente cono-

cido como resolución [9,10,11], es una regla de inferencia que puede ser codificada y descrita exactamente.

3.3.3 MODULO EXPLICATIVO.

Como se ha mencionado anteriormente, uno de los módulos que caracterizan a los SE es el de explicación. Este módulo se encarga de que el SE dè a entender lo que està efectuando. Es decir, en todo momento un SE responde a las actividades solicitadas por el usuario. Este tipo de explicaciones se presentan en tres puntos bàsicamente:

- Requerimientos que el sistema hace al usuario.
- Explicaciones que el usuario solicita al sistema.
- Datos o evidencias que el sistema cuestiona al usuario.

De estas características podríamos decir que cualquier otro sistema que cuente con ellas, podría identificarse como SE. Este módulo es muy parecido a la conversación cliente-experto. Esto es, platicando con el cliente, el experto puede concluir o imaginarse alguna solución al problema en cuestión.

Para el caso hombre-màquina se trata de formular una conversaciòn con un experto. Esto se lleva a cabo incluyendo los procesos de inferencia que el experto hace en una forma muy reducida. Esto es, el mòdulo explicativo està conectado con la màquina inferencial que a su vez es comunicada con la base de conocimiento.

Asì, el usuario a partir de una evidencia o hecho podrà llegar a una conclusiòn o explicaciòn de una acciòn que el sistema està haciendo con esos datos. De este modo el usuario va desde este mòdulo hasta la base de conocimiento, pasando por la màquina inferencial. Este recorrido se hace con la finalidad de seleccionar a travès de su control, la regla màs apropiada para concluir alguna meta parcial o total.

Este mòdulo es realmente importante dentro del desarrollo de SE. El mòdulo està encargado de efectuar el trabajo de consulta, y de ofrecer una explicaciòn nìtida y bien eslabonada para el usuario. Dicho de otra manera, una explicaciòn de las reglas y de como los hechos y evidencias han sido escogidos por el sistema para inferir un resultado.

3.4. TIPOS DE SISTEMAS EXPERTOS.

De acuerdo a múltiples aplicaciones en las que los SE pueden ser utilizados y al servicio que dan, los SE se pueden clasificar en los siguientes tipos [12]:

- Interpretación.
- Predicción.
- Diagnóstico.
- Diseño.
- Planeación.
- Instrucción.
- Control.
- Monitoreo.

INTERPRETACION

Sistemas de este tipo infieren descripciones de situaciones o estados a partir de datos observados con un cierto significado. Tal es el caso del sistema DENDRAL [3].

PREDICCION.

Sistemas de este tipo infieren consecuencias probables de situaciones dadas, usando modelos dinámicos que se ajusten a una

situaciòn. Casos típicos son los problemas de pronòstico, como las predicciones sobre tràfico, poblaciòn, clima, etc.

DIAGNOSTICO.

Estos sistemas infieren deterioros en las funciones de algòn dispositivo, o bien irregularidades observadas del comportamiento con causas adyacentes. Tal es el caso de MYCIN [4] usado en la medicina, electrònica, mecànica, etc.

DISEÑO.

Estos sistemas infieren mejores esquemas de objetos, a partir de datos y restricciones establecidas por el objeto. Por ejemplo en el caso SYN [12], XCON [15], estos se usan en el de diseño de circuitos impresos y otros similares.

PLANEACION

Estos sistemas infieren mejores modelos de organizaciòn y distribuciòn de objetos, bajo un conjunto de acciones que permiten elegir un modelo. Comunmente se ven en programaciòn automàtica y robòtica, comunicaciones y proyectos con SE del tipo de OP-PLANER [16].

INSTRUCCION

Estos sistemas infieren a través de un diagnóstico y corrección para ampliar el conocimiento en un área especifica en forma interactiva.

CONTROL

Estos sistemas infieren el comportamiento y control completo de un sistema, mediante la interpretación de la situación actual. Se usan para predecir fallas, diagnóstico de problemas a priori, haciendo planes correctivos y pruebas bajo esas situaciones. Tales casos se ven en control de tráfico, manejo de ecosistemas y emisiones.

MONITOREO

Estos sistemas infieren características cruciales, de éxito o fracaso en un determinado evento, a través de información corriente y supuestos de ocurrencia, así como de datos establecidos. Tales sistemas se usan en depuración de sistemas y corrección de problemas diagnosticados.

3.5. SUMARIO.

La arquitectura de los SE ha evolucionado. Inicialmente se tenían SE donde el conocimiento estaba mezclado en el programa. A estos SE se les denomina de primera generación. Después los SE llegaron a tener capacidad de explicar como llegaban a los resultados. A este nuevo tipo de SE se le consideró de segunda generación. Por último, se separaron los conocimientos de los programas. Esto dió origen a lo que se conoce como esqueletos de SE o SE de tercera generación. Estos esqueletos se visten de diferentes bases de conocimientos para formar SE particulares.

Los SE tienen dos partes importantes: la base de conocimientos y la máquina inferencial. La base de conocimientos es un conjunto de hechos y reglas tomados de la experiencia de una persona. La máquina inferencial es un programa de computadora capaz de manejar la base de conocimientos y hacer inferencias útiles para resolver algún problema.

Este es el esquema básico de la arquitectura de un SE. Sin embargo, algunas características pueden cambiar según el tipo de SE. Por ejemplo, un SE para diagnóstico médico [4,13] no tiene las mismas necesidades que un SE para análisis estructural [14]. Por último se puede decir que los SE tienden a ser aplicados en todos aquellos problemas que para su solución requieren más razonamiento que cálculo.

REFERENCIAS.

[1] **Feigenbaum, et al.** 1981. The Handbook of Artificial Intelligence, Wm, Kaufmann, Inc.

[2] **Chapa V. Sergio V.** 1985. Arquitectura de Sistemas Expertos, Reporte Técnico AM 14, CINVESTAV-IPN. México, D.F.

[3] **Buchanan, B. G., G. L. Sutherland y E. A. Feigenbaun.** 1969. Heuristic DENDRAL: A program for generating explanatory hypotheses in organic chemistry. En B. Meltzer y D. Michie, eds., Machine Intelligence, vol. 4. Edinburgh: Edinburgh University Press, pp. 209-254.

[4] **Shortliffe, E. H.** 1976. Computer-based medical consultation: MYCIN, New York, Elsevier.

[5] **Harmon, P. y D. King.** 1985. Expert Systems AI in Business. John Wiley & Sons.

[6] **Hayes, J.E. y D. Michie.** 1983. Intelligence Systems. Ellis Horwood limited. John Wiley & Sons.

[7] **Clark, K. L. y F. G. McCabe.** 1984. Micro-Prolog: Programing in Logic, Prentice-Hall.

[8] **Slagle, J.R.** 1971. Artificial Intelligence: The Heuristic Programing Aproach. McGraw-Hill Book Company.

[9] **Chang Chin-Liang y Richard Char-Tang Lee.** 1973. Symbolic logic & mechanical theorem proving. New York. Academic Press.

[10] **Kowalski, Robert.** 1979. Logic for problem solving. Artificial Intelligence Series. Elsevier North Holland Inc.

[11] Nilson, N.J.. 1980. Principles of Artificial Intelligence. Tioga, Palo Alto, California.

[12] **Andriole, S.** 1985. Applications in Artificial Intelligence. Petrocelli Books, Inc., Princeton, N.J.

[13] **Clancey, W. J., E. H. Shortliffe y B. G. Buchanan.** 1979. Intelligence Computer-Aided Instruction for Medical Diagnosis. Proceedings of the 3rd. Symposium on Computer Application in Medical Case. pp. 175-183.

[14] **Bennet, J., Creary, L. Engelmore, R. S. y Melosh, R.** 1978. SACON: A Knowledge Based Consultant in Structural Analysis., Heuristic Programming Project Report No. HPP-78-28 & STAN-CS-78-699. Computer Science Department Stanford University.

CAPITULO 4.

4. ETAPAS EN LA CONSTRUCCION DE UN SE.

La construcción de un SE se lleva a cabo en varias etapas [1,2]. Estas etapas son: identificación del problema, conceptualización, formalización, implantación y pruebas. Para la identificación del problema se reúnen los expertos con el ingeniero de conocimiento y analizan las necesidades del problema a resolver. La fase de la conceptualización consta de tres partes: la adquisición de conocimiento, la elección de una representación adecuada para ese conocimiento y finalmente se determina el tipo de máquina inferencial que se necesita.

En la parte de formalización se dan los detalles del SE. Estos detalles dependen del tipo de SE. Es decir, si se requiere interfase con algunos archivos, con dispositivos externos o con el usuario. En que medio trabajará, a cuantos usuarios dará servicio, etc. Una vez diseñado el SE se programa. Finalmente se implanta y se prueba. En el período de prueba, un experto verifica que los resultados sean verídicos y confiables. En base a los resultados obtenidos, el experto checa y refina la base de conocimientos.

4.1. IDENTIFICACION DEL PROBLEMA.

El primer paso en el desarrollo y construcción de un SE es identificar el problema. En la etapa de identificación del problema, el ingeniero de conocimiento y las personas expertas en el àrea, deben analizar el problema que se va a resolver. Esto lo deben hacer en tèrminos de las necesidades que se tienen, los recursos con que se cuentan y en general deben contemplar los siguientes aspectos:

i) Decidir si el problema que se va a resolver amerita el desarrollo de un SE. El desarrollo de un SE es costoso, porque involucra gente especializada y por un tiempo considerablemente largo. Muchos problemas pueden ser resueltos aceptablemente con herramientas tradicionales existentes.

ii) Analizar los recursos con los que se cuenta, tanto financieros, humanos, de cômputo y de infraestructura para el desarrollo del sistema. Por ejemplo, con que capital se cuenta, que nûmero de gente participarà en el proyecto y por cuanto tiempo. Se debe tambièn considerar si existe gente preparada en el àrea del conocimiento en cuestiòn y en el àrea de computaciòn para el desarrollo del prototipo del sistema.

iii) Minimizar la probabilidad de èxito en el desarrollo del

sistema. El desarrollo de un SE es una tarea difícil que toma bastante tiempo, sobretodo en la adquisición del conocimiento que es un proceso lento y costoso. La cuestión es determinar en cuanto tiempo se va a desarrollar el sistema y si se cuenta con la gente preparada para el proyecto, durante todo ese tiempo. Es importante que el ànimo no decaiga y el proyecto llegue a culminarse exitosamente.

iv) Por último la rentabilidad del sistema. Esto es, hay que analizar si el sistema que se va a desarrollar es un sistema comercial que pueda ser ofrecido a varias empresas. Es necesario analizar si vale la pena el esfuerzo de las personas que lo construyan y la inversión que se vaya a hacer. Por lo tanto, se debe hacer un anàlisis de factibilidad costo/beneficio, para determinar si en realidad si su desarrollo es costeable.

Los puntos arriba mencionados deben ser considerados a conciencia antes de comenzar a desarrollar un SE. Cabe mencionar que dentro del anàlisis de identificación del problema hay otros puntos finos a los que se han mencionado y varían de acuerdo a la naturaleza del problema.

4.2. CONCEPTUALIZACION.

Una vez que se ha indentificado el problema, que se han determinado los alcances del sistema, sus posibles metas, etc., se debe conceptualizar. La fase de conceptualizaciòn es la parte en la que se entrevistan el ingeniero de conocimiento con el(los) experto(s) para abstraer el conocimiento del àrea en cuestiòn. A este proceso se le denomina adquisiciòn de conocimiento.

Al mismo tiempo, conforme se adquieren partes de la base de conocimiento y se observan sus caracterìsticas, el ingeniero de conocimiento debe determinar que esquema de representaciòn del conocimiento es màs adecuado para esa pieza de conocimiento que se desea modelar. El ingeniero de conocimiento debe determinar si existe algùn esqueleto que se ajuste a sus necesidades. Tambièn el ingeniero de conocimiento determina si èste tiene el suficiente poder expresivo para modelar su problema, o bien si se construye uno especial.

Otro aspecto importante en la conceptualizaciòn del problema es la selecciòn de un mecanismo que trate la incertidumbre, acorde a la naturaleza del problema y que use los modelos ya desarrollados para hacerlo.

4.2.1. ADQUIBICION DEL CONOCIMIENTO.

Se le denomina adquisición del conocimiento [3] al proceso mediante el cual el experto y el ingeniero de conocimiento plasman el conocimiento del área en cuestión en la base, codificada de manera congruente a la representación elegida. El experto tiene varios tipos de conocimiento, tales como experiencias personales de problemas pasados, métodos propios para resolver problemas, conocimiento de métodos a utilizar en la solución de problemas particulares, etc.

Usualmente la tarea de relacionar las experiencias personales y llevarlas a la forma de reglas es una tarea complicada para el experto. Con el fin de superar esta barrera, se requiere del desarrollo de lenguajes de adquisición de conocimiento y de representaciones adecuadas que faciliten esa tarea. Loque éxitos de algunos SE, se debe a que el conocimiento ha sido representado como un conjunto de reglas modulares con una sintáxis muy simple.

Es de importancia en el proceso de adquisición de conocimiento la selección del experto, esto es, a quien se acudirá para construir la base. En la mayoría de los SE actuales, el ingeniero de conocimiento entrevista al experto, obtiene las principales características del problema y construye un prototipo. Sin embar-

go, la mayoria de los problemas reales que ameritan ser modelados por un SE son demasiado complejos, como para que un inexperto pueda ayudar a la adquisición de conocimiento.

En suma, el proceso de adquisición de conocimiento es una de las fases màs dificiles en la construcción de un SE. Seria entonces conveniente tener una herramienta adecuada mediante la cual el experto pudiera construir la base de conocimientos directamente. Esto es, que la construcción fuera lo màs natural posible, dado que por lo general, los expertos son gente no familiarizada con las computadoras.

4.2.2. REPRESENTACION DEL CONOCIMIENTO.

En el proceso de adquisición del conocimiento, el ingeniero de conocimiento debe identificar los problemas que se presentan y que el SE va a resolver. Entonces debe elegir una representación adecuada para ese conocimiento. La representación elegida deberà tener dos caracteristicas importantes [4]:

- el poder para expresar el conocimiento de esa àrea.
- simplicidad para describir, actualizar o explicar el conocimiento del modelo.

El poder expresivo de una representación puede ser medido de varias maneras. Una es que el conocimiento más relevante del experto pueda ser relativamente fácil de representar para resolver un problema. Además del poder expresivo, hay un compromiso entre la facilidad de representación del conocimiento en la computadora y la riqueza de estructuras de posibles relaciones semánticas que pueden ser descritas en la representación.

Finalmente, para una buena selección de la representación del conocimiento, es importante que el SE tenga herramienta que ofresca la facilidad para actualizar la base. Los SE son notoriamente cambiantes en sus descripciones, sobretodo en las primeras etapas del desarrollo del prototipo. Por consiguiente, es necesario tener flexibilidad para cambiar las reglas en el momento que se requiera. Un SE que requiere de estructuras complejas no facilitará por lo general esta tarea. Los esquemas más usuales de representación de conocimiento son:

- Redes Semanticas,
- Objeto Atributo Valor,
- Reglas,
- Marcos Jerarquicos y

Las cuales se describen en seguida.

4.2.2.1 REDES SEMANTICAS.

Uno de los esquemas de representación del conocimiento más antiguos y generales usados en Inteligencia Artificial, son las redes semánticas [5]. Una red semántica es una colección de objetos llamados nodos. Estos nodos están relacionados por arcos y tanto los nodos como los arcos están etiquetados. Los nodos usualmente representan objetos o descriptores. Los objetos pueden ser entidades físicas o conceptuales. Los descriptores proporcionan información adicional y en ocaciones clasificativa de los objetos. Los arcos relacionan objetos y descriptores. Algunas etiquetas comunes para los arcos son: es-un, tiene, etc. La figura 4.1. muestra una red semántica que describe una mesa.

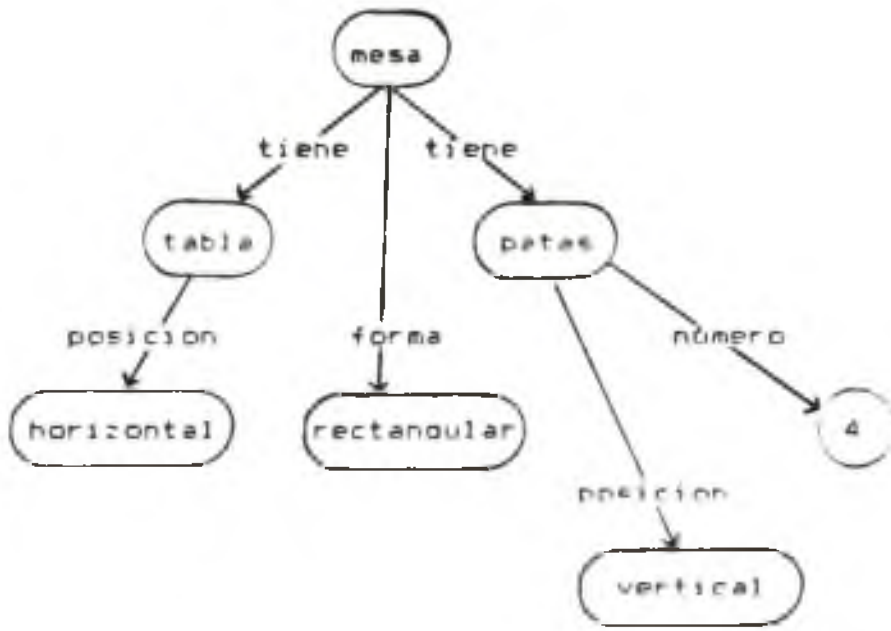


FIG. 4.1. RED SEMANTICA.

4.2.2.2 OBJETO ATRIBUTO VALOR.

Otra manera comùn de representar conocimiento es mediante las tripletas OAV [6]. Este OAV es el esquema es usado en MYCIN para representar hechos. Los objetos pueden ser entidades físicas o conceptuales, por ejemplo una puerta o una cuenta de banco. Los atributos son generalmente características o propiedades asociadas a los objetos, como tamaño, forma, color, etc. Finalmente el valor especifica la naturaleza del atributo en una situaciòn particular. Por ejemplo el color de una manzana puede ser rojo. De alguna manera, el esquema OAV es un caso particular de la representaciòn por redes semànticas. Por ejemplo, la relaciòn OBJETO->ATRIBUTO puede ser representada por un arco **tiene** y la relaciòn ATRIBUTO<-VALOR por un arco **es-un**.

4.2.2.3. REGLAS.

Las reglas [7] son usadas para representar relaciones. Las reglas pueden ser muy complicadas o simples, como en el caso de las reglas de PROLOG [8,9] o MYCIN [10]. Las reglas estàn formadas de dos partes, las premisas y la conclusiòn, o bien si deseamos llamarlas de otra manera, los antecedentes y el consecuente o las condiciones y la hipòtesis. Una regla tiene la

forma:

Hipótesis IF
Condición 1 &
Condición 2 &
...
...
Condición n

Las condiciones y las hipótesis pueden ser representadas como relaciones o predicados lógicos (un caso especial serían las tripletas OAV). Por ejemplo un regla de MYCIN:

	Atributo	**Objeto**	**Valor**
IF	Sitio	Cultivo	Sangre
	Morfología	Organismo	Rod
	...		
THEN	Identidad	Organismo	Pseudomona-aeruginosa

Estas reglas pueden tener como condiciones otras reglas que a su vez estarán definidas en base a hechos, evidencias cuestionables al usuario o también por otras reglas. Incluso pueden llegar a tener variables en cualquier sitio.

4.2.2.4 MARCOS JERARQUICOS.

Los marcos jerarquicos proporcionan otro esquema para representar hechos y relaciones [4]. Un marco es una descripciòn de un objeto por medio de sus atributos, colocados en ranuras . Las ranuras almacenan valores (pueden ser valores por default), apuntadores a otros marcos o el procedimiento mediante el cual los valores pueden ser obtenidos. Finalmente los marcos son estructuras muy ricas en poder expresivo para representar conocimiento. aunque su implementaciòn es mucho màs complicada que la de reglas o tripletas OAV. La figura 4.2. muestra un marco que describe la situacion de un salòn de clase.

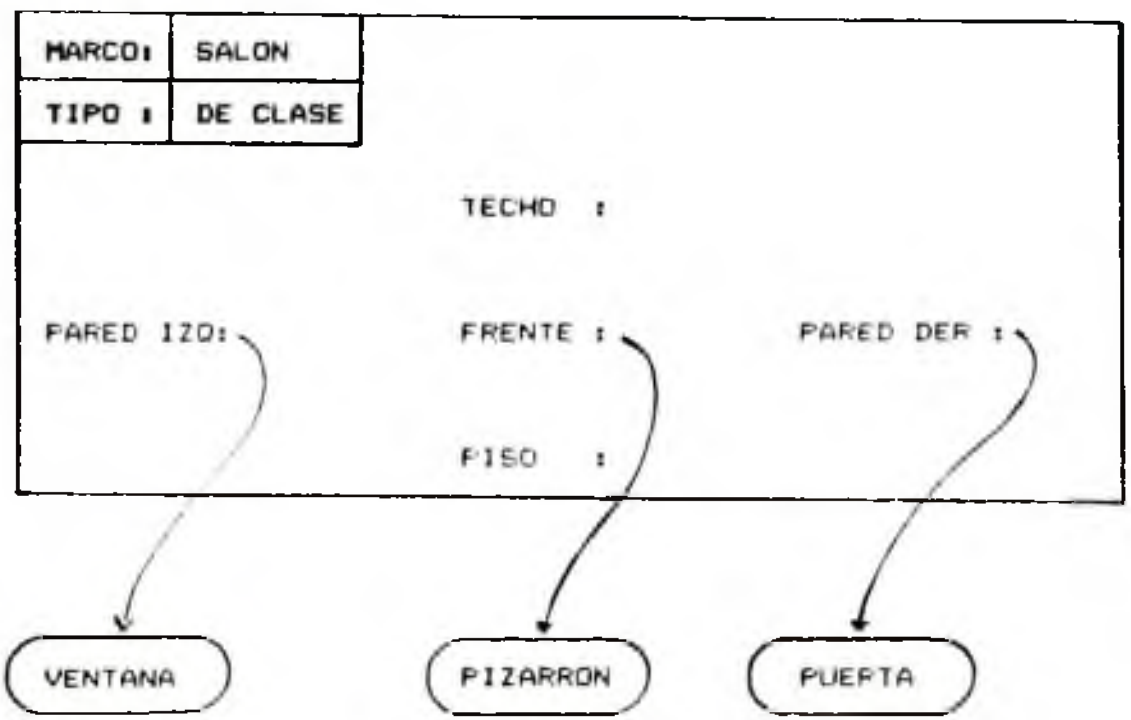


FIG. 4.2. MARCO JERARQUICO.

4.2.2.5. REPRESENTACION DECLARATIVA .VS. PROCEDURAL.

La inclusión de procedimientos en los marcos es un ejemplo de la combinación de dos estrategias de representación de conocimiento en un solo esquema : la procedural y la declarativa. La representación declarativa de un hecho simplemente asegura que es cierto, por ejemplo "Enero tiene 31 días". La representación procedural de un hecho es un conjunto de instrucciones que nos permiten calcularlo.

Las representaciones declarativa y procedural son estrategias alternativas que finalmente pueden rendir los mismos resultados. Sin embargo debido a su modularidad inherente, la representación declarativa es más transparente al usuario y más fácil de mantener. O sea que que los usuarios y expertos se entienden mejor con este tipo de representación. Por otro lado, la representación procedural es mucho más eficiente, pero más difícil de mantener. Sin embargo los ingenieros de conocimiento, están más influenciados por ella.

4.2.3. RAZONAMIENTO BAJO INCERTIDUMBRE.

Uno de los sueños en computación desde sus inicios fué hacer máquinas inteligentes. Hoy en día, todos los programas de computadora consisten de la repetición de pasos o accesos veloces a grandes volúmenes de información. Sin embargo, cualquier decisión que toma la máquina, debe estar previamente codificada en cierto lenguaje que es inflexible. De hecho fue el programador quien aportó el conocimiento y quien, en última instancia, tomó la decisión.

Por los medios tradicionales de programación, somos incapaces de hacer que las máquinas puedan competir con el hombre en tareas inteligentes. Esto es, no ha sido posible hacer tareas como reconocimiento o traducción de lenguajes, hacer abstracciones o generalizaciones, tomar decisiones bajo incertidumbre, etc.

Esta incapacidad para programar las máquinas se debe a la dificultad que existe para que la inteligencia humana pueda ser simulada a través de las máquinas. Esta dificultad estriba principalmente en que el cerebro humano tiene la capacidad de pensar, razonar e inferir en términos imprecisos, de incertidumbre o "borrosos" [11]. Por ejemplo, la capacidad de razonar en términos borrosos es la que permite descifrar letras mal escritas o remover la ambigüedad que se presenta en el lenguaje natural. Es la falta de esa capacidad la que hace que aún las mejores y más sofisticadas computadoras sean incapaces de desarrollar tales

tareas.

La mayoría de los problemas con los que el hombre se enfrenta son borrosos y las decisiones que toma son bajo incertidumbre. Por ejemplo, cuando un médico trata de hacer un diagnóstico, debe considerar evidencias a interrogantes como: ?le duele la cabeza?. Del mismo modo un mecánico puede preguntar: ?el carro tira aceite?. Las respuestas a esas interrogantes son por lo general imprecisas; respuestas como: a veces, un poco, frecuentemente, etc.

Existen otros tipos de problemas, a los cuales nos hemos enfrentado bajo ciertos modelos matemáticos sin comtemplar la posibilidad de incertidumbre. Por ejemplo se puede determinar cual es la probabilidad de que al tirar un par de dados caigan dos aces. Sin embargo, la pregunta bien podría ser : ? cuál es la probabilidad de que al tirar un dado caiga un número cercano al tres?.

En las investigaciones hechas para la solución de esos problemas, se han desarrollado modelos matemáticos exactos que dan lugar a la borrosidad. Alrededor de los 70's, L. Zadeh [12] desarrolló la teoría de subconjuntos borrosos, modelo matemático que nos permite considerar la incertidumbre, con la cual tenemos que trabajar a diario.

Por otro lado, Shortliffe [10] desarrolló en el modelo de MYCIN una interpretación especial del teorema de Bayes. En esta interpretación se definen factores de certeza en función de las probabilidades a priori y condicionadas.

4.2.3.1. PROBABILIDAD.

La mayoría de los programas de toma de decisiones en el campo de la medicina son basados en teorías estadísticas. Sin embargo, los algoritmos de solución de estos programas varían desde simples árboles binarios de decisión hasta modelos basados en probabilidad condicional y análisis discriminante.

Hasta la fecha, la técnica más empleada en la solución de toma de decisiones bajo incertidumbre es el teorema de Bayes [13]. El teorema de Bayes en este contexto se utiliza para propagar las probabilidades que generan las evidencias independientes que soportan una determinada hipótesis.

El problema de toma de decisiones bajo incertidumbre puede ser expresado asignando una medida de probabilidad de que se cumplan las hipótesis en vista de las evidencias relevantes. Si al conjunto de evidencias relevantes le denominamos **E** y **hi** a la i-ésima hipótesis, **P(hi|E)** es la probabilidad condicional de que

se cumpla la hipòtesis en vista de las evidencias E. El teorema de Bayes nos permite calcular las P(hi|E) por medio de la siguiente fòrmula:

$$P(hi|E) = \frac{P(hi)\ P(E|hi)}{\sum_{j=1,n} P(hj)\ P(E|hj)}$$

donde P(hi) es la probabilidad a priori de que se presente la hipòtesis hi, sin consideraciòn alguna de la evidencia. P(E|hi) es la probabilidad inversa, o sea la probabilidad de que se presenten las evidencias, a sabiendas de que se cumple la hipòtesis.

Existen varias maneras de aplicar el teorema de Bayes en sistemas de clasificaciòn. Por ejemplo, el SE PROTECTOR utiliza una tècnica de propagaciòn de probabilidades basada en el concepto de disparidad y verosimilitud.

La fòrmula de Bayes nos dice que:

$$P(H|E) = \frac{P(E|H)\ P(H)}{P(E)}$$

Igualmente para el complemento de la hipótesis:

$$P(\bar{H}|E) = \frac{P(E|\bar{H})\ P(\bar{H})}{P(E)}$$

Dividiendo se obtiene:

$$\frac{P(H|E)}{P(\bar{H}|E)} = \frac{P(E|H)\ P(H)}{P(E|\bar{H})\ P(\bar{H})}$$

Duda [13] define ventajas a priori ("priori odds") disparidad a priori de H como:

$$O(H) = \frac{P(H)}{P(\bar{H})}$$

y las ventajas a posteriori ("posteriori odds") como:

$$O(H|E) = \frac{P(H|E)}{P(\bar{H}|E)}$$

la razòn de probabilidad λ como:

$$\lambda = \frac{P(E|H)}{P(E|\bar{H})}$$

y la fòrmula de ventaja de probabilidad como:

$$O(H|E) = \lambda\ O(H)$$

Esta ecuaciòn nos dice como actualizar o propagar las ventajas de H dada la observaciòn de la evidencia E. En un SE como PROSPECTOR, se asume que la razòn de probabilidad \ la proporciona el experto humano como una medida de veracidad de cada regla.

Una vez que se conocen las P(hi|E), se pueden considerar varios criterios. Por ejemplo la hipòtesis màs recomendable es la que tiene la mayor probabilidad. Tambièn puede darse el caso de que las hipòtesis màs recomendables sean las de probabilidad màs alta (cuya suma sea cercana a la unidad). Aun màs, se puede hacer uso de algunas consideraciones heurìsticas o pragmàticas. Por ejemplo, en el caso de diagnòsticos mèdicos se puede presentar una probabilidad baja de una enfermedad crìtica y aunque la

probabilidad es baja, amerita atención terapêutica, dado que la probabilidad de ocurrencia no es cero y la enfermedad puede ser mortal.

4.2.3.2. EL MODELO DE MYCIN.

Al igual que a cada regla se le puede asociar una medida de probabilidad, también a la regla se le puede asociar un factor de certeza CF [10]. Este CF refleja el grado con el que se cree que la regla se va a cumplir.

Un CF es un número en el intervalo [-1,1], que refleja el grado de creencia de una hipótesis. Mientras mayor sea el CF, mayor es la certeza que se le va a dar a una hipótesis. Cuando CF=1, se sabe con completa seguridad que la hipótesis es cierta.

Cuando el CF es negativo, expresa una medida de creencia de que la hipótesis es falsa. Mientras menor sea el CF, hay mayor seguridad de que la hipótesis es falsa.

Cuando CF=0, no se puede asegurar nada ni en favor ni en contra de la hipótesis. Esto se puede dar en dos casos:

i) a causa de información incompleta;

ii) si la evidencia que soporta la hipótesis está balanceada con la evidencia que sugiere que es falsa.

Dado que el CF debe reflejar tanto la evidencia que soporta la hipótesis como la evidencia que la refuta, se puede descomponer en dos partes: una medida de creencia en la hipótesis y una medida de creencia en que no se cumple la hipótesis.

CF(H,E) es el factor de certeza de que la hipótesis se cumple, basados en la evidencia E.

MB(H,E) es la parte confirmante de CF(H,E), a la cual se le llama medida de creencia.

MD(H,E) es la parte desconfirmante de CF(H,E), a la cual se le llama medida de descreencia.

Estas medidas de creencia no son medidas de probabilidad. Se comportan de acuerdo a otras reglas y para su propagación en las redes de inferencia, se debe recurrir a la teoría de conjuntos borrosos.

De acuerdo a la teoría de probabilidad, la probabilidad P(h) refleja la creencia del experto en esa hipótesis y 1-P(h) puede ser vista como una estimación de la descreencia del experto en h. De modo que si P(h|e) > P(h) entonces la observación indica que se incrementa la creencia del experto en h. Así que el decremento en la descreencia o medida del incremento en la creencia de h

como resultado de la observación de e, se puede expresar como:

$$MB(h,e) = \frac{P(h|e) - P(h)}{1 - P(h)}$$

Análogamente, si P(h|e) < P(h), la observación de e nos llevó a decrementar nuestra creencia en h. Entonces la medida de incremento de descreencia de h dada la observación de e, se puede expresar por la razón:

$$MD(h,e) = \frac{P(h) - P(h|e)}{P(h)}$$

Resumiendo, podemos expresar MB y MD como:

$$MB(h,e) = \begin{cases} 1 & \text{si } P(h) = 1 \\ \dfrac{\max[\,P(h|e),\,P(h)\,] - P(h)}{\max[\,1,\,0\,] - P(h)} & \text{en otro caso} \end{cases}$$

$$MD(h,e) = \begin{cases} 1 & \text{si } P(h) = 0 \\ \dfrac{\min[\,P(h|e),\,P(h)\,] - P(h)}{\min[\,1,\,0\,] - P(h)} & \text{en otro caso} \end{cases}$$

Con el fin de entender mejor en lo que son las medidas de creencia y lo que representan, se enumeran las siguientes características:

a) Rangos.

$$0 \leq MB(h,e) \leq 1$$
$$0 \leq MD(h,e) \leq 1$$
$$-1 \leq CF(h,e) \leq 1$$

b) Hipótesis mutuamente exclusivas.

Si h es cierta:	Si h es falsa:
MB(h,e) = 1	MB(h,e) = 0
MD(h,e) = 0	MD(h,e) = 1
CF(h,e) = 1	CF(h,e) = -1

Si h1 y h2 son mutuamente exclusivas:

$$MB(h1,e) = 1 \Rightarrow MD(h2,e) = 1$$

$$MD(h1,) = 1 \Rightarrow MB(h2,e) = 1$$

c) Falta de evidencia:

Si

$MB(h,e) = 0$ si h no es confirmada por e. (h y e son independientes)

Y

$MD(h,e) = 0$ si h no es desconfirmada por e.

Entonces

$CF(h,e) = 0$

d) Si existe evidencia soportando una hipótesis en grado X, esta evidencia soporta la negación de la hipótesis en grado -X.

$$CF(h,e) + CF(\text{no } h,e) \neq 1$$

$$CF(h,e) + CF(\text{no } h,e) = 0$$

Esta característica es intuitiva, dado que si cierta evidencia soporta una hipótesis, refuta la negación en igual medida. O sea:

$$MB(h,e) = MD(\text{no } h,e)$$

e) Por último, cabe remarcar que si se tienen n hipótesis mutuamente exclusivas, confirmadas por la evidencia e, entonces:

$$-n <= \sum_{i=1}^{n} CF(hi,e) <= 1$$

FUNCIONES COMBINADAS.

Existe un número de convenciones que se han establecido con el fin de poder aplicar el modelo de factores de certeza al problema de la toma de decisiones médicas. Se supone que las medidas de creencia y descreencia que el experto proporciona son

aproximaciones aceptables de las medidas que serían calculadas si se conocieran las probabilidades requeridas. Por lo tanto podemos afirmar que E = E+ U E- si E+ la parte confirmante de la evidencia y E- la parte desconfirmante de la misma. De lo anterior tenemos que:

1) Límites.

MB(h,E+) -> 1 conforme se encuentra evidencia confirmante.
MD(h,E-) -> 1 conforme se encuentra evidencia desconfirmante.
CF(h,E-) <= CF(h,E+ & E-) <= CF(h,E+)

2) Confirmación o desconfirmación absoluta.

MB(h,E+) = 1 => MD(h,E-) = 0 ∴ CF(h,E) = 1
MD(h,E-) = 1 => MD(h,E+) = 0 ∴ CF(h,E) = -1
MB(h,E+) = MD(h,E-) = 1 es contradictorio
CF está indefinido

3) Conmutatividad.

MB(h,E1&E2) = MB(h,E2&E1)
MD(h,E1&E2) = MD(h,E2&E1)
CF(h,E1&E2) = CF(h,E2&E1)

4) Información incompleta.

Denotemos por E? una pieza de evidencia cuya veracidad o falsedad se desconoce.

MB(h,E1&E?) = MB(h,E1)
MD(h,E1&E?) = MD(h,E1)
CF(h,E1&E?) = CF(h,E1)

Tomando en cuenta esas convenciones, se pueden presentar las siguientes funciones combinadas:

1) Evidencia adquirida incrementalmente.

$$MB(h,E1\&E2) = \begin{cases} 0 & \text{si } MD(h,E1\&E2) = 1 \\ MB(h,E1) + MB(h,E2)[1 - MB(h,E1)] & \end{cases}$$

$$MD(h,E1\&E2) = \begin{cases} 0 & \text{si } MB(h,E1\&E2) = 1 \\ MD(h,E1) + MD(h,E2)[1 - MD(h,E1)] & \end{cases}$$

2) Conjunción de hipótesis.

$$MB(h1\&h2,E) = \min[MB(h1,E),MB(h2,E)]$$
$$MD(h1\&h2,E) = \max[MD(h1,E),MD(h2,E)]$$

3) Disyunción de hipótesis.

$$MB(h1+h2,E) = \max[MB(h1,E),MB(h2,E)]$$
$$MD(h1+h2,E) = \min[MD(h1,E),MD(h2,E)]$$

4) Veracidad de la evidencia.

$$MB(h,E) = MB'(h,E) * \max[0,CF(E,E1)]$$
$$MD(h,E) = MD'(h,E) * \min[0,CF(E,E1)]$$

La primera función implica que como MB (MD) representa un decremento proporcional en descreencia (creencia), el MB de una nueva evidencia debe aplicarse sólo a la descreencia (creencia) restante.

La segunda función implica que la MB (MD) en la conjunción de dos hipótesis es tan buena como el mínimo (máximo) de las dos.

La tercera función implica que la MD (MB) en la disyunción de dos hipótesis es tan buena como el máximo (mínimo) de las dos.

La cuarta función implica que si la veracidad de una evidencia E no es conocida, pero se conoce el CF de E dada otra evidencia El, la medida de creencia puede ser calculada.

EJEMPLOS DE USO.

En este modelo, el conocimiento está representado en reglas de producción. Cada regla tiene un CF asociado que refleja la MB y MD que el experto sugiere. El modelo recuerda las hipótesis alternativas que son confirmadas o refutadas por las reglas. Con cada regla se almacenan su MB y MD, que inicialmente se consideran como cero. Cuando una regla para inferir una hipótesis se hace cierta a luz de una evidencia, se actualiza su MB o MD, según corresponda. Por ejemplo, sea la regla:

```
Juan debe-tomar aspirina (0,8) if
     Juan sufre-de dolor &
     aspirina quita dolor &
     aspirina no-hace-mal-a Juan
```

es de la forma H <- S1 & S2 & S3. Si del usuario se obtuvo que **Juan sufre-de dolor** (0.87), entonces CF(S1,E) = 0.87.

Además se conoce que **aspirina quita dolor** (0.9), o sea que CF(S2,E) = 0.90 y por otra regla se ha determinado que **aspirina no-hace-mal-a Juan** (0.6), o sea, CF(S3,E) = 0.60. Por medio de las funciones combinadas:

$$CF(S1\&S2\&S3,E) = \min[0.87, 0.9, 0.61]$$

$$CF(H,S1\&S2\&S3) = MB(H,S1\&S2\&S3) - MD(H,S1\&S2\&S3)$$

$$= 0.8 * CF(S1\&S2\&S3,E)$$

$$= 0.8 * 0.6 = 0.48$$

Como otro ejemplo, si una hipótesis H es confirmada por una sola regla con un MB(H,E) = 0.3 y MD(H,E) = 0, entonces CF(H,E) = 0.3. Si se encuentra una nueva regla con CF(H,E) = 0.2, ahora CF(H,E) = 0.2 + 0.3 - 0.2 * 0.3 = 0.44. Si finalmente se encuentra una regla con CF(H,E) = -0.1, el factor de certeza resultante será CF(H,E) = 0.44 - 0.1 = 0.34.

4.2.3.3. CREENCIA .VS. PROBABILIDAD.

Aunque no lo sea, se ha dicho que el factor de certeza, la medida de creencia y la de descreencia pueden ser vistas como probabilidades. Por eso a continuación mostraré las diferencias. Además presentaré las ventajas y desventajas entre estos modelos.

Los factores de certeza no pueden ser considerados como probabilidades en sentido estricto. Por ejemplo, el que una evidencia soporte una hipótesis en grado X, no significa que soporte la negación de la hipótesis con grado 1-X, como se haría en la teoría de probabilidades. Esto es, en términos de factores de certeza:

$$CF(h,e) + CF(\text{no } h,e) \neq 1$$

desarrollando tenemos:

$$CF(\text{no } h,e) = MB(\text{no } h,e) - MD(\text{no } h,e)$$

$$= 0 - \frac{P(\text{no } h|e) - P(\text{no } h)}{-P(\text{no } h)}$$

$$= \frac{(1 - P(h|e)) - (1 - P(h))}{1 - P(h)}$$

$$CF(h,e) = \frac{P(h) - P(h|e)}{1 - P(h)}$$

$$= MB(h,e) - MD(h,e)$$

$$= \frac{P(h|e) - P(h)}{1 - P(h)} - 0$$

o sea que:

$$CF(h,e) + CF(no\ h,e) = \frac{P(h|e) - P(h)}{1 - P(h)} + \frac{P(h) - P(h|e)}{1 - P(h)} = 0$$

Expresando de otra manera MB(h,e) = MD(no h,e). O sea que la evidencia que soporta una hipótesis, desfavorece su negación en el mismo grado, lo cual es intuitivo. Sin embargo, cuando la probabilidad a priori de una hipótesis es demasiado pequeña, el factor de certeza es aproximadamente igual a la probabilidad condicional.

$$CF(h,e) = \frac{P(h|e) - P(h)}{1 - P(h)} = P(h|e)$$

$$CF(no\ h,e) = \frac{P(h) - P(h|e)}{1 - P(h)} = - P(h|e)$$

Por lo tanto los factores de certeza no pueden por lo general ser considerados como probabilidades, salvo el caso particular mencionado.

Existen algunos problemas para aplicar la teoría Bayesiana en los sistemas de producción bajo incertidumbre. Estos problemas fueron los que llevaron a Shortliffe a desarrollar el modelo de creencias en MYCIN.

En algunos casos (sobretodo en el caso de medicina) es muy difícil llevar estadísticas que asocien evidencias con hipótesis. En otros casos es, casi imposible determinar las probabilidades asociadas a una regla de producción con conjunciones de premisas (evidencias compuestas). Más difícil aún es tratar de determinar las probabilidades inversas. Por ejemplo, la probabilidad de que una enfermedad presente ciertos síntomas. Incluso en el caso de que se pudieran obtener, estas llegarán a ser volúmenes muy grandes de información. Así, resultarían las preguntas: ?Son fidedignos? o ?Son realmente actuales?.

Además hay que cuidar que la máquina inferencial no se eternice calculando y propagando probabilidades.

4.3. FORMALIZACION.

La siguiente fase es la formalización [2]. Para esta etapa ya se ha determinado si vale la pena y si va a ser redituable la construcción del sistema. También se han determinado las metas que se persiguen, el alcance que se le pretende dar, así como las posibles áreas de aplicación. Se ha adquirido parte de la base de conocimientos y se ha elegido el esquema de representación más conveniente para el conocimiento involucrado. Hasta este punto las condiciones son óptimas para el diseño del SE. Sólo falta debe elegir también en esta fase el lenguaje de programación en el que se va a desarrollar este sistema. Esta fase consta de dos partes: i) Diseño del SE y ii) Programación.

4.3.1. DISEÑO DEL SISTEMA EXPERTO.

Ahora se puede proceder al diseño de la base de conocimientos y del esqueleto, en función de la representación elegida. Al llegar a esta etapa ya se conocen los recursos con que se cuenta y las necesidades que se tienen. Por lo tanto se pueden diseñar los diálogos, tipo de interacción con el usuario, y en que modo el sistema de cómputo (tiempo real, tiempo compartido, etc.). En consecuencia, el esqueleto del SE deberá tener ciertas características que son indispensables, como:

- El sistema deberá resolver problemas, cuyas conclusiones sean iguales a las que el experto humano llegaría, así como dar consejos apropiados en los momentos justos.

- No es suficiente que el sistema resuelva problemas, sino que debe tener la capacidad de explicar de alguna manera como llegó a las soluciones y que reglas o procedimientos siguió. Cuando el sistema proporcione explicaciones, deberá mostrar los caminos o líneas de razonamiento que se siguen en la base de conocimiento para llegar a alguna meta. Es decir, bajo una consulta deberá explicar que reglas y hechos utilizó para dar una respuesta.

- Debe tener un mecanismo que le permita hacer inferencias bajo condiciones inciertas y en algunos casos dadas por las evidencias se presentan de manera imprecisa, vaga o en ocasiones incompleta.

- El sistema debe ofrecer módulos que hagan fácil la interacción con el usuario. Pudiera también contener algún módulo que permita al usuario aportar evidencias de manera arbitraria, sin que tenga que esperar a que el sistema las pregunte. A este módulo se le denomina interpretador de evidencias.

- Se debe contar con algún módulo que nos facilite la tarea de adquisición de conocimiento. Esto es, algún editor que nos permita cambiar fácilmente las reglas y hechos que contenga la base de conocimiento. Este editor debe ser flexible y la estructura de la representación debe ser sencilla, dado que en el momento de desarrollo de un prototipo, la base de conocimientos estará sufriendo cambios. Por consiguiente sería conveniente que la base de conocimientos fuera lo más modular e independiente posible.

4.3.2. PROGRAMACION.

Es necesario seleccionar el lenguaje de programación en el que se va a desarrollar el SE. La selección del lenguaje depende en gran parte de la naturaleza del problema. Si el tipo de SE es de clasificación o diagnóstico, los lenguajes declarativos facilitan su desarrollo (PROLOG o LISP). Si por el contrario el problema involucra gran cantidad de cálculos numéricos o cuestiones de procedimiento, es conveniente utilizar un lenguaje procedural (PASCAL o C).

Conforme el desarrollo de aplicaciones en el área de inteligencia artificial, han surgido varios lenguajes. Estos lenguajes fueron diseñados para permitir el manejo simbólico de expresiones, manejo de listas y tienen la característica de ser declarativos. Un lenguaje declarativo nos permite el "QUE" de la programación evitando el "COMO". A continuación mencionaré algunos de estos lenguajes [16]:

IPL (Information Processing Language).

Este es uno de los primeros lenguajes de programación diseñados específicamente para inteligencia artificial (Newell, 1960). Su principal característica es el manejo de listas, sin embargo es más parecido a lenguaje de máquina que a un lenguaje de alto nivel.

LISP (LISt Processing).

Lisp (McCarthy, 1960) es hasta ahora el lenguaje más importante en desarrollos de inteligencia artificial. Sus principales características son:

- La estructura de datos más importante es la lista.
- Una colección de hechos acerca de un individuo pueden almacenarse mediante la lista de propiedades.

- La estructura de control màs natural es la recursiòn.
- Tanto los datos como los progrmas son almacenados en listas.
- La mayoría de las versiones son interactivas.

SAIL (System for Artificial Intelligence Language).

De los lenguajes de programaciòn de inteligencia artificial, SAIL (Swinehart, 1971). Es un lenguaje derivado de ALGOL, con algunas capacidades primitivas para el manejo de memoria asociativa. Dado que este es el lenguaje que màs se asemeja a los lenguajes tradicionales de proposito general, es el que màs se ha utilizado en aplicaciones que involucran gran cantidad de càlculos numèricos.

PLANNER.

Planner (Hewitt, 1971) es un lenguaje construido sobre LISP para manejar razonamiento hacia adelante o hacia atràs. Los programas en Planner tienen varios componentes:

- Hechos.
- Teoremas, los cuales describen como pueden inferirse nuevos hechos a partir de los ya existentes.
- Teoremas consecuentes, que describen razonamiento hacia atràs o dirigido a metas.

- Teoremas antecedentes, que describen razonamiento hacia adelante o dirigido a los datos.

KRL (Knowledge Representation Language).

Este lenguaje es también construido sobre LISP (Bobrow, 1977). KRL facilita la representación de conocimiento en estructuras de marcos jeràrquicos.

PROLOG (PROgramming in LOGic).

PROLOG (Colmerauer, 1973. Warren, 1977) es un lenguaje de reglas de producción, en el cual los programas son escritos como reglas para probar relaciones entre objetos. Cada programa de PROLOG està compuesto por un conjunto de cláusulas donde el intèrprete trata de encontrar pruebas de la veracidad de una determinada relación. Usualmente la relación contiene variables y parte del proceso de prueba involucra determinar para que instancias de las variables, la relación se convierte en verdadera.

Estos lenguajes quizà sean menos eficientes que los lenguajes de procedimiento, pero son màs claros y sobretodo ofrecen facilidades para el desarrollo de aplicaciones de inteligencia artificial, particularmente en el àrea de SE.

4.4. IMPLEMENTACION Y PRUEBAS.

Esta es la última etapa en la construcción y desarrollo de SE. En esta etapa se entra en un ciclo de pruebas y verificación, tanto del esqueleto como de la base de conocimientos.

Por una parte se debe verificar que el esqueleto efectúe las inferencias lógicamente, de acuerdo al algoritmo que se ha utilizado. El esqueleto debe de propagar la incertidumbre en la red de conocimiento que forman las reglas de inferencia. La propagación de estar acorde al modelo adoptado, tomando en cuenta todo tipo de situaciones y reglas formadas con cualquier tipo de conectivos lógicos. También debe explicar los resultados, mostrando las líneas de razonamiento seguidas en la red de inferencia para alcanzar una meta. Además se debe tener la capacidad de explicar como y porque se solicita un dato.

Por otra parte es conveniente desarrollar un prototipo de la base de conocimientos. Este prototipo deberá contener un número pequeño de reglas de inferencia, de modo que pueda ser depurado sin mayores problemas y se pueda mostrar el correcto funcionamiento del sistema. Las reglas que contenga el prototipo, aunque sean pocas, deben ser representativas de la base, para verificar al menos a grandes rasgos su comportamiento frente a situaciones extremas.

Otro detalle importante a cuidar en el desarrollo de la base de conocimiento es la consistencia de las reglas, para evitar contradicciones y redundancias. Por ejemplo, si del conjunto de reglas se deduce la hipótesis H y por otra línea de razonamiento se deduce no H, existe una inconsistencia en la de conocimiento.

Debe tenerse especial cuidado con los factores de certeza que el experto humano proporcione. Si existen dos o más hipótesis mutuamente exclusivas, los CF's que nos conducen a ellas no deben sobrepasar la unidad.

La representación del conocimiento elegida debe proporcionar suficiente flexibilidad para la verificación y refinamiento de la base de conocimiento. Además, se debe contar con un editor que nos permita hacer todos los cambios de una manera fácil y rápida.

4.5. SUMARIO.

En este capítulo se han descrito las fases principales en el desarrollo de un SE. Esta fases son: identificaión del problema, conceptualización, formalización, implantación y pruebas.

Se hace énfasis en la fase de conceptualización. En la sección (4.2.) se discuten algunos esquemas de representación de

conocimiento. Estas técnicas de representación son: redes semánticas, tripletas OAV, reglas de inferencia, marcos jerárquicos, etc. También se discuten dos modelos para la propagación de la incertidumbre: el modelo de probabilidad y el modelo de creencias.

Finalmente se mencionan varios de los lenguajes que más éxito han tenido en Inteligencia Artificial. Estos lenguajes presentan algunas ventajas para el desarrollo de SE. Algunas de sus características son:

- Son declarativos.
- Trabajan con listas.
- Tienen capacidades para hacer manejo simbólico.
- Son recursivos.
- Los programas y los datos tienen la misma estructura.

REFERENCIAS.

[1] **Forsyth, R.** 1985. Expert Systems. Chapman and Hall.

[2] **Hayes-Roth, F., D. A. Waterman, D. B. Lenat.** 1983. Building Expert Systems. Addison Wesley.

[3] **Hayes-Roth, F.** 1984. Knowledge Based Expert Systems. IEEE Computer. Octubre. pp 263-273.

[4] **Winston P. H.** 1981. Artificial Intelligence. Addison Wesley.

[5] **Elaine Kids.** 1984. Natural Language Interphases. IEEE Computer. pp 39-47.

[6] **Harmon P., D. King.** 1985. Expert Systems: AI in bussines. John Wiley and sons.

[7] **Hayes-Roth, F.** 1985. Rule-Based Systems. Communications of the ACM. vol 28 no. 9.

[8] **Clocksin W. F., C. S. Mellish.** 1984. Programming in Prolog. Springer Verlag.

[9] **Clarck K. L., F. G. Mc. Gabe.** 1985. Micro-Prolog: Programming in Logic. Prentice Hall International.

[10] **Shortliffe, E. H.** 1976. Computer Based Medical Consultations: MYCIN. New York. Elsevier Publishers Co.

[11] **Negoita, C. V.** 1985. Expert Systems and Fuzzy Systems. The Benjamin/Cummins Publishing Co. Inc.

[12] **Zadeh, L. A.** 1978. Fuzzy Sets as a Basis for the Theory of Possibility. Fuzzy Sets and Systems 1. pp 3-28.

[13] **Duda, R. O., P. E. Hart and Nils J. Nilson.** 1976. Subjective Bayesian Methods for Rule Based Inference Systems. National Computer Conference. SRI.

[14] **Elaine Rich.** 1983. Artificial Intelligence. Mc Graw Hill International Book Co. International Student Edition.

CAPITULO 5.

5. MODELO DE MYCIN IMPLEMENTADO EN PROLOG.

En este capítulo explicaré el conjunto de metareglas de PROLOG [1,2] que forman el esqueleto para SE. Este esqueleto llamado MYPRO, tiene las siguientes características:

- A partir de la base de conocimientos, debe hacer inferencias útiles para la solución de problemas.
- Debe explicar su comportamiento, mostrando al usuario las líneas de razonamiento seguidas para llegar a una meta.
- Tiene un mecanismo que le permite efectuar las inferencias aún en condiciones de evidencias inciertas o incompletas.

Además, MYPRO cuenta con otras características adicionales. Estas características son:

- El usuario puede aportar evidencias de manera voluntaria, sin necesidad de que el espere a que el sistema se las pregunte.

- Tiene comandos que nos permiten hacer un tratamiento adecuado de los diálogos. Con estos comandos, el usuario puede listar el diálogo, guardarlo en disco, cargar un diálogo de disco a memoria, etc.

- Permite listar la base de conocimientos.

5.1. MANEJO DE INCERTIDUMBRE.

Existen características que debe tener un esqueleto. De esas características dos son muy importantes: el manejo de razonamiento bajo incertidumbre y el seguimiento de las inferencias por las líneas de razonamiento. El esqueleto que desarrollé trata la incertidumbre con el modelo desarrollado por Shortliffe para el sistema MYCIN [3]. En el capítulo anterior veíamos que de acuerdo al modelo de Shortliffe, a cada regla de inferencia le podemos asociar un CF. Esto lo podemos representar en PROLOG como:

```
(Argumentos) Relación CF if
     Condición 1 y
     Condición 2 y

     . . .

     Condición n .
```

Por ejemplo:

```
(Juan aspirina) debe-tomar 0.8 if
     Juan sufre-de dolor &
     aspirina quita dolor &
     aspirina no-hace-mal-a Juan
```

5.1.1. MANEJO DE CLAUSULAS.

Dada una meta a alcanzar, nuestras metareglas deben localizar una regla que soporte esa hipótesis y determinar sus condiciones. Una vez determinado el grado de certeza con que se cumplen esas condiciones, se deben propagar las medidas de creencia hacia la meta.

Este conjunto de reglas manejarán la base de conocimientos en forma clausal. La regla confirmado nos transforma una pregunta en forma de frase a su forma clausal y verifica para què instancias y con que grado de certeza se verifica. Esa pregunta se convierte en nuestra meta. Esta regla en PROLOG es:

```
X confirma if
     Parse-of-SS(Y X ()) &
     Z isall((X!x) : Y conf x) &
     Z imprime
```

La regla conf en su versión más simple es:

```
(X|Y) conf (Z x) if
     ((X Y y)|z) CL &
     y descompone (X1 Y1) &
     z confCC (Z1 x1) &
     TIMES (X1 Z1 Z) &
     TIMES (Y1 x1 x)
```

El primer argumento es la meta a probar en su forma clausal, de modo que Y nos proporciona el nombre de los argumentos y X el nombre de la relación. En el segundo argumento se regresan las medidas de descreencia y creencia respectivamente, con las que se verifica esa relación.

Para determinar en qué medida se cumple una relación, primero debemos encontrar una cláusula que la soporte. Esto se logra mediante la relación primitiva CL, que tiene la forma:

```
((X Y Z)|x) CL
```

donde X es la relación, Y los argumentos, Z el CF y x la conjunción de las condiciones. Estas condiciones se probarán y se obtendrá el CF resultante mediante la relación confCC (confirma Conjunción de Condiciones). Por las funciones combinadas, el CF resultante es el CF de las condiciones multiplicado por el CF a priori de la regla. La regla confCC es:

```
() confCC (0 1)

(X!Y) confCC if
      X conf(y z) &
      Y confCC (X1 Y1) &
      MAX (y X1 Z) &
      MIN (z Y1 x)
```

Esta regla nos calcula recursivamente el CF de cada condición (por la relación **conf**, que a su vez llama a **confCC**) y calcula el CF resultante para la conjunción (el mínimo para MB y el máximo para MD, según las funciones combinadas).

5.1.2. TRATAMIENTO DE LA NEGACION.

Aquí el modelo comienza a funcionar, sin embargo falta considerar el tratamiento de las condiciones negadas como se muestra a continuación:

```
aspirina no-hace-mal-a Juan if
      aspirina agrava ulcera &
      not Juan tiene ulcera
```

Debemos entonces considerar que las reglas a confirmar pueden tener negaciones. Para ello hacemos uso de las funciones combinadas. Estas funciones nos dicen que si

```
MB(h,E) = X1      y     MD(h,E) = X2      entonces
MB(no h,E) = X2   y     MD(no h,E) = X1
```

La regla conf debe ser entonces modificada, añadiendo la acepción:

```
(NOT X!Y) conf (Z x) if
     (X!Y) conf (x Z)
```

5.1.3. CLAUSULAS CON DISYUNCIONES.

Asimismo, una clàusula puede tener disyunciones en sus condiciones. Por ejemplo:

```
relación (argumentos) if
     condición 1 &
     (either condición 2  or  condición 3)
     . . . etc.
```

La regla conf quedaría entonces de la siguiente manera:

```
(OR|X) conf (Y Z) if
     X confDC (Y Z)
```

La regla confDC (confirma disyunción de condiciones es análoga a la regla confCC, sólamente ahora se toma el máximo de los MB y el mínimo de los MD. esto es:

```
() confDC (1 0)

(X|Y) confDC (Z x) if
     X conf (z X1) &
     Y confDC (Z1 x1) &
     MAX (X1 x1 x) &
     MIN (z Z1 Z)
```

5.1.4. REGLAS CON MULTIPLES CLAUSULAS.

Una relación puede estar definida por varias cláusulas. O sea que una hipótesis sobre esa relación puede ser confirmada en una parte por ciertas evidencias y/o por conjunto de evidencias. Esto es, se puede decir que se tiene la evidencia en forma incremental.

Para tratar con evidencia incremental, utilizamos otra de las funciones combinadas (4.2.3.2):

$$MB(h,E1\&E2) = MB(h,E1) + MB(h,E2)[1 - MB(h,E1)].$$

De modo que la regla conf ahora tiene que checar si existen varias clàusulas y propagar los CF's adecuadamente. Para ello utilizamos la relaciòn primitiva CL en su versiòn de tres argumentos. En esta versiòn CL(X Y Z), X es la clàusula a buscar, Y es un paràmetro que le indica la clàusula inicial de busqueda y en Z nos regresa al lugar donde encontrò la clàusula.

Si deseamos checar una relaciòn, basados en la primitiva CL, comenzamos buscando clàusulas que la soporten desde la primera. Checar la siguiente, etc. y actualizar los CF's de acuerdo a las fòrmulas del modelo.

```
(X:Y) conf ((Z x) z) if
      conf-CL (X Y z X1 Y1) &
      SUM (z 1 Z1) &
      conf-sig-CL (X Y Z1 X1 Y1 x1 y1) &
      +^ (x1 X1 Z) &
      +^ (y1 Y1 x)
```

La regla conf-CL es igual que la anterior, sòlo que ahora le indicamos los límites de bùsqueda.

```
conf-CL (X Y x y z) if
          x VAR &
          (((X Y X1)! Y1) 1 x) CL &
          / &
          Y1 confCC (Z1 x1) &
          TIMES (X1 Z1 y) &
          TIMES (X1 x z)

     conf-CL (X Y x y z) if
          not x VAR &
          (((X Y X1)! Y1) x x) &
          / &
          Y1 confCC (Z1 X1) &
          TIMES (X1 Z1 y) &
          TIMES (X1 x1 z)
```

La regla conf-sig-CL checa si la anterior se cumplió. De ser asi, verifica la siguiente, si no, regresa CF = 0, que es el neutro de la operación +^.

```
conf-sig-CL (X Y x 0 0 0 0) if
     /

conf-sig-CL (X Y x y z X1 Y1) if
     (X!Y) conf ((X1 Y1) x)
```

5.2. RELACIONES PREGUNTABLES.

Hasta ahora he considerado sólo las relaciones que están

definidas por cláusulas en la base de conocimientos. Estas relaciones son nodos internos en la red de inferencias y los nodos hojas de esta red representan las relaciones que deben ser preguntadas al usuario.

De modo qe la regla conf debe contemplar que al no encontrar cláusulas que soporten una relación, entonces la cláusula debe ser preguntada al usuario. A la regla conf debe agregársele:

```
(X|Y) conf (Z x) if
     X es-preguntable &
     Parse-of-SS ((X|Y) y ()) &
     y es-reportado z &
     z descompone (Z x)
```

La meta-relación **es-preguntable** debe estar previamente definida y contener todas las relaciones susceptibles de ser preguntadas al usuario.

Para interactuar con el usuario, es conveniente regresar de nuevo a la forma de frases, usando la relación **Parse-ofSS** [4]. Por otro lado, el usuario deberá aportar el CF en un sólo número y debe ser descompuesto en su MB y su MD.

La relación **es-reportado** es la que se encarga de interactuar con el usuario. En esta regla se debe considerar que una evidencia puede ser utilizada por varias reglas y debe ser preguntada al usuario una sola vez.

Para llevar un registro de lo que el usuario ha aportado, se utiliza la relación **fue-confirmado**, donde guardamos la frase que confirmó junto con su CF. La relación **es-reportado** es:

```
X es-reportado Y if
     (fue-confirmado (X Y)) CL

X es-reportado Y if
     not (fue-confirmado (X Y)) CL &
     X es-dicho Y &
     ((fue-confirmado (X Y)) ADDCL
```

La relación **es-dicho** lee una palabra clave y enseguida verifica esa respuesta. Esta respuesta debe ser una de las siguientes:

solo : si sólamente se va a dar una respuesta. (evita el backtracking).

resp : si se van a seguir dando respuestas.

si,no: afirmaciones y/o negaciones rotundas. Una vez respondiendo de esta manera, no preguntará el CF, lo considera 1 y -1, respectivamente.

Enseguida se listan las relaciones **es-dicho** y **respuesta**:

```
X es-dicho (Y Z) if
     X no-tiene-vars &
     X lee-CF (Y Z) &
     /


Y es-dicho Y if
     P true-of X &
     ? P &
     Z R &
     respuesta(X Y Z)


respuesta (X Y solo Z ) if
     / &
     X compara &
     X lee-CF (Y Z)

respuesta (X Y resp Z ) if
     X compara &
     X lee-CF (Y Z)

respuesta (X Y Z x) if
     not Z EQ resp &
     P true-of (Z ignorado,teclear: si/no/resp/solo...)&
     PP &
     FAIL

respuesta (X Y Z x) if
     X es-dicho (Y x)
```

La regla **compara** verifica si la frase a preguntar tiene variables sin instanciar y si es así, las pregunta al usuario.

```
() compara if
     /

(X|Y) compara if
     X VAR &
     / &
     X R &
     Y compara

((X|Y)|Z) compara if
     (X|Y) compara &
     Z compara
```

```
(X|Y) compara if
      Y compara
```

5.3. MANEJO DE LINEAS DE RAZONAMIENTO.

En ocasiones es necesario explicarle al usuario las líneas de razonamiento. Esto es, darle a entender las reglas que serían usadas para llegar a cierta meta (razonamiento hacia adelante). Pero si usáramos razonamiento hacia atrás, justificar las reglas que estamos usando para hacer alguna pregunta y que meta perseguimos en ese instante.

5.3.1. RESPUESTAS A PORQUE.

Una línea de razonamiento es el conjunto de reglas que hemos utilizado para llegar a una meta. De modo que debemos contar con un mecanismo que nos permita ordenar las reglas que vamos utilizando. Si uso una pila, en el momento que el usuario me pida que le muestre la línea de razonamiento que se persigue, le listo la pila.

Como la regla conf es la que elige las reglas que vamos utilizando, es esa relación la que debe contar con el mecanismo de empilamiento de reglas. Para esto, agregamos a la regla conf

un argumento, que llevarà la pila de reglas en forma de una lista. Al principio la pila estarà vacía y se irà llenando conforme utilicemos màs reglas. La regla conf quedarà de la siguiente manera:

```
(X!Y) conf ((Z x) y z) if
      conf-CL (X Y y z X1 Y1) &
      SUM (z 1 Z1) &
      conf-sig-CL (X Y y Z1 X1 Y1 x1 y1) &
      +^ (x1 X1 Z) &
      +^ (y1 Y1 x)

conf-CL (X Y Z x y z) if
      (((X Y X1)!Y1) 1 x) &
      / &
      Y1 confCC((Z1 x1) (((X Y X1)!Y1)!Z1)) &
      TIMES (X1 x1 z)
```

Las reglas que forman una línea de razonamiento, deben ser mostradas al usuario. Estas reglas pasan como un parâmetro atravès de la regla **es-reportado** hasta la regla **es-dicho**, que es la que interactúa con el usuario.

La regla **es-dicho** lee una palabra clave y la regla respuesta la analiza. Si esa palabra clave es "porque", imprime la línea de razonamiento; si es diferente, ejecuta la acción pertinente. A la regla **respuesta** se le debe adicionar otra cláusula, que es:

```
respuesta (X Y porque Z) if
      / &
      Z son-impresos &
      X es-dicho (Y Z)
```

5.3.2. RESPUESTAS A PREGUNTAS COMO.

Habrán momentos durante la interacción con el usuario, en que se llegue a una meta determinada y es necesario hacerlo saber. En estos momentos el usuario puede estar interesado en conocer la línea de razonamiento (pueden haber varias disyuntivas) que llevó al sistema a tal conclusión. Consecuentemente, el sistema debe contar con un mecanismo que nos muestre tales líneas de razonamiento. Para ello se introduce la relación como:

```
X como if
     Parse-of-SS ((Y|Z) Z ()) &
     probado ((Y|Z) x y z)    &
     X muestra-como x         &
     (forall (X1 Y1) ON y then
          (Y demuestra-clausula X1 & Y1 son-explicados))
```

Esta regla utiliza la regla **probado**, la cual nos determina el CF con el cual se cumple esa meta y además encuentra el número de la cláusula que utilizó para probarlo. Enseguida la regla **muestra-como** explica como llegó a esa conclusión, explicando a su vez las condiciones de la cláusula que utilizó. A continuación listo las reglas **probado, muestra-como, no-se** y **es-demostrado.**

```
(X|Y) probado (Z x y) if
```

```
        x isall ((z X1) :
                 ((((X Y Y1)|X1) 1 z) CL &
                 (X|Y) conf ((Z1 x1) (1 y1)) &
                 SUM (Z Z1 x1))

X muestra-como 0 if
     X no-se & /

X muestra-como Y if
     X es-demostrado Y

X no-se if
     P ("No puedo asegurar nada acerca de que ") &
     P true-of X &
     P ("Se cumpla (CF = 0) " &
     PP &
     P(" Para hacerlo utilizaria la regla ") &
     PP

X es-demostrado Y if
     P true-of (Para probar que X) &
     P ("Se cumple con ") &
     P true-of (CF = Y ) &
     PP &
     P("utilize la regla ") &
     PP
```

La regla es-explicado justifica si la condición se probó en base a: reglas, negaciones de reglas, hechos, aportación directa del usuario, etc.

```
() son-explicados

(X|Y) son-explicados if
      X es-explicado (Z x) &
      Y son-explicados

(NOT|X) es-explicado (Y Z) &
      X es-explicado (Y Z) &
      TIMES (Z -1 x) &
      P ("O sea qie not ") &
      P true-of Y &
      P ("con CF = ") &
      PP X &
      /
```

```
(DR¦X) es-explicado (Y Z) if
     (DR¦X) probado (Z x y) &
     Parse-of-CC ((DR¦x) Y ()) &
     justifica (Z 1 z X1 y)
(X¦Y) es-explicado (Z¦x) if
     X dict &
     (X¦Y) Probado (x y z) &
     Parse-of-SS ((X¦Y) Z ()) &
     justifica (x 1 X1 Y1 Z)

(X¦Y) es-explicado (Z x) if
     X es-preguntable &
     Parse-of-SS ((X¦Y) Z ()) &
     ((fue-confirmado (Z x)) CL &
     justifica (X 2 y z Z) &
     /
```

La regla justifica, accesa una relación de patrones de respuesta para contestar al usuario de una manera más versátil y atractiva. Esta regla es:

```
justifica (X Y Z w y) if
     explica-CF (X Y Z w) &
     / &
     PP &
     P true-of Z &
     P true-of Y
```

y la relación de patrones de respuesta tiene la forma:

```
explica-CF   (0   1 ("No puedo asegurar nada acerca  de  que")
                                                  ("(CF = 0)"))
explica-CF   (X   1 ("Puedo demostrar que ")   ("con CF = " X))
...
...
... etc.
```

5.4. OTRAS CARACTERISTICAS.

A continuación se explican otras caracteristicas del esqueleto MYPRO. Este SE ofrece una amplia flexibilidad en el manejo del dialogo y el tratamiento de la base de conocimientos.

5.4.1. MANEJO DE EVIDENCIAS.

Una característica deseable es que el usuario pueda aportar las evidencias a voluntad, sin tener que esperar a que el sistema se las pregunte. De esta manera, cuando el sistema las requiera, ya las tiene disponibles y no necesita de preguntarlas al usuario, acelerando así el proceso.

En (5.2) se muestra la necesidad del almacenamiento del diàlogo en una relación, con el principal objetivo de no hacer preguntas repetidas al usuario. Pues bien, esta relación que nos mantiene el diàlogo, permite que el usuario pueda aportar evidencias de manera voluntaria.

En esta fase, el sistema cuenta con un aceptador de evidencias. Este aceptador verifica si las evidencias aportadas corresponden a relaciones preguntables. De ser así las añade al registro del diàlogo. Por ejemplo, si deseamos añadir la evidencia de que el escape humea demasiado (con CF = 0.7), podemos hacerlo de la manera siguiente:

```
&.hecho hay (hucho humo en el escape) 0.7
```

5.4.2. MANIPULACION DEL DIALOGO.

El sistema actualmente cuenta con varios comandos para el manejo del diàlogo. Como se viò en la secciòn anterior, el comando hecho nos permite adicionar hechos a voluntad. Podemos tambièn listar el diàlogo mediante el comando lista dialogo, como se muestra en el siguiente ejemplo:

```
&.lista dialogo
ME HAS DICHO QUE:
hay(mucho humo en el escape) CF = 0.7
...
```

Es tambièn muy conveniente tener la posibilidad de guardar un registro fijo del diàlogo, para consultas posteriores al sistema. Este registro lo podemos guardar en un archivo para consultas posteriores, mediante el comando:

```
&.guarda dialogo <nombre del archivo>
```

o bien cargarlo a memoria mediante el comando:

&.dialogo <nombre del archivo>

por último, podemos borrar el diálogo, por medio del comando:

&.elimina dialogo.

5.4.3. LISTADO DE LA BASE DE CONOCIMIENTOS.

El sistema cuenta también con un comando para listar toda o parte de la base de conocimientos. Con el comando

&.lista base

el cual nos lista la base de conocimientos completa y con el comando

&.lista <relacion>

el cual nos lista la relación especificada.

5.5. SUMARIO.

En este capítulo presento el desarrollo de un esqueleto de un SE en lenguaje PROLOG. Este esqueleto consta de un conjunto de meta-reglas que forman los diferentes módulos del SE. Primero desarrollé un pequeño esqueleto que hiciera inferencias. Enseguida le añadí la capacidad de tratar negaciones, disyunciones, etc. Una vez que fuincionó, le di la capacidad de inferir en condiciones de incertidumbre. Esto es, implementé el modelo desarrollado por Shortliffe para MYCIN. Finalmente desarrollé el módulo de explicación de las líneas de razonamiento. Y compruebo que este esqueleto tiene las tres características que mencioné al principio:

- hace inferencial que ayudan a la solución de problemas;
- tiene la capacidad de razonan bajo incertidumbre.
- puede explicar su comportamiento.

REFERENCIAS.

[1] Clocksin, W. F. and C. S. Mellish. 1984. Programming in Prolog. Springer Verlag.

[2] Clarck, K. L. and F. G. Mc Cabe. 1985. Micro-Prolog: Programming in Logic. Prentice Hall International.

[3] Shortliffe, E. H. 1976. Computer Based Medical Consultations: MYCIN. New York. Elsevier Publishing Co.

[4] F. G. Mc Cabe, K. L. Clarck and B. D. Steel. 1984. Micro-Prolog Reference Manual. Logic Programming Associates Ltd.

CAPITULO 6

6. RESULTADOS.

En este capítulo se muestran resultados obtenidos con dos bases de conocimiento simples, que permitan ver la flexibilidad del esqueleto.

6.1. ESQUELETO PROGRAMADO EN PROLOG: MYPRO.

La fabricación del esqueleto del SE MYPRO está basado en el modelo de MYCIN [1]. El esqueleto se programo en el lenguaje de programación PROLOG en la versión de micro-PROLOG [2], en una micro computadora IBM-PC con 256 Kbytes y bajo el sistema operativo MS-DOS. El sistema tiene aproximadamente 500 líneas de código PROLOG.

los resultados obtenidos con las dos bases de conocimiento mostraron excelentes respuestas que confirman la solidez del esqueleto. Además cuenta con varios módulos importantes y características que lo distinguen como SE.

- **Módulo interfase usuario.** Este módulo permite al sistema interactuar con el usuario. Esto es, si alguna relación no la encuentra en la base y sabe que es preguntable, interactúa con el usuario, considerándolo como una extensión de su base de conocimientos.

- **Modulo explicativo.** Este módulo se encarga de llevar el rastro de las líneas de razonamiento, de modo que pueda mostrárlas al momento que el usuario las solicite. Por ejemplo, en el transcurso de una consulta, el sistema puede hacer una pregunta que al usuario le paresca desconcertante. En este momento el usuario puede solicitarle al sistema que le diga porqué le está haciendo tal pregunta. El sistema le mostrará la línea de razonamiento seguida. Esto es, que metas persigue y que relaciones está utilizando para tratar de alcanzarla.

- **Módulo para el manejo de incertidumbre.** Este módulo le da al esqueleto la capacidad de hacer inferencias lógicas en condiciones de evidencias imprecisas o incompletas.

- **Módulo de manejo de diálogos y evidencias.** Este módulo le permite al esqueleto hacer la interacción con el usuario de manera más inteligente. Esto es, le permite llevar un registro del diálogo, de modo que el sistema no haga la misma pregunta dos veces. Le permite también llevar un registro externo del diálogo (en disco), para ser utilizado en consultas posteriores.

- **Módulo de adquisición de conocimiento.** Aunque no se desarrolló un módulo expreso para la adquisición de conocimiento, el mismo módulo que sirve para editar los programas en PROLOG nos sirve para editar la base de conocimientos en el proceso de adquisición y refinamiento.

Cabe mencionar que uno de los módulos más interesantes es el que maneja la incertidumbre. Para este módulo se utilizo el modelo de creencias de Shortliffe [1]. Los fundamentos se describen completamente en el capitulo 4 y su implementación en PROLOG en el capitulo 5.

6.2. RESULTADOS.

A continuación se muestran los resultados de una consulta a la base de conocimiento de medicina. Al cargar el programa aparece el siguiente desplegado y pide el nombre de la base de conocimientos:

```
ESQUELETO DE SISTEMAS EXPERTOS

MYPRO VERSION 1.0

AUTOR: JUAN JOSE FLORES ROMERO

MCMLXXXVI

ESQUELETO EN MEMORIA.
DAME EL NOMBRE DE LA BASE DE CONOCIMIENTOS KB
```

Por ejemplo, le podemos preguntar por una prescripción para un paciente, digamos Juan. O sea, ?qué debe tomar Juan?.

```
&.confirma(Juan debe-tomar x)
Juan sufre-de Y ? solo dolor
QUE TAN FUERTEMENTE CREES QUE JUAN SUFRE DE DOLOR ? 0.8
QUE TAN FUERTEMENTE CREES QUE JUAN TIENE ULCERA? porque
```

El sistema le pregunta de què sufre Juan y el usuario le contesta que tiene dolor con una certeza de 0.8. En este momento, el sistema le pregunta si Juan tiene úlcera y si el usuario desea saber porquè el sistema le està haciendo esta pregunta. El sistema le dice:

```
EXISTE  UNA  EVIDENCIA DE 0.9 DE QUE

aspirina  no-hace-mal-a Juan si
                aspirina agrava ulcera y
                not Juan tiene ulcera


EXISTE UNA EVIDENCIA DE 0.8 DE QUE

Juan debe-tomar aspirina si
                Juan sufre-de dolor y
                aspirina quita dolor y
                aspirina no-hace-mal-a Juan

QUE TAN FUERTEMENTE CREES QUE JUAN TIENE ULCERA ? no


EXISTE UNA EVIDENCIA DE 0.576 DE QUE

Juan debe-tomar aspirina

no hay (mas) evidencias de que

Juan debe-tomar aspirina
```

El usuario puede también preguntar como fué que se llegó a inferir algo:

```
&.como(Juan debe-tomar aspirina)

PARA PROBAR QUE
     Juan debe-tomar aspirina SE CUMPLE CON CF = 0.576
UTILIZE LA REGLA

EXISTE UNA EVIDENCIA DE 0.8 DE QUE X debe-tomar Y SI
          X sufre-de Z y
          Y quita Z y
          Y no-hace-mal-a X


TU ME DIJISTE QUE Juan sufre-de dolor con CF = 0.8

PUEDO DEMOSTRAR QUE aspirina quita dolor CON CF = 0.9

PUEDO DEMOSTRAR QUE
      aspirina no-hace-mal-a Juan CON CF = 0.72
```

Se puede en cualquier momento listar el diàlogo, por ejemplo:

```
&. lista dialogo

ME HAS DICHO QUE :

Juan sufre-de dolor CON CF = 0.8

Juan tiene ulcera CON CF = - 1
```

O bien, añadir hechos al registro del diàlogo voluntariamente.

```
&. hecho (Juan sufre-de diarrea) 0.5
```

6.3. DESARROLLOS FUTUROS.

Existe un gran número de proyectos que pudieran llevarse a cabo para complementar el trabajo que aquí presento. Donde los anexos que propongo podrían configurarse como módulos que funcionaran alrededor del esqueleto.

Una posible extensión es el desarrollo de una interfase usuario-máquina en lenguaje natural, que permitiera una interacción más amigable con el usuario. En este campo de la inteligencia artificial se han hecho muchos proyectos y trabajos [3,4,5]. Sin embargo, el problema del entendimiento del lenguaje natural con sus problemas de ambiguedades, sigue siendo un campo de interés para la investigación.

Otro posible módulo podría ser un interpretador de evidencias. Hasta ahora, el sistema cuenta solamente con un módulo que acepta que el usuario aporte hechos voluntariamente. Sin embargo, un interpretador de evidencias debe aceptar los hechos y determinar en base a ellos, qué metas perseguir. Esto es, un SE debe determinar las metas más costeables o quizá las que tengan mayor número de evidencias para aportar un resultado de manera más inmediata.

También, se espera contar con otro SE (de diferente arquitectura) a corto plazo. Este SE tendrà bases de conocimiento estratificadas en temas, que nos permitiràn elucidar mas ràpidamente la soluciòn del problema. El SE manejarà las evidencias del usuario para construir una estrategia de una forma inteligente y mas eficiente que los SE desarrollados [6].

Finalmente seria conveniente investigar la posibilidad de hacer intefase entre PROLOG y otros lenguajes de procedimiento, con el fin de poder desarrollar otro tipo de SE que requieran de càlculo numèrico.

REFERENCIAS.

[1] **Shortliffe, E. H.** 1976. Computer Based Medical Consultations: MYCIN. New York Elsevier Publishing Co.

[2] **Clark, K. L. and F. G. Mc Cabe.** 1985. Micro-Prolog: Programming in Prolog. Prentice Hall International.

[3] **Erman, L. D., P. E. London y S. F. Fickas.** 1981. The design and an example use of HEARSAY-III. En IJCAI 7, pp. 409-415.

[4] **King, Margaret.** 1983. Parsing Natural Language. Academic Press.

[5] **Forgy, C. C.** 1980. The OPS-5 User's Manual. Technical Report, Carnegie-Mellon University.

[6] **González Hernández Manuel.** 1986. Skeleton for an Expert System with a Knowledge Base Stratified in Themes. The 6th International Workshop on Expert Systems and their aplications. Agence de l'informatique. Avignon Francia.

CAPITULO 7.

7. CONCLUSIONES.

En la sección (4.2.2). se analizaron varios esquemas de representación de conocimiento. PROLOG cuenta con uno de ellos, el de reglas de inferencia. Este formalismo es un subconjunto de la lógica de predicados de primer orden y tiene por consiguiente una gran flexibilidad y poder expresivo. Esta representació tiene una sintaxis muy simple, de modo que un experto puede aprenderla y plasmar los conocimientos en la base de una forma relativamente sencilla. Además su modularidad nos facilita la tarea de adquisición y refinamiento de la base de conocimiento, que es muy inestable en toda la vida del SE.

En (4.2.3.) se discutieron dos modelos que permiten tratar con información imprecisa o incompleta. El modelo bayesiano y el modelo de creencias de Shortliffe. Se optó por implementar el modelo de creencias por sus ventajas respecto al otro.

El modelo bayesiano maneja cantidades muy grandes de información para poder calcular la propagación de la incertidumbre. Este modelo tiene la desventaja de que el cálculo de las probabilidades a priori de que ciertas evidencias confirmen una hipótesis es muy complicado. Además quizá para ciertos campos del conocimiento, los métodos estadísticos no nos lleven a resultados confiables.

En cuanto a la programación puedo afirmar que para aplicaciones en inteligencia artificial, los lenguajes declarativos nos proporcionan ventajas sobre los lenguajes de procedimiento. Algunas de ellas son:

i) Su estructura de datos principal son las listas. Esto facilita el manejo simbólico que la mayoría de los problemas de inteligencia artificial necesitan.

ii) En estos lenguajes se expresa el QUE y no el COMO de la programación. Esta característica permite al programador fijar su atención en la aplicación y no en los métodos o herramientas necesarias para su desarrollo.

iii) En el caso de PROLOG, se cuenta con características importantes para el desarrollo de SE, como el tener un algoritmo de unificación utilizado por el algoritmo de resolución. Este algoritmo es la máquina de inferencias con que cuenta PROLOG.

De lo anterior se desprende que las ventajas de programar en un lenguaje como PROLOG, son claras, sobretodo para aplicaciones de inteligencia artificial. Una de estas ventajas es la facilidad que ofrece para pasar casi directamente de las especificaciones del problema a los programas. Sin embargo, hay que acostumbrarse a pensar de una manera especial para programar problemas en PROLOG. Esto es, pensar en forma recursiva y declarativa.

Es conveniente mencionar que este es uno de los primeros

esqueletos escritos en PROLOG. Existen un conjunto de herramientas para el desarrollo de SE [1] hechas en PROLOG. Sin embargo, esas herramientas no tienen la capacidad para inferir bajo condiciones de incertidumbre.

REFERENCIAS.

[1] **Hammond Peter.** 1984. APES. Logic Programming Associates Ltd.

APENDICE A.

SUMARIO DE CARACTERISTICAS DE MICROPROLOG.

COMANDOS DE EDICION.

Los siguientes son comandos de edición de programas. Estos comandos están disponibles en el módulo SIMPLE del disco de micro-PROLOG para IBM-PC o compatibles.

COMANDO	SINTAXIS	
add	add (frase) add n (frase)	añade la frase al final de la lista de la relación.
delete	delete (frase) delete relación n	borra la frase de la relación.
list	list relación list all	lista una relación o todas.
save	save archivo	guarda el programa en disco.
load	load archivo	carga el programa a memoria.
kill	kill relación kill all	borra una relación o todo el programa.
edit	edit relación n	edita una relación.
QT	QT .	regresa al sistema operativo.

QUERIES.

En el módulo SIMPLE existen dos maneras de hacer queries a la base de datos. Una que simplemente confirma si una instancia particular de una relación se encuentra en la base o si puede ser deducida a partir de ella. La otra forma de interrogar la base es preguntàndole para què instancias se cumple cierta condición.

COMANDO	SINTAXIS
is	is(condición [& condición])
which	which(patrón de respuesta : condición [& condición])

RELACIONES ARITMETICAS.

El intèrprete de micro-PROLOG tiene las siguientes relaciones aritmèticas primitivas:

RELACION	SIGNIFICADO
SUM(X Y Z)	Z = X+Y
TIMES(X Y Z)	Z = X*Y
X LESS Y	X < Y

```
     X INT Y          Y = INT(X)

     X INT            Prueba si X es entero.
REGLAS.
```

El principal objetivo de PROLOG es obtener conclusiones de la base de datos. Por ejemplo sabemos que Pedro Padre-de Juan y Juan Padre-de María, son hechos en la base de datos. De ello podemos concluir que Pedro Abuelo-de María. En ocasiones es necesario abreviar queries repetitivos. Para lograr estos objetivos debemos declarar reglas. Por ejemplo:

```
X Abuelo-de Y if
     X Padre-de Z &
     Z Padre-de Y
```

Estas reglas nos definen instancias de la base de datos intensional (los hechos forman la parte extensional). Las reglas pueden estar a su vez definidas en base a otras reglas o incluso de manera recursiva y estar definidas por una o varias clausulas. Por ejemplo:

```
X Ancestro-de Y if
     X Padre-de Y

X Ancestro-de Y if
     Y Padre-de Z &
     Z Ancestro-de Y
```

LISTAS.

Una lista de un número arbitrario de elementos tiene la forma:

donde al primer elemento se le denomina cabeza y al resto cola. La cola de una lista es otra lista que puede llegar a ser la lista vacia (). micro-PROLOG cuenta con la construcción (X:Y), que nos permite trabajar con listas. En esta construcción, la X representa la cabeza e Y la cola. Ejemplo: La relación LONGITUD calcula la longitud de una lista.

```
() LONGITUD 0

(X:Y) LONGITUD Z <-
     Y LONGITUD X1 &
     SUM (X1 1 Z)
```

CONDICIONES COMPLEJAS.

Existen algunas primitivas que nos permiten expresar de manera fàcil ciertas condiciones que no son simples predicados y sin embargo se presentan muy frecuentemente. Estas primitivas son **not, or, isall y forall.**

- **not.:** Algunas condiciones se expresan màs naturalmente por la negaciòn de una cierta condiciòn que no se debe satisfacer. La sintaxis es:

  not C

  donde C es cualquier condiciòn o conjunciòn de condiciones, en cuyo caso se encierran entre parèntesis.

- **isall:** Esta primitiva permite meter a una lista las instancias de las variables que satisfacen ciertas condiciones. La sintaxis es:

  L isall (patròn de los elementos : condiciones)

donde L es la lista resultante, formada por las instancias que satisfacen las condiciones. La forma de los elementos de la lista está definida por un patrón.

forall. La sintaxis es:

(forall C1 then C2)

Esta condición se puede leer como: "para todas las x1, x2, ... , xk que satisfacen C1, entonces satisfacen C2".

or: Existen también condiciones que se expresan de manera natural utilizando disyunciones. La sintaxis del or es:

(either C1 or C2).

METAREGLAS.

En ocasiones es necesario hacer programas que manejen programas, èsto es, reglas que manejen reglas (metareglas). Por ejemplo, en el caso del esqueleto de un SE, el programa debe manejar ciertas reglas, para hacer inferencias y llegar a ciertas metas. Sin embargo, el programa no conoce el nombre de las relaciones que va a manejar, ni el nûmero de argumentos que tienen, entonces debe de manejar como variables el nombre de estas relaciones y sus argumento. Micro-PROLOG cuenta con la relaciòn true-of para el desarro-llo de metareglas. Esta relaciòn tiene la sintaxis:

variable o nombre de relaciòn **true-of**
variable o argumentos.

Si el primer argumento es variable, entonces debe tener un valor asignado al tiempo de que la condiciòn true-of es resuelta. Si el segundo argumento es variable, entonces representa cualquier lista de argumentos de la relaciòn. A estas variables se les denomina metavariables.

Por ejemplo, la condiciòn:

X true-of (x y) checa si x e y satisfacen alguna relaciòn X.

Existen otras primitivas que facilitan el desarrollo de metareglas o bien que en un momento dado permiten seleccionar las reglas a utilizar dependiendo de las condiciones de las metavariables. Estas primitivas son:

RELACION	SIGNIFICADO
X CON	Cierta si X está instanciada con una constante.
X VAR	Cierta si X no está instanciada.
X LIST	Cierta si X es una lista.
X NUM	Cierta si X está instanciada con un valor numérico.
X CL	Checa si la cláusula X está definida en la base de PROLOG.

Es conveniente hacer notar el carácter metalógico de la primitiva CL. Si hacemos una condición que la utilice, pero con variables no instanciadas, podemos conocer què cláusulas existen para una determinada relación, que argumentos y que condiciones tienen. Por ejemplo, la condición (((X!Y)!Z)) CL, con la variable X instanciada con el nombre de una relación, obtendríamos en Y la lista de argumentos y en Z la lista de condiciones de esa relación.

LA SINTAXIS CLAUSAL DE micro-PROLOG.

Los programas que se han discutido están escritos en una sintaxis especial, que nos proporciona el módulo SIMPLE. Este módulo es un intérprete de la sintaxis estándar de micro-PROLOG.

En la sintaxis estándar, una frase se compila a una cláusula y la condición e hipótesis de la cláusula forman átomos. Una regla en sintaxis simple se convierte entonces en una lista de átomos, donde el primer átomo es la hipótesis y el resto son las condiciones. En seguida se muestran algunas frase en sintaxis simple y sus respectivas cláusulas:

FRASE	CLAUSULA
Juan padre-de Maria	((padre-de Juan Maria))
SUM(1 2 3)	((SUM 1 2 3))
X es-hombre	((es-hombre x))
(X|Y) long Z if Y long x & SUM(x 1 Z)	((long (X|Y) Z)(long Y x) (SUM x 1 Z))

Existe un conjunto de primitivas que nos permiten programar en sintaxis estandar. Enseguida se muestran algunas primitivas con su sintaxis.

RELACION	SIGNIFICADO
ADDCL X	Añade la cláusula X.
DELCL X	Elimina la cláusula X.
?((DELCL Relación n))	Elimina la n-ésima cláusula de la relación.
LIST ALL	Lista todas las cláusulas.
LIST Relación	Lista las cláusulas de la relación.

La relación "?" es una primitiva para hacer queries. Esta relación es unaria y toma como argumento una conjunción de condiciones. Por ejemplo:

```
&.?((padre-de Juan Luis)(padre-de Luis Maria))
&.
```

Si el query fue resuelto, no nos indica nada. Si no, imprime una interrogación.

Todas las relaciones que nos sirven para hacer queries están definidas a partir de esta relación. Por ejemplo la relación is y la relación which se pueden definir en base a "?" :

```
((is X) (? X) / (PP YES))
((is X) (PP NO))

((which (X : Y))
     (FORALL Y ((PP X)))
     (PP No (more) answers))
```

La condición / de la relación is hace que el backtracking se elimine, a menos de que la condición (? X) falle.

Como aplicación, haremos la metarelación APLICA, que aplica la operaión que le indiquemos a una lista de números. Por ejemplo:

```
&.which(x : APLICA((2 3 4) SUM x))
  9

&.which(x : APLICA((2 3 4) TIMES x))
  24
```

La relación aplica se puede escribir en PROLOG como:

```
APLICA(() Y X)

APLICA((X|Y) Z x) if
     APLICA(Y Z y) &
     Z true-of (X y x)
```

APENDICE B.

LISTADO DE LAS BASES DE CONOCIMIENTO.

1. BASE DE CONOCIMIENTO DE MEDICINA.

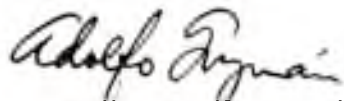
```
sufre-de es-preguntable
tiene    es-preguntable

(X Y) debe-tomar 0.8 if
      Y sufre-de Z and
      Y quita Z and
      Y no-hace-mal-a X

(X Y) no-hace-mal-a 0.9 if
      X agrava Z and
      not Y tiene Z

(aspirina dolor)  quita 0.9
(lomotil diarrea) quita 0.85

(aspirina ulcera)                        agrava 0.8
(lomotil enfermedades-pancreaticas) agrava 0.95
```

2. BASE DE CONOCIMIENTO DE MECANICA AUTOMOTRIZ.

```
es-dificil-de es-preguntable
hay es-preguntable
han-sido es-preguntable
llega-combustible es-preguntable
arranca es-preguntable

esta-bloqueado (((el inyector de arranque)) 0.6) if
     es-dificil-de (arrancar el carro) and
     la-mezcla (es pobre .)
esta-bloqueado (((el inyector de arranque)) 0.7) if
     not arranca (el carro .) and
     not llega-combustible (a la maquina) and
     not llega-combustible (a la camara de flotacion)

esta-bloqueado (((los inyectores principales)) 0.8) if
     es-dificil-de (revisar la maquina) and
     la-mezcla (es pobre .)

esta-bloqueado
    (((el tubo de la bomba a la camara de flotacion)) 0.9) if
     not arranca (el carro .) and
     not llega-combustible (a la bomba)

esta-bloqueado (((el tubo del tanque a la bomba)) 0.9) if
     not arranca (el carro .) and
     not llega-combustible (a la bomba)

mezcla (((pobre por ajuste incorrecto del carburador)) 0.8) if
     la-mezcla (es pobre .)
mezcla (((rica por ajuste incorrecto del carburador)) 0.8) if
     la-mezcla (es rica .)

falla (((la valvula de aguja en la camara de flotacion)) 0.6) if
     alto (nivel de combustible en la camara de flotacion)
falla (((la valvula de aguja en la camara de flotacion)) 0.5) if
     la-mezcla (es pobre .)
falla (((la valvula de aguja en la camara de flotacion)) 0.6) if
     not arranca (el carro .) and
     not llega-combustible (a la camara de flotacion) and
     llega-combustible (al carburador .)
falla (((los inyectores .)) 0.4) if
     la-mezcla (es rica .)
falla (((los inyectores .)) 0.8) if
     (either
          alto (kilometraje recorrido) or
          han-sido (limpiados con alambre los inyectores))
falla (((la bomba de la gasolina)) 0.5) if
     la-mezcla (es pobre .)
falla (((la bomba de la gasolina)) 0.9) if
     not arranca (el carro .) and
     not llega-combustible (al carburador .) and
     llega-combustible (a la bomba)
```

```
la-mezcla ((es pobre .) 0.7) if
     es-dificil-de (arrancar el carro)
la-mezcla ((es pobre .) 0.3) if
     hay (deposito gris en las bujias)
la-mezcla ((es pobre .) 0.5) if
     hay (sobrecalentamiento en la maquina)
la-mezcla ((es pobre .) 0.8) if
     arranca (el carro .) and
     hay (falta de potencia en la maquina)
la-mezcla ((es rica .) 0.2) if
     hay (falta de potencia en la maquina)
la-mezcla ((es rica .) 0.6) if
     hay (alto consumo de combustible)
la-mezcla ((es rica .) 0.8) if
     hay (explosiones en el escape)
la-mezcla ((es rica .) 0.5) if
      hay (mucho humo en el escape)

alto ((nivel de combustible en la camara de flotacion) 0.8) if
     la-mezcla (es rica .)
```

Los miembros del jurado, designado por la Sección de Computación del Departamento de Ingeniería Electrica, certifican que han leído esta tesis y que es completamente adecuada, en contenido y calidad, como disertación para obtener el grado de Maestro en Ciencias.



. . . . . .
Dr. Adolfo Guzmán Arenas

. . . . . .
M. en C. Manuel González Hernández

. . . . . .
Dr. Felipe de J. Contla y Martínez del Río

. . . . . .
Dra. Ana María Martínez Enríquez

MEXICO, D. F., FEBRERO DE 1986.

AUTOR FLORES ROMERO, JUAN JOSE

TITULO ESQUELETO DE UN SISTEMA EXPERTO EN PROLOG.

CLASIF. XI'. 86.1

ROTRO 81-10,235

NOMBRE DEL LECTOR	FECHA PREST.	FECHA DEVOL.
HECTOR SALDAÑA A.	6.6.92	7/VII/92
BC Maestría	21-4-87	21 MAY 87
BC Doctorado	22-4-87	29/IV/87
Alumno Maling	22.04.87	25/IV/87
Copias Xerox	30/IV/87	1/IX/87
Dra Edith Manuella E.	1/IX/87	2/IX/87
Dra. J Flores S.	5/IX/87	24 OCT 87
[illegible]	[illegible]	2 MAY 88
Juan Carlos Cham P.	16/IX/88	2 FEB 89
Juan Carlos Cham P.	13/II/89	7 MAR 89
[illegible]	[illegible]	18 ABR. 89
[illegible]	[illegible]	[illegible]