Desarrollo de sistemas inteligentes

Copyright © 2022 Juan Marín Noguera, juan.marinn@um.es.
Esta obra está bajo la licencia Reconocimiento-CompartirIgual 4.0 Internacional de Creative Commons (CC-BY-SA 4.0). Para ver una copia de esta licencia, visite https://creativecommons.org/licenses/by-sa/4.0/.

Bibliografía:

- Apuntes de clase.
- Wikipedia, la Enciclopedia Libre. *Hipónimo*, *Hiperónimo*, *Meronimia*, *Holonimia*. Recuperado de https://es.wikipedia.org/ el 25 de septiembre de 2022.
- Diccionario de la lengua española. *Lexicón*. Recuperado de https://dle.rae.es/lexic%C3%B3n el 25 de septiembre de 2022.
- Charles L. Forgy. *Rete: A Fast Algorithm for the May Pattern/Many Object Pattern Match Problem* (1981).
- Gordon S. Novak Jr. *TMYCIN: Tiny EMYCIN-like Expert System Tool* (2006). University of Texas at Austin.
- Wikipedia, the Free Encyclopedia. *T-norm*. Recuperado de https://en.wikipedia.org/ el 6 de diciembre de 2022.
- Rajesh Kr. Singh, Amitkr. Arya, M. Z. Alam (P.G. Dept. of Mathematics, College of Commerce, Arts & Science, Patna (Bihar), India). *Convex Fuzzy Set, Balanced Fuzzy Set, and Absolute Convex Fuzzy Set in a Fuzzy Vector Space* (2016). IOSR Journal of Mathematics, Volume 12, Issue 2, Ver. VI, pp. 17–24. Recuperado de https://iosrjournals.org/iosr-jm/papers/Vol12-issue2/Version-6/D1202061724.pdf.

Capítulo 1

Ontologías

La **ontología** es una rama de la metafísica que busca una explicación sistemática de lo que hay, qué entidades existen y cuáles no, y plantea preguntas como qué son las cosas, si existen los conceptos fuera de nuestra mente, cuál es la esencia de las cosas que permanece a través de los cambios o cómo clasificar entidades del mundo real.

Una **conceptualización** es un modelo abstracto de un fenómeno real mediante la identificación de los conceptos relevantes de este. Una **ontología** es un sistema de categorías para sistematizar cierta visión del mundo, y define los términos básicos y relaciones que componen el vocabulario de un cierto tema junto con reglas para combinar términos y relaciones con los que definir extensiones al vocabulario. Es una especificación explícita en un lenguaje formal de una conceptualización compartida y aceptada por un grupo.

Desde los 90, las ontologías se han consolidado como parte de la inteligencia artificial, pues permiten comprender los aspectos básicos del dominio, aseguran que todas las partes implicadas lleguen a un acuerdo sobre cómo describir los conceptos y permiten reutilizar el conocimiento entre distintas aplicaciones al representarlo de forma declarativa.

Según la riqueza de su estructura, las ontologías pueden ser:

1. **Vocabularios controlados:** Listas finitas de términos, como un catálogo.
2. **Glosarios:** Listas de términos junto a sus significados en lenguaje natural.
3. **Taxonomías informales:** Listas de términos organizados en una relación «es un» no estricta.
4. **Tesauros:** Listas de términos que representan conceptos, temas o contenidos de documentos y mejoran el acceso a los mismos. Pueden incluir **relaciones jerárquicas** a modo de taxonomía informal; **de equivalencia** como sinonimia, homonimia, antonimia y polisemia, y **asociativas**, que mejoran las estrategias de recuperación de datos y ayudan a reducir la cantidad de jerarquías entre términos.
5. **Taxonomías formales:** Incluyen una jerarquía de clases, de modo que si B es subclase de A, toda instancia de B lo es también de A, y puede incluir instancias concretas de las clases.

6. **Basadas en *frames*:** Como taxonomías formales pero las clases tienen propiedades o atributos, que se heredan. Una propiedad es **intrínseca** si es inherente a la instancia, y es **extrínseca** en otro caso, como ocurre con el nombre o la procedencia.

7. **Con restricciones para los valores:** Como las basadas en *frames* pero pudiendo restringir los valores que pueden adoptar los atributos.

8. **Con restricciones lógicas generales:** Las más expresivas. Permiten definir restricciones arbitrarias sobre los conceptos e instancias.

Según lo que representan:

1. **Para la representación del conocimiento,** como *Frame Ontology*, *RDF Schema* u *OWL Reference*, que describen primitivas para modelar y formalizar conocimiento siguiendo un paradigma de representación.

2. **De alto nivel,** que describen conceptos muy generales e indicaciones para organizar los conceptos raíz en otras ontologías. Algunas son `ontology4.us`, `schema.org` y la *IEEE Standard Upper Ontology* (SUO), creada con la idea de consensuar una ontología común y que dio lugar a la *Suggested Upper Merged Ontology* (SUMO).

3. **De dominio,** como las de *DERI vocabularies* y *Ontohub Repository*, que representan descripciones reutilizables en un cierto dominio (médico, legal, ingeniería, etc.), con un vocabulario sobre los conceptos, relaciones y actividades del dominio y las teorías y principios que lo rigen. En general sus conceptos son instancias de las ontologías de alto nivel.

4. **De métodos,** para describir cómo se resuelve un cierto problema.

5. **De tareas,** como *Scheduling Task*, con vocabulario sobre una tarea genérica o actividad (diagnóstico, planificación, etc.), para describir un problema independiente del dominio. Puede tener asociadas ontologías de métodos.

6. **De aplicación,** como las de *Ontohub Repository*, con las definiciones necesarias para modelar el conocimiento requerido en una cierta aplicación, y que se suelen crear extendiendo ontologías de dominio y de tareas.

Una **clase** es una descripción formal de un concepto del universo o dominio a representar. Las clases se organizan en jerarquías o taxonomías con la relación «es un» de subclase. Para decidir qué clases considerar, se tienen en cuenta los objetivos de la ontología. Una **instancia** es un objeto de una clase. La **base de hechos** es la colección de todas las instancias. Una instancia es **indirecta** respecto a una clase si pertenece a una subclase directa o indirecta de esta. Las **clases abstractas** no pueden tener instancias directas, y se usan para agrupar conceptos e introducir cierto orden en la jerarquía.

Las clases pueden tener **propiedades**, **atributos** o ***slots***, que representan características que permiten describir más detalladamente la clase y sus instancias. Los valores de las propiedades pueden ser de tipos básicos como cadenas de caracteres o números.

Una **relación** es una asociación entre conceptos de distintas clases, $R \subseteq C_1 \times \cdots \times C_n$, donde cada C_i es el conjunto de instancias directas o indirectas de una clase. Normalmente las ontologías se basan en relaciones binarias, con dos clases llamadas **dominio** y **rango** de

la relación. Las relaciones pueden tener atributos y formar parte de taxonomías, y entonces se suelen definir como conceptos. Una **función** $F : C_1 \times \dots C_{n-1} \rightarrow C_n$ es una relación $F \subseteq C_1 \times \cdots \times C_n$ que no contiene tuplas que difieran solo en su último elemento.

Una **restricción**, **faceta** o ***facet*** es una propiedad de las relaciones, conceptos y atributos, como cardinalidad, obligatoriedad, valores por defecto, etc.

Un **axioma** es una sentencia lógica que siempre debe ser cierta y no puede ser modelada por los elementos anteriores. Los axiomas sirven para verificar la consistencia de la ontología o del conocimiento almacenado y para inferir nuevo conocimiento.

Algunas ontologías:

1. Las **ontologías lingüísticas** definen estructuras semánticas, y no un dominio específico. *WordNet* es una gran base de datos con información léxica del inglés basada en teorías psicolingüísticas. Organiza la información según el significado de las palabras y no su forma, aunque tiene en cuenta inflexiones morfológicas, y consta de unos 70000 conjuntos de sinónimos o *synsets*, organizados en las categorías de nombres, verbos, adjetivos, adverbios y palabras funcionales, y relaciones entre ellos de sinonimia, antonimia, **hiponimia** (subclase), **hiperonimia** (superclase), **meronimia** (parte de) y **holonimia** (tiene como parte).

2. Las **ontologías para el comercio electrónico** dan marcos de definiciones para identificar productos y servicios e intercambiar información entre proveedores y consumidores. **UNSPSC** (***United Nations Standard Products and Services Codes***) es una codificación estándar para facilitar el comercio electrónico intercambiando descripciones de productos y servicios según una taxonomía de 5 niveles: segmentos, familias, clases, productos y función empresarial. Cada elemento de cada nivel tiene un código numérico y una descripción. La versión 16.0901 contiene 55 segmentos y unos 50000 productos.

3. Las **ontologías médicas** permiten intercambiar información médica entre distintos agentes, como datos de pacientes, usando una terminología médica clara y precisa, y proporcionan criterios semánticos para propósitos estadísticos.

   a) ***Gene Ontology*** es un modelo para comunicar información de sistemas biológicos como estructura, proceso biológico, genes, proteínas, etc.

   b) ***The OBO foundry*** contiene muchas ontologías sobre medicina y biología.

   c) **UMLS** (***Unified Medical Language System***) es una base de datos de términos de distintos sistemas de clasificación y vocabularios médicos como MeSH o SNOMED. Consta de una **red semántica**, una ontología de alto nivel de conceptos biomédicos y relaciones entre ellos para resolver problemas de ambigüedad; un **meta-tesauro**, con conceptos asociados a uno o más conceptos de la red semántica, información asociada como definición o fuente, y relaciones como sinónimos o conceptos hijos y padres, y un **lexicón especializado**, con información sintáctica sobre los términos biomédicos para procesamiento del lenguaje natural.

4. Las **ontologías para ingeniería** describen modelos matemáticos usados por científicos para analizar sistemas físicos, permitiendo compartir y reutilizar dichos modelos entre ingenieros y herramientas. **EngMath** es un conjunto de ontologías del servidor **Ontolingua** que incluye las bases conceptuales para describir cantidades escalares, vectoriales

y tensoriales, funciones y unidades de medida, proporcionando un vocabulario para representar modelos y teorías de ingeniería y compartirlas entre distintas herramientas de software y sentando las bases para ontologías y lenguajes más específicos. Algunas de las ontologías de EngMath son:

a) **Álgebra abstracta**, con descripciones de grupos, anillos, dominios, cuerpos, etc. con sus operadores.

b) **Medidas físicas**, para representar medidas de aspectos cuantificables con 11 clases, 3 relaciones, 12 funciones como suma, producto, cociente, etc. y 2 instancias.

c) **Dimensiones estándares**, con 27 instancias organizadas en 18 clases como masa, longitud, temperatura, etc.

d) **Unidades estándar**, con una clase que representa una unidad del Sistema Internacional de Unidades y 60 instancias como kilo, metro, etc.

e) **Cantidades escalares**, para modelar cantidades con magnitudes reales, con una sola clase y 6 funciones que especializan las funciones de la ontología de medidas físicas.

Se han escrito ontologías para casi todos los dominios, como empresarial, gestión del conocimiento, inteligencia ambiental, sistemas multiagente, etc. La web semántica ha permitido la creación de importantes repositorios de ontologías al facilitar la reutilización.

Capítulo 2

Desarrollo de ontologías

Una **metodología** es una teoría general sobre cómo desarrollar cierto tipo de trabajo que integra una serie de técnicas y métodos comprensibles. Desde los 90 ha ido creciendo el interés en tener una metodología estructurada para acelerar el desarrollo de ontologías, la reutilización y el uso en distintas aplicaciones. La primera *workshop* de ingeniería ontológica se celebró en el ECAI de 1997, y se continuó en en el *AAAI Spring Symposium*. Desde entonces la mayoría de conferencias internacionales sobre el tema incluyen aspectos metodológicos de desarrollo de ontologías.

La ingeniería ontológica engloba:

1. **Re-ingeniería:** reutilización de ontologías expresadas en formalismos distintos.
2. **Aprendizaje:** Métodos automáticos para desarrollar o poblar ontologías.
3. **Alineación:** Enlace de ontologías sobre un mismo dominio manteniendo las originales.
4. **Fusión:** Creación de una única ontología a partir de varias.
5. **Desarrollo distribuido y colaborativo:** Definición de protocolos para añadir y consensuar conocimiento nuevo.
6. **Evaluación:** Control de calidad del producto.

No hay una única forma correcta de modelar un dominio, pues la mejor solución en general depende de la aplicación y las extensiones previstas, y debemos preguntarnos para qué la vamos a usar, cómo de detallada debe ser, qué alternativa funcionará mejor en la aplicación, cuál será más intuitiva, extensible y mantenible, etc., y después hay que evaluar esta versión inicial, depurarla y consensuarla con expertos. Los conceptos de una ontología deben ser cercanos a los objetos físicos y lógicos y las relaciones del dominio, y generalmente se corresponden con sustantivos o verbos que lo describen.

Una metodología permite desarrollar eficientemente ontologías complejas consistentes de forma distribuida. Una de ellas es **desarrollo de ontologías 101**, con los siguientes pasos:

1. **Determinar el dominio y alcance de la ontología.** Determinar qué dominio cubrirá, para qué la usaremos, a qué tipo de preguntas deberá responder y quién las usará y mantendrá. Esto puede variar durante el desarrollo, y una forma de determinar el alcance

es elaborar un conjunto de **preguntas de competencia** que la ontología debe poder responder. Este no tiene que ser exhaustivo, y se puede usar al final para control de calidad.

2. **Considerar la reutilización de ontologías existentes.** Si hay otras ontologías sobre nuestro dominio para otras aplicaciones, como las de Ontolingua, las ontologías DAML, las de Protégé y muchas otras disponibles en la web, estas se pueden reutilizar. De hecho podríamos tener que usarlas si queremos que nuestra aplicación interactúe con otras que usen alguna de estas ontologías.

3. **Enumerar los términos importantes.** Escribir los términos que queremos usar para hacer enunciados o dar explicaciones al usuario. No hay que preocuparse por cubrir todos los términos, ni por sus relaciones, propiedades o si estos son clases o atributos.

4. **Definir las clases y su jerarquía.** Podemos usar un enfoque **descendente** (***top-down***), en el que se empieza definiendo conceptos más generales para luego ir especializándolos; **ascendente** (***bottom-up***), en que se empieza con las clases más específicas para luego ir agrupándolas, o lo que es más común, **combinado**, en que se empieza por los conceptos más relevantes para luego especializarlos y generalizarlos según convenga, ya que los conceptos más descriptivos suelen ser los intermedios.

   El enfoque depende de la visión que el desarrollador tenga del dominio, y en cualquier caso las clases iniciales se toman de los conceptos del paso 3 que tengan existencia independiente, que corresponden a predicados unarios sobre objetos, no predicados binarios, y entonces estos se organizan taxonómicamente.

   Hay muchas jerarquías válidas para el mismo dominio según el uso que se vaya a dar a la ontología. Recomendaciones:

   a) **Asegurar que la jerarquía es correcta,** de modo que toda instancia de una subclase lo es de su superclase, de forma transitiva. Las jerarquías de clases pueden variar según el dominio evoluciona y mantenerlas es complejo. El nombre de una clase puede cambiar al cambiar la terminología siempre que la clase represente el mismo concepto, y los sinónimos de un concepto no representan clases distintas. Se deben evitar los bucles en la jerarquía, que equivalen a decir que las clases en el bucle son equivalentes.

   b) **Analizar las clases hermanas,** subclases directas de la misma clase. Todas deben tener el mismo nivel de generalidad, salvo en el nivel superior. No hay reglas sobre cuántas clases directas puede tener una clase, pero si una clase tiene solo una subclase directa, puede haber un problema en el modelado o la ontología puede ser incompleta, y si tiene más de 12 puede haber que definir categorías intermedias, aunque si no hay clases naturales para agrupar estos conceptos no hace falta crear clases ficticias.

   c) **Considerar herencia múltiple.** La mayoría de sistemas de representación la permiten, pero debe usarse con cuidado. La subclase hereda los atributos y facetas de todas sus superclases.

   d) **Introducir subclases.** Se hace sólo si hay algo que podemos decir de ella que no podamos decir de la superclase. En general las subclases tienen propiedades o

restricciones que la superclase no tiene o participan en relaciones en que la superclase no participa. En la práctica las subclases deben añadir nuevos atributos respecto a la superclase, nuevos valores para un atributo o sustituir algunas facetas de los atributos heredados

Crear nuevas clases que no introducen nuevas propiedades puede ser útil para modelar diferencias entre conceptos aunque no se vaya modelar la distinción en sí, facilitar la definición del nivel de generalidad de cada término y mejorar la exploración y la navegación. No se debería crear una subclase para cada restricción adicional, sino que se busca un equilibrio en que se crear clases útiles para organizar la jerarquía sin crear demasiadas clases.

e) **Diferenciar entre clase y atributo.** En general, si una distinción es importante en el dominio y pensamos en objetos con distintos valores de esta como de distintos tipos, o si los conceptos con distintos valores de esta se vuelven restricciones para atributos o relaciones de otras clases, la distinción se representa mediante subclases, y en otro caso se usa un atributo. Si la distinción entre clases se hace por propiedades extrínsecas a los conceptos y no intrínsecas, como suele ser el caso de números, colores, ubicaciones, etc., las instancias tendrán que cambiar de subclase a menudo, lo que no debería ocurrir, por lo que estas distinciones se representan con atributos.

f) **Diferenciar entre clase e instancia.** El nivel más bajo de granularidad depende del alcance de la ontología. Las instancias son los conceptos más específicos representados, y a veces pueden desplazarse al conjunto de clases, por ejemplo cuando forman una jerarquía natural, en cuyo caso puede ser útil definir clases abstractas.

g) **Limitar el alcance.** La ontología no debe ni puede contener toda la información del dominio. No se debe especializar o generalizar más de lo requerido por la aplicación, como mucho un nivel adicional a cada lado, y no debe contener todas las posibles propiedades de las clases y distinciones posibles en la jerarquía.

h) **Considerar clases disjuntas.** Algunos sistemas de representación permiten definir clases disjuntas, de modo que ninguna instancia puede ser miembro de ambas. Estas facilitan validar la ontología.

5. **Definir los atributos de las clases.** La mayoría de los términos que queden del paso 3 serán propiedades de clase, y hay que indicar qué clase es descrita por cada propiedad. Los propiedades pueden ser intrínsecas, extrínsecas o relaciones con otros individuos.

6. **Definir las facetas de los atributos.** Algunas son:

   a) **Cardinalidad.** Algunos sistemas sólo permiten indicar si esta es simple o múltiple, mientras que otros permiten indicar máximo y mínimo y a veces se puede indicar un mínimo de 0 para señalar un atributo como opcional.

   b) **Tipo de valor.** `String`, `Number` (`Float` o `Integer`), `Boolean`, `Enumerated` con una lista de valores admitidos, `Instance` para instancias de otros individuos, en cuyo caso el atributo representa una relación. En Protégé-2000 los enumerados se definen como instancias de tipo `Symbol`.

   c) **Dominio.** Conjunto de clases descritas por el atributo. En Protégé-2000 no hay que definirlo por separado.

d) **Rango.** Conjunto de clases al que pueden pertenecer las instancias si el atributo es de tipo `Instance`.

e) **Valor por defecto.**

Al definir el dominio o rango, se deben usar las clases más genéricas posibles, pero no demasiado genéricas, pues todas las clases en el dominio deben poder incluir un valor para el atributo y todas las del rango deben ser posibles valores de este. Si una lista de clases en el dominio o rango incluye una clase, no debería incluir ninguna de sus subclases; si incluye todas las subclases de una clase pero no dicha clase, debería contener dicha clase y no las subclases, y si solo contiene algunas de las subclases, podría ser más apropiado que contuviera dicha clase.

Un **atributo** o ***slot* inverso** es uno de tipo `Instance` que define la relación inversa a otro de tipo `Instance`, es decir, existe una relación $R \subseteq A \times B$ tal que esta es $\{(b,a)\}_{(a,b)\in R} \subseteq B \times A$. Es redundante, pero es necesario desde el punto de vista de modelado del conocimiento. Algunos sistemas de adquisición de conocimiento pueden rellenar automáticamente los *slots* inversos para asegurar la consistencia.

7. **Crear instancias.** Se elige una clase, se crea una instancia de ella y se rellenan los valores de los atributos, repitiendo el proceso hasta completar lo suficiente la base de conocimiento.

Debemos elegir un convenio para nombrar los conceptos y adherirnos a él estrictamente. Este dependerá de características del sistema de representación como si soporta más de un espacio de nombres para las clases, atributos e instancias, si diferencia entre mayúsculas o minúsculas o qué delimitadores se permiten.

Protégé-2000 tiene un único espacio de nombres, distingue entre mayúsculas y minúsculas y permite espacios, barras bajas y guiones. Así, cuando el nombre de un concepto tiene varias palabras, podemos separarlas por espacios, escribirlas juntas con mayúscula al principio de cada una, o separarlas con un guion o barra baja. Lo más intuitivo es usar espacios, pero hay que tener cuidado al importar otras ontologías.

Para nombres de conceptos es más intuitivo usar el plural porque las clases representan colecciones de objetos.[1] Ninguna opción es mejor que otra, pero hay que ser consistente con la elección. Algunos sistemas solicitan a los usuarios declarar la opción por adelantado.

Para los atributos, algunas metodologías sugieren usar prefijos y sufijos como «tiene-», «-de» o «-por», según algún convenio, para distinguir atributos y clases a cambio de tener nombres de atributos más largos.

No se recomienda usar cadenas como «clase», «propiedad» o «*slot*» en los nombres de clases y propiedades, ni abreviar nombres de conceptos. En los nombres de subclases directas, o todos tienen el nombre de la superclase o ninguna lo tiene.

[1] Mentira. Es más intuitivo el singular porque «telediarios es-un programas» no tiene el más mínimo sentido y usando el singular por defecto es más fácil razonar sobre la cardinalidad de las relaciones.

Capítulo 3

Sistemas basados en conocimiento

Desde siempre la humanidad ha intentado hacer máquinas para resolver problemas. En la antigua Grecia y el antiguo Egipto encontramos estatuas móviles operadas por monjes; en el siglo XIII Alberto Magno creó una cabeza parlante y un mayordomo; en el XVII, René descartes creó una muñeca autómata; en el XVIII, Jacques de Vaucanson creó músicos de tamaño natural y un pato con aparato digestivo, y Pierre Jaquet-Droz un pianista, un dibujante y un escritor, y en el siglo XIX se crearon máquinas que podían hacer y contestar preguntas.[1]

Los primeros ordenadores los hicieron británicos y estadounidenses.[2] Alan Turing creó un computador de propósito general en base a operadores lógicos, y en 1946 se crearon en EEUU los primeros ordenadores analógicos, con operadores numéricos. También por esa época se empezó a trabajar en psicología cognitiva y la creación de programas para emular el pensamiento humano.

En 1943, McCulloch y Pitts crearon una red de neuronas artificiales que podía realizar cualquier operación[3] y tenía capacidad de aprendizaje. Así surgió la **rama conexionista**, que consideraba clave la creación de redes neuronales para conseguir inteligencia. En 1956 se acuña el término inteligencia artificial hospedado por McCarthy e IBM y al que asistieron McCulloch, Pitts, von Neumann, Minsky, Edmonds y Shannon.[4]

De los 50 a los 70 se dan los primeros éxitos en IA, con programas generales para resolver problemas como GPS de Newell y Simon en 1963, STRIPS de Fikes y Nilsson en 1971, el programa de damas de Samuel[5] y Lisp de McCarthy.

A finales de los 60 se ve que muchos de los métodos generales no son escalables y que es difícil modelar situaciones y sentido común que se suelen expresar en lenguaje natural, por lo que en 1966 Estados Unidos cancela la ayuda a estas investigaciones y en 1970 Reino Unido hace lo propio en base al «informe Lighthill».

Esto lleva a una etapa de pesimismo en torno a los 70, pero también en esta época nacen los sistemas basados en conocimiento (SBC), una de las técnicas más usadas en inteligencia artificial junto con redes neuronales, lógica difusa, algoritmos genéticos y sistemas híbridos.

[1] Por supuesto, no he verificado ninguno de estos datos, y aparecen en unas diapositivas que llaman a Alan Turing «Alan Touring» así que no son muy fiables.

[2] Esto es, si no cuentas el que hicieron los nazis.

[3] Representable en las 40 neuronas o así que tenía.

[4] O algo así, está todo muy esquemático, pero de todas formas no va a preguntar por esto.

[5] Que por el nombre no parece muy general.

Algunos de los primeros SBC son:

1. **DENDRAL** (1969), el primer SBC y proyecto de programación heurística,[6] que busca determinar estructuras moleculares a partir de mediciones de un espectroscopio.

2. **MYCIN** (1972), para diagnóstico y terapia de enfermedades infecciosas, que usa incertidumbre mediante factores de certeza.

3. **PROSPECTOR** (1979), un SBC probabilista para prospección minera que incluye una red semántica, y que encontró un depósito de molibdeno haciendo a un rico 100 millones de dólares más rico.

4. **XCON** (1980), para logística de pedidos de ordenadores VAX, con una precisión de entre el 95 y el 98 %, que supuso un ahorro anual de 25 millones de dólares.

5. **INTERNIST** (1982), para medicina interna.

6. **LES** (1987), para monitorizar y diagnosticar procesos de carga de oxígeno líquido en un transbordador espacial.

Por otro lado, en 1973 surge la lógica difusa, en 1976 la teoría de la evidencia, en 1984 la de la incertidumbre, y sobre esa época surge «Ops», el lenguaje Prolog y «Marcos» de Minsky.[7] Surgen representaciones de conocimiento manipulables por ordenador con esquemas de representación adaptados al tipo de problema, procesos de razonamiento para usar dicho conocimiento y sistemas de aprendizaje y procesamiento del lenguaje natural. La mayor potencia de los ordenadores lleva a la aparición de los primeros entornos de desarrollo.

En los 80 comienza el **periodo industrial**, con empresas que poseen equipos propios de IA. A finales de la década hay una crisis que en los 90 es resuelta por nuevas metodologías de ingeniería del conocimiento, resultando en éxitos como **MARVEL**, para monitorización y análisis de la telemetría del Voyager; **PEGASUS**, una interfaz inteligente en «LN» para consultar información y reservar vuelos; *soft bots* en la web semántica; ERP, o CRM.

Un **sistema inteligente** es uno que usa técnicas que no garantizan soluciones aceptables, sino que esto depende del contexto y del dominio. Los **sistemas basados en conocimiento** (SBC) son un sistema informático que emula la capacidad de razonamiento del ser humano, usando sus mismas fuentes de conocimiento del dominio específico, con estructuras de control separadas del conocimiento del domino para poder usarse en distintos dominios. Un **sistema experto** es un SBC enfocado a resolver problemas reales pequeños pero intelectualmente complejos, como un tutor inteligente o un sistema inteligente CASE.

Un SBC está formado por:

1. Una **memoria de trabajo** o **base de hechos**, que contiene la información para la ejecución de una tarea particular, tanto de hechos establecidos como de metas a alcanzar. Algunas partes son permanentes, como los hechos establecidos y los datos, y otras son temporales del proceso de resolución en curso, actuando como una memoria a corto plazo que también almacena todos los cambios en la información que se producen en el sistema.

[6] Si no se cuentan los proyectos anteriores como el algoritmo A*.

[7] Esto es lo mejor que he podido hacer con una enumeración inconexa de términos.

2. Una **base de conocimiento**, en la que se almacena la mayoría del conocimiento de resolución de problemas. Este suele estar organizado en **reglas**, con un **antecedente** que contiene condiciones de la existencia de elementos en la base de hechos que deben cumplirse para ejecutar la regla y un **consecuente** que puede modificar la base de hechos o ejecutar acciones.

   Las reglas son independientes y no pueden referenciar una a otra, sino que la comunicación entre ellas debe hacerse a través de la base de hechos. Al estar basadas en la experiencia, no reflejan implicaciones lógicas sino convicciones del experto, por lo que muchas veces se usan medidas de certidumbre de la regla en sí.

   Existen formalismos para tratar la incertidumbre y especificar conclusiones complejas, y en algunos el consecuente de las reglas puede activar o desactivar un conjunto de estas.

3. Un **motor de inferencia** encargado de activar reglas y encadenarlas para resolver problemas. Este primero carga los hechos de base en la base de hechos. Entonces, en bucle:

   a) Si se da la condición de terminación, que normalmente corresponde a que se ha verificado un hecho objetivo al que se quería llegar, o si se ejecuta una acción de parada, normalmente por ausencia de otras reglas aplicables, para.

   b) **Equipara** las bases de conocimiento y de hechos para obtener un conjunto de reglas que se pueden aplicar, el **conjunto conflicto** (**CC**).

   c) **Resuelve** el conjunto conflicto, es decir, selecciona la regla a aplicar, de manera informada o desinformada.

   d) Aplica la regla y actualiza la base de hechos.

   El coste computacional de la iteración es la suma del **coste de control**, de la equiparación y resolución, y el **coste de aplicación**, de modo que con una resolución informada predomina el coste de control y con una desinformada predomina el de aplicación. El motor de inferencia debe ser sistemático y eficiente y no causar bucles infinitos, y se puede implementar con dos modos de inferencia:

   a) **Encadenamiento hacia delante.** Se parte de la colección de hechos y se van aplicando reglas para generar nuevos hechos hasta que se llegue al objetivo o no se puedan aplicar más reglas. Aplicar una regla supone actualizar la base de hechos y, en algunos casos, realizar otras acciones.

   b) **Encadenamiento hacia atrás.** Se parte de las hipótesis y se aplican reglas hacia atrás para obtener qué hechos hacen falta para verificarla. Primero se eligen todas las reglas cuyo consecuente sea el objetivo, y si en una de ellas se verifican todas las premisas del antecedente, el objetivo queda verificado, y de lo contrario las premisas no verificadas son nuevos objetivos.

   La elección del mecanismo de inferencia depende del número de estados iniciales y de metas, siendo preferible moverse del conjunto más pequeño al más grande; del factor de ramificación en cada caso, que se puede medir según el tamaño del conjunto conflicto, siendo preferible ir en la dirección con menor factor de ramificación, y de si se debe justificar el razonamiento, en cuyo caso es importante ir en la dirección que mejor se aproxime a la forma de pensar del usuario.

4. Un **módulo de explicación**, para convencer a los usuarios de que el razonamiento es correcto y las soluciones son apropiadas y para permitir a los desarrolladores verificar que el mecanismo de inferencia se adecúa al problema.

   El módulo debe responder de forma comprensible y completa a preguntas sobre el conocimiento y el razonamiento como cómo se ha llegado a la conclusión, cómo se usa la información, qué decisión se ha tomado para un subproblema, por qué no se ha usado o necesitado un tipo de información concreto, por qué no se ha podido llegar a una conclusión, cómo se describe la información necesaria en el proceso de resolución, o qué está haciendo el sistema.

   Incluye un módulo para analizar el estado del proceso del razonamiento y otro para consultar el estado del conocimiento del problema que tiene el sistema y el proceso de resolución el curso, y debe ser fácil de usar.

5. **Interfaces de usuario**, de al menos dos tipos:

   a) Una para usuarios no avanzados, que permita introducir la información y obtener las respuestas y explicaciones. Esta puede ser todo lo compleja que se requiera, incluso con interfaces de usuario inteligentes que se adapten a los conocimientos del usuario.

   b) Una para usuarios avanzados que permita la edición de la base de conocimiento y la depuración. Se divide en:

      1) Una interfaz para el experto adaptada a la representación de conocimiento, que permita la edición controlada de la base de conocimiento incluyendo el vocabulario y la terminología a utilizar y evite la introducción de errores.
      2) Una para el ingeniero del conocimiento, que permita edición avanzada de la base de conocimiento, acceso al resto de módulos del SBC y depuración del sistema en tiempo de ejecución.

Todos los componentes salvo la base de conocimiento existen en una ***shell***, una herramienta específica para la representación de conocimiento.

Los SBC son eficientes, potentes y flexibles, pues una regla solo requiere la información necesaria para su ejecución. Son modulares, separando el conocimiento del proceso de resolución, y naturales, permitiendo a los expertos expresar su conocimiento de resolución de problemas como reglas en la mayoría de situaciones. Sin embargo, puede ser difícil establecer reglas, pues un formalismo del tipo «si-entonces» entraña cierta rigidez a la hora de expresar las condiciones en los antecedentes y premisas en distintas reglas, y también es difícil diseñar reglas concretas, requiriendo haber estudiado el dominio de aplicación con suficiente profundidad. También es difícil usar las reglas, pues estas solo se pueden comunicar a través de la base de hechos y esto es un inconveniente a la hora de ejecutar algoritmos.

Los SBC tienen una serie de diferencias respecto a sistemas convencionales como los de resolución de circuitos eléctricos o cálculo de estructuras:

- Separan el conocimiento de las estructuras de control, incluyen explicaciones y suelen usar *shells*, mientras que los sistemas convencionales solo separan algoritmos de datos, no dan explicaciones y usan gestores de bases de datos convencionales.

- Interpretan datos con métodos declarativos no deterministas, intentando seguir líneas de razonamiento similares a las de los humanos, son muy interactivos y contemplan

abstracción, incertidumbre, aprendizaje, etc., mientras que los sistemas convencionales manipulan datos con algoritmos, centrándose en la solución y no en la forma de obtenerla, y usan bases de datos y procesos predecibles, fiables y exactos.

- Destacan en problemas mal definidos,que no se pueden especificar con precisión y requieren conocimiento heurístico, normalmente en dominios sin experiencia previa computacional, mientras que los sistemas convencionales se usan en problemas bien definidos, especificables sin ambigüedad y resueltos por algoritmos específicos, normalmente en dominios con experiencia computacional.
- El conocimiento usado es tácito, numérico, simbólico y con incertidumbre, procedente de la interacción con expertos, mientras que el de sistemas convencionales es algorítmico y los datos son numéricos, sin incertidumbre y procedentes de la interacción con usuarios.

Capítulo 4

Técnicas de equiparación

El rendimiento del proceso de equiparación en un sistema basado en reglas es afectado por un número elevado de reglas en la base de conocimiento y por la aparición de variables en los antecedentes de las reglas, que en general siempre se da al escribir las reglas con la mayor generalidad posible, convirtiendo la equiparación en un proceso no trivial.

Para mitigar los problemas por el número elevado de reglas usamos **técnicas de indexado**. Una de ellas consiste en dividir el problema en varias etapas y agrupar las reglas según dichas etapas, añadiendo condiciones en el antecedente de las reglas relacionadas con las etapas. Sin embargo, esto reduce la generalidad de las reglas.

4.1. Equiparación con variables

Representamos reglas con una sintaxis basada en el lenguaje CLIPS:

«regla» → **SI** «antecedente» **ENTONCES** «consecuente»
«antecedente» → «condición» | «condición» ∧ «antecedente»
«condición» → «patrón» | ¬«patrón» | (`test` «sexpr»)
«consecuente» → `Añadir`«patrón» | `Borrar`«patrón»
«patrón» → («constante» «parámetros»)
«parámetros» →| «constante» «parámetros» | «variable» «parámetros»
«constante» → [^?)][^)]∗
«variable» → \?[^)]∗

Una **condición negada** es un patrón inmediatamente precedido por ¬. Todas las variables en el consecuente de una regla deben aparecer en algún patrón no negado en el antecedente.

Los elementos o hechos de la BH son patrones sin variables. Usamos la **hipótesis del mundo cerrado:** todo hecho que no aparece en la BH se considera falso.

Una **instanciación** es un par formado por una regla de la BC y una lista de hechos de la BH de forma que:

1. La lista tiene tantos elementos como patrones no negados en el antecedente.

2. Existe una asignación de las variables que aparecen en la regla a constantes tal que, al sustituir en la regla:

   a) Los hechos de la lista son iguales a los correspondientes patrones no negados del antecedente de la regla.

   b) Los patrones negados no aparecen en la BH.

   c) Los «sexpr» en la regla evalúan a un valor verdadero en un entorno léxico resultante de añadir al entorno global esta asignación de variables en el espacio de nombres de valores.[1]

El conjunto conflicto es el conjunto de instanciaciones posibles.

4.2. Algoritmo RETE

RETE (**REdundancia TEmporal**) es un algoritmo propuesto por Forgy en 1979 y 1982 para calcular el conjunto conflicto de forma incremental, teniendo en cuenta que cada ejecución de una regla hace pocos cambios a la BH y por tanto al conjunto conflicto.

Un **testigo** es un par formado por una etiqueta + o − y una lista de hechos. Cada vez que se añade o borra un hecho de la BH se crea un **testigo** con etiqueta + o − respectivamente y el hecho, y se envía al nodo raíz de la **red RETE**, un grafo dirigido cuyos nodos reciben testigos o listas de testigos, los procesan y pueden propagarlos a sus sucesores. Tipos de nodos en la red:

1. **Raíz:** Hay uno. Recibe testigos de fuera y los propaga.

2. **De componente:** Contiene un índice (de la lista de los componentes de los hechos en el testigo) y una constante. Recibe un testigo y lo propaga sólo si tiene esa constante en ese índice.

3. **Terminal:** Contiene una regla. Cuando recibe un testigo positivo, crea una instanciación con dicha regla y los hechos, y cuando recibe uno negativo la borra.

4. **De comprobación de variables:** Contiene dos índices. Recibe un testigo de un solo hecho y lo propaga sólo si los valores en esos índices del hecho son iguales.

5. **De unión de elementos condición de una misma regla:** Le llegan testigos de un puerto izquierdo y uno derecho, y cada puerto tiene asociada una memoria de testigos positivos. Cuando recibe un testigo positivo, lo añade a la memoria de ese puerto, y cuando recibe uno negativo lo borra de esa memoria. Entonces, para cada testigo en la memoria del otro puerto, propaga un testigo con la etiqueta del testigo recibido cuyos hechos resultan de concatenar los del testigo recibido y el de la memoria, primero los de la izquierda y luego los de la derecha.

[1] La terminología es del estándar de Common Lisp (`http://www.lispworks.com/documentation/HyperSpec/Body/03_a.htm`), pero básicamente esto significa que cada «sexpr» es una condición en la que aparecen las variables y que deben cumplir los valores asignados a estas variables.

6. **De unión de elementos condición de una misma regla que contiene varias variables:** Como el anterior pero además contiene una lista de pares de índice del testigo izquierdo e índice del derecho, y sólo propaga los testigos concatenados si en cada par de índices de la lista los valores son iguales.

7. **De unión de elementos de condición negados con los restantes elementos de condición de una misma regla:** También tiene dos puertos, una memoria de testigos positivos en cada uno y una lista de pares de índices, pero los testigos en la memoria izquierda tienen un contador asociado. Cuando llega un testigo positivo por la izquierda, se guarda con un contador igual al número de testigos por la derecha para los que se da la igualdad en los pares de índices, y se propaga si el contador es 0. Cuando llega uno negativo, si había uno positivo, se borra, y si su contador era 0, se propaga el testigo negativo. Cuando llega un testigo positivo por la derecha, se guarda y se aumenta en 1 el contador de los testigos de la izquierda para los que se da la igualdad en los pares de índices, y para los que dejan de estar a 0 se propaga un testigo igual pero negativo. Cuando llega uno negativo por la derecha, si había uno igual positivo, se borra y, para cada testigo de la izquierda para el que se da la igualdad, el contador se reduce en 1, propagando el testigo si llega a 0.

Podemos suponer que todos los hechos con igual primer elemento tienen la misma longitud, pues de lo contrario basta renombrarlo. Podemos suponer que toda regla tiene un patrón no negado en el antecedente, pues de lo contrario basta crear un hecho `(T)` que siempre está en la base de hechos y añadirlo al antecedente de la regla.

Para compilar un conjunto de reglas en una red:

1. Se crea el nodo raíz.

2. Para cada patrón del antecedente de cada regla:

   a) Se crea un nodo componente por cada componente constante del patrón.

   b) Para cada variable que aparece $n \geq 2$ veces en el patrón, se crean $n - 1$ nodos de comprobación de variables para asegurar que las apariciones son iguales.

   c) Se crea un eje del nodo raíz al primer nodo creado en este paso y uno de cada nodo al siguiente.

3. Para cada regla con $n \geq 2$ patrones:

   a) Elegimos un patrón no negado del antecedente y llamamos L al último nodo de su secuencia.

   b) Para cada otro patrón del antecedente:

      1) Llamamos R al último nodo del patrón.
      2) Creamos un nodo de unión de elementos de condición, o de elementos de condición negados si el patrón de R está negado.
      3) Conectamos L al puerto izquierdo y R al derecho.
      4) Para cada variable que aparezca tanto en algún patrón de L como en el de R, añadimos un par de índices a la lista del nodo creado para asegurar que sean iguales.

4. Mientras haya dos nodos iguales con el mismo predecesor, estos nodos se unen en uno solo.

Esta versión no considera *tests* en el antecedente, pero estos se representan trivialmente como variantes de nodos de componente, comprobación de variables o unión que en vez de comprobar igualdades, o además, comprueba la condición.

Muchos de estos pasos no especifican exactamente el orden de los elementos, por lo que estos se pueden reordenar para poner las comprobaciones de *tests* y otras con más posibilidades de fallar lo más arriba posible y maximizar el número de uniones de nodos, aumentando la eficiencia.

4.3. Resolución de conflictos

Consiste en elegir qué instanciación se ejecuta del conjunto conflicto. Algunas estrategias:

1. **Selección arbitraria.** Se elige aleatoriamente. «Sólo es aplicable cuando todas las reglas tienen la misma posibilidad de ser efectivas.»

2. **Selección de la primera regla.** Se elige la primera instanciación en un cierto buen orden del conjunto conflicto, o una de la primera regla en un cierto buen orden de la base de reglas.[2] Hay problemas en los que establecer un buen orden de la base de reglas no es deseable.

3. **Selección por prioridad.** Cada regla tiene asignada una prioridad y se elige una instanciación con máxima prioridad de su regla. Algunos sistemas permiten establecer la prioridad según el orden de escritura de la base de conocimiento.

4. **Regla más específica.** Una regla es más o igual de específica que otra si tiene todas las condiciones de la otra. Se ignora la instanciación más genérica.

5. **Elemento añadido más recientemente.** Cada elemento de la BH tiene asociada una marca de tiempo con el momento de su creación o última modificación (regeneración del hecho por parte de una regla), medida bien en número de ciclos (actualizaciones del CC), o en número de acciones ejecutadas, en cuyo caso no todos los elementos añadidos por la misma instanciación de una regla tienen la misma prioridad. Se elige la instanciación que contiene el hecho más reciente en la CC.

6. **Seleccionar una nueva regla.** Se elige una instanciación no ejecutada previamente, evitando que una misma regla se ejecute indefinidamente (**principio de refracción**). Una instanciación que desaparece del conjunto conflicto y luego vuelve a aparecer no se considera ejecutada.

7. **Seleccionar todas las reglas.** Se aplican todas las reglas a la vez en el mismo ciclo. Es especialmente útil en dominios médicos para examinar todas las posibilidades, pero hay que tener cuidado de no incurrir en una explosión combinatoria o recorrer caminos idénticos.

[2] Las diapositivas dicen una cosa y luego la otra, y la segunda sería una forma de selección por prioridad.

Cada una de estas estrategias salvo la última se puede ver como una función que recibe un conjunto de instanciaciones no vacío y devuelve un subconjunto de este. Dadas dos estrategias f y g de este tipo, llamamos $f \bullet g$ a la estrategia dada por $A \mapsto f(A) \cap g(A)$, y $f \to g$ a la estrategia dada por

$$A \mapsto \begin{cases} f(A), & |f(A)| \leq 1, \\ g(f(a)), & \text{en otro caso.} \end{cases}$$

Ambos operadores son asociativos, y el segundo no necesita que la estrategia g sea de este tipo pero si no lo es la estrategia $f \to g$ en general tampoco lo es.

Capítulo 5

MYCIN

Un **agente antimicrobiano** es un fármaco para eliminar microbios o detener su crecimiento. Elegir uno sería fácil si hubiera un único agente no tóxico efectivo para cada tipo de bacteria. MYCIN es un sistema basado en reglas escrito en Lisp que usa información clínica para aconsejar sobre el tratamiento de una infección. Fue creado por un equipo de programación heurística del Stanford Research Institute de la Universidad de Stanford, formado entre otros por:

- Edward Feigenbaum, que defendió el uso de sistemas de producciones para codificar conocimiento específico de un dominio en base a trabajos de Allan Newell que defendían de estos sistemas como un formalismo elegante y eficiente para el modelado psicológico.
- Bruce Buchanan y Edward Shortliffe, que participaron previamente en DENDRAL y el sistema de alerta de interacción de medicamentos MEDIPHOR.[1]
- Stanley Cohen, jefe del departamento de farmacología clínica de Stanford que también trabajó en MEDIPHOR.
- Stanton Axline y Thomas Merigan, del departamento de enfermedades infecciosas.

Tiene 5 módulos:

1. **De consulta.** Núcleo del sistema, que interactúa con los médicos para recoger información del paciente y generar recomendaciones.
2. **De explicación.** Genera explicaciones y justificaciones de las recomendaciones de conocimiento.
3. **De adquisición de conocimiento.** Usado por los expertos para actualizar la base de conocimiento.

[1] Las diapositivas dicen que estos se incorporaron posteriormente; Wikipedia que era la tesis doctoral de Shortliffe dirigida por Buchanan. Las diapositivas no dicen que estos estuvieran en DENDRAL y MEDIPHOR pero lo intuyen, Wikipedia dice que Buchanan estuvo en DENDRAL y MEDIPHOR no aparece en Internet. En casos como este se da prioridad a lo estudiado en clase porque es lo que entra en el examen, y no porque necesariamente sea verdad.

4. **Base de conocimiento.** Almacena las reglas.

5. **Base de datos de pacientes.** Va almacenado los datos relativos al paciente que está siendo analizado.

MYCIN determina el tratamiento en 4 fases:

1. Decidir si la infección es significativa.

2. Determinar los organismos implicados.

3. Seleccionar fármacos apropiados.

4. Elegir el fármaco o combinación de fármacos más apropiado para el paciente.

5.1. Base de datos

Un **parámetro clínico** es una característica con un valor como el nombre del paciente, el lugar del cultivo, la morfología del organismo, la dosis del fármaco, etc. Tiene las siguientes propiedades:[2]

`Expect` Tipo de valor esperado: `yn`, `numb`, `one-of` o `any`. Los parámetros pueden ser multivaluados, de un solo valor o booleanos.

`Prompt, prompt1` Pregunta que hay que hacer para solicitar un valor.

`Labdata` Indica si el dato procede de laboratorio.

`Trans` Información para traducir lo expresado por el parámetro al inglés.

`Default` Unidad en que se expresan los valores numéricos.

`Condition` Expresión a ejecutar antes de preguntar el valor del parámetro, que devuelve verdadero si no hay que preguntar el valor.

`Lookahead` Reglas que referencian al parámetro en su premisa.

`Updated-by` Reglas que lo actualizan.

`Contained-in` Reglas que lo contienen en el consecuente.

Las únicas propiedades obligatorias son `expect` y `trans`. MYCIN tiene 65 parámetros clínicos en 6 clases: `prop-cul`, `prop-drg`, `prop-op`, `prop-org`, `prop-pt` y `prop-ther`.

Los valores de un parámetro clínico se representan con una lista `(value cf)` formada por el valor del parámetro y un **factor de certeza**, un número del -1 al 1 que indica un grado de certeza subjetiva de que el parámetro tenga ese valor, donde 1 significa que se está totalmente seguro, -1 que se está totalmente seguro de que es falso y 0 que no se tiene evidencia a favor ni en contra o las evidencias se contrarrestan perfectamente.

El módulo de traducción considera que algo es definitivo si $|\text{CF}| = 1$, que hay una fuerte evidencia si $0{,}8 \leq |\text{CF}| < 1$, que hay evidencia si $0{,}4 \leq |\text{CF}| < 0{,}8$ y que hay una débil evidencia si $|\text{CF}| < 0{,}4$.

[2] Aprovecho que Lisp es *case insensitive* para aumentar la legibilidad.

Si un parámetro solo puede tomar un valor, la suma de los CFs de los distintos parámetros no puede ser mayor que 1 (sí puede ser menor que -1), y si una hipótesis tiene CF $= 1$ el resto se pueden suponer con CF $= -1$. Si un parámetro es booleano su único valor almacenado es `yes` y el CF de «no» será el opuesto al de «yes».

Las inferencias se hacen dentro de un **contexto**, formado por un tipo de contexto, valores de los parámetros clínicos, y un posible contexto padre, formando los contextos un **árbol de contexto**.

Un tipo de contexto se define con `(defcontext` *`contextname parms initialdata goals`*`)` y lo forman:

1. Un nombre (un símbolo).
2. Una lista de parámetros clínicos aplicables.
3. Una lista de parámetros cuyos parámetros se han de obtener al principio.
4. Una lista de objetivos (parámetros).

Tipos de contexto predefinidos:

`Person` Raíz del árbol, con los datos del paciente.

`Priorculs` Hijo de `person`, cultivo en la historia clínica.

`Priororgs` Hijo de `priorcul`, organismo identificado en la historia clínica.

`Priordrogs` Hijo de `priororgs`, fármaco administrado en la historia clínica.

`Curcul` Hijo de `person`, cultivo en la sesión actual.

`Curorg` Hijo de `curcul`, cultivo en la sesión actual.

`Curdrogs` Hijo de `curorg` o `priororgs`, fármaco en la sesión actual.

`Opers` Hijo de `person`, procedimientos terapéuticos seguidos por el paciente.

`Opdrogs` Hijo de `opers`, fármacos tomados por el paciente.

`Possther` Tratamiento candidato a ser recomendado.

Todos salvo `person` pueden tener más de una instancia.

5.2. Base de conocimiento

La última versión de MYCIN de 1978 tenía unas 500 reglas, definidas con `(defrules (`*`rule-name premise action`*`)*)` y formadas por:

1. Un **nombre**, un símbolo no evaluado.
2. Una **premisa**, una expresión que devuelve un factor de certeza o `NIL` (equivalente a 0), como puede ser:
   a) `($and` *`expr`*`*)`. Conjunción de premisas; toma el mínimo de sus CF si es mayor que 0,2.

Función	Valor según el CF	Función	Valor según el CF
`same`	$(\text{CF} > ,2 \to \text{CF}; \texttt{T} \to \texttt{NIL})$	`defis`	$\text{CF} = 1$
`thoughtnot`	$(\text{CF} < -,2 \to -\text{CF}; \texttt{T} \to \texttt{NIL})$	`defnot`	$\text{CF} = -1$
`notsame`	$\text{CF} \leq ,2$	`notdefis`	$\text{CF} \in (,2, 1)$
`mightbe`	$\text{CF} \geq -,2$	`notdefnot`	$\text{CF} \in (-1, ,2)$
`vnotknown`	$\lvert\text{CF}\rvert \leq ,2$		

Cuadro 5.1: Funciones de evaluación de factores de certeza. Reciben un contexto (normalmente el actual en la variable global `cntxt`), un parámetro (símbolo no evaluado) y una serie de valores (símbolos no evaluados, normalmente uno) y actúa según el máximo de los CFs de que el parámetro tenga cada valor en el contexto, que se toma 0 si el conjunto de valores es vacío.

Función	Valor según CF	Función	Valor según CF
`known`	$\text{CF} > ,2$	`definite`	$\text{CF} = 1$
`notknown`	$\text{CF} \leq ,2$	`notdefinite`	$\text{CF} < 1$

Cuadro 5.2: Funciones de evaluación de parámetros. Como las del Cuadro 5.1 pero tomando el máximo CF entre todos los posibles valores del parámetro.

b) `(`***fn cntxt parm value*****`*)`**, donde ***fn*** es una de las funciones del Cuadro 5.1.

c) `(`***fn cntxt parm*****`)`**, donde ***fn*** es una de las funciones del Cuadro 5.2.

3. Una **acción**, una expresión de la forma:

`(conclude` ***cntxt parm value tally rulecf*****`)`** Establece el factor de certeza de que ***parm*** (símbolo no evaluado) valga ***value*** (no evaluado sólo si es un símbolo) en el contexto ***cntxt*** a $f \coloneqq \textit{tally} \cdot \frac{\textit{rulecf}}{1000}$. Si este parámetro ya tenía asignado un factor de certeza e para ese valor, este se actualiza a

$$\begin{cases} e + f(1-e), & e, f \geq 0; \\ e + f(1+e), & e, f < 0; \\ \frac{e+f}{1-\min\{|e|,|f|\}}, & \text{en otro caso,} \end{cases}$$

lanzando un error si $\{e, f\} = \{1, -1\}$ al ser esto una contradicción.

`(do-all` ***expr*****`*)`** Ejecuta todas las expresiones.

Una disyunción en la premisa se puede representar con varias reglas con la misma acción. Para aplicar una regla, se evalúa la premisa, se guarda el factor de certeza devuelto en la variable global `tally` y, si este es mayor que 0,2, se evalúa la conclusión.

Las reglas se organizan en grupos según el tipo de contexto en que se pueden aplicar (ver Cuadro 5.3).

La regla `rule092` define el objetivo global del sistema: si existe un organismo que requiere tratamiento y existen indicios de la existencia de otros organismos que requieren tratamiento, aunque no hayan sido detectados en los cultivos en curso, entonces recopilar los posibles tratamientos que puedan ser efectivos contra los organismos considerados y determinar cuál es la mejor terapia de la lista.

Tipo de regla	Tipos de contexto	Tipo de regla	Tipos de contexto
`Culrules`	`Curcul`, `priorculs`	`Drgrules`	`Curdrogs`, `priordrogs`
`Curculrules`	`Curcul`	`Oprules`	`Opers`
`Curorgrules`	`Curorg`	`Patrules`	`Person`
`Pdrgrules`	`Priordrogs`	`Orderrules`	Recomendaciones terapéuticas
`Prculrules`	`Priorculs`	`Therrules`	Selección de fármacos
`Prorgrules`	`Priororgs`		

Cuadro 5.3: Tipos de reglas predefinidos y tipos de contexto en que se aplican.

Para recopilar los tratamientos, se usan los CF de las hipótesis para seleccionar las identificaciones más probables y, para cada organismo identificado, se dispara una regla de tipo `therules`.[3] Una regla de este tipo puede ser «si el organismo es pseudomonas entonces el tratamiento es colistina (,98), polimyxcina (,96), gentamicina (,96), carbenicilina (,65) o sulfisoxazolona (,64)», donde los números indican la posibilidad de que el organismo aislado en el hospitar de Stanford sea sensible al fármaco, pero se pueden modificar si el sistema se instala en otro hospital, y puede cambiarlos el propio MYCIN si tiene datos reales relativos a dicho organismo en el paciente o si puede inferirlos de la información sobre los cultivos realizados.

Las reglas `therules` no son disparadas directamente por el mecanismo de inferencia ya que no se encuentran en ninguna lista `updated-by` de los parámetros clínicos. El resultado final es una o más listas de posibles fármacos junto a su sensibilidad inferida por MYCIN.

Para seleccionar la mejor terapia, se tienen en cuenta la sensibilidad del organismo al fármaco, si se está administrando un fármaco de similar sensibilidad y la cobertura y las contra-indicaciones de los fármacos.

5.3. Módulo de consulta

Usa las bases de datos y de conocimiento; un **diccionario** para el procesamiento del lenguaje natural para entender las preguntas del usuario; **listas** para referenciar variables sin duplicar su contenido, y **tablas de conocimiento** que indican qué valores deben tomar ciertos parámetros clínicos bajo ciertas circunstancias.

El proceso de consulta tiene dos pasos:

1. Crear el contexto del paciente como nodo raíz del árbol de contexto.
2. Aplicar las reglas que definen los objetivos principal de dicho contexto.

El motor de inferencia usa encaminamiento hacia atrás mediante dos procesos: `monitor`, que analiza las premisas de una regla, y `findout`, que deriva el valor de un parámetro de las reglas o preguntando al usuario. Concretamente:

`Monitor` Evalúa las premisas. Para cada una, si no tiene toda la información para evaluar el parámetro, llama a `findout` y termina si la premisa resulta no ser cierta (si CF $\leq 0{,}2$). Finalmente, si se cumple la premisa (no devuelve `nil`), evalúa las conclusiones.

[3] ¿Es esto lo mismo que `therrules`? Las diapositivas son horribles así que no lo sé.

`Findout` Si `labdata` es cierto, primero pregunta el valor al usuario y, si este no lo conoce, ejecuta `monitor` para cada regla relevante (en `updated-by`). En otro caso primero ejecuta `monitor` para cada regla relevante y, si esto no consigue el valor (ninguna de las reglas es aplicable) pregunta al usuario. Los valores se guardan en el contexto en la memoria dinámica para no volver a preguntarla o calcularla, en lo que llamamos la **agenda**.

El árbol de contexto se puede extender **explícitamente** cuando una regla hace referencia a contextos que no han sido creados, en cuyo caso se ejecuta `findout` y si se devuelven uno o más valores se crea el nodo, o **implícitamente** cuando no hay referencia explícita a un contexto necesario para evaluar una condición o cuando al evaluar una regla no existen los contextos apropiados.

Se pueden producir bucles por reglas auto-referenciadas, en que un parámetro aparece tanto en la premisa como en la conclusión para aumentar el grado de certeza de la conclusión, y por ciclos en la cadena de razonamiento. Para evitarlo se mantiene una lista de los parámetros que están siendo evaluados por `findout` y se consideran desconocidos los parámetros en la lista, evitando considerar las reglas que lo tienen en la premisa.

5.4. Módulo de explicación

Está formado por:

- **RSC** (***Reasoning Status Checker***): Permite al usuario preguntar, cada vez que se le lanza una pregunta:
  - Por qué se ha hecho la pregunta (`why`), para lo que el sistema recorre el árbol de contexto en sentido ascendente para determinar qué reglas y objetivos de más alto nivel se está intentando seguir.
  - Cómo se ha llegado a cierta conclusión (`how`), para lo que se recorre el árbol en sentido descendente para determinar qué reglas y subobjetivos se han satisfecho.
- **GQA** (***General Question Answerer***): Permite consultas en lenguaje natural sobre **conocimiento estático**, información almacenada de parámetros clínicos, hechos médicamente ciertos, y **conocimiento dinámico**, conclusiones e información usada en el proceso de consulta, derivada de las reglas y no completamente cierta.

Para entender una consulta, se reduce a un conjunto de palabras terminales, se identifica el tipo de consulta según una serie de patrones preestablecidos para decidir si responde el RSC o el GCA. Entonces se determina qué parámetros, valores y pesos son relevantes en la consulta a partir de la información en el diccionario, usando el peso para descartar parámetros no relevantes, se determinan las reglas que pueden responder a la consulta, se eligen las que cumplen las restricciones para los valores de los parámetros y se muestran estas al usuario.

5.5. Proceso de evaluación

Se seleccionaron 10 casos clínicos con las condiciones de que no debía haber más de 3 casos de meningitis viral y debería haber como mínimo uno de tuberculosis, uno micótico, uno vírico, uno bacteriano grampositivo y otro bacteriano gramnegativo. Estos se presentaron a MYCIN, 5

médicos docentes, un becario pos-doctoral, un médico residente, un alumno, que recomendaron un tratamiento para cada caso, y las 90 recomendaciones junto a los 10 tratamientos que se prescribieron en realidad se presentaron a cada uno de 8 evaluadores que hicieron sus propias recomendaciones y clasificaron cada una de las 100 como **equivalente** si coincide o equivale a la del evaluador, **alternativa aceptable** si es distinta pero aceptable o **no aceptable** si es inapropiada o inaceptable.

Se analizaron las 800 valoraciones con ANOVA para ver si había diferencias estadísticamente significativas entre MYCIN y los otros 9 prescriptores, se usó el test de Tukey para determinar esas diferencias y se hizo lo mismo con los evaluadores.

El 65 % de los tratamientos de MYCIN fueron catalogados como aceptables frente a un 55,5 % entre los 5 médicos docentes, aunque había mucha variabilidad entre ellos con un ratio entre 42,5 % y 62,5 %. Se define[4] el consenso entre evaluadores como que al menos 5 de ellos acepten el mismo tratamiento, y MYCIN dio 7 tratamientos con consenso y 0 fallos frente a una media de 4,4 y 0,8 entre los médicos docentes. Esto muestra un rendimiento de MYCIN ligeramente superior al de los miembros del departamento de enfermedades infecciosas del hospital de la universidad de Stanford, pero el estudio es limitado por el reducido número de casos frente a la gran cantidad de posibles infecciones.

MYCIN no fue usado en la práctica porque los usuarios no se sentían cómodos con la interfaz[5]. Además, la mayoría de hospitales no tenían mucha potencia de cálculo y la base de conocimiento sólo cubre una pequeña parte del dominio de enfermedades infecciosas.

5.6. Clasificación heurística

William J. Clancey analizó una serie de sistemas expertos para caracterizar el «nivel de conocimiento»[6] y concluyó que casi todos usan lo que llamó **clasificación heurística**, un proceso de clasificación para relacionar conceptos de dos jerarquías distintas por procedimientos aproximados para problemas en que una gran cantidad de atributos para cada categoría imposibilita una comparación directa. Fases:

1. **Abstracción de datos.** Convertir datos observados en datos abstractos. Tipos de abstracción:

   Definicional Se definen clases de objetos en base a sus propiedades esenciales.

   Cualitativa Se crean categorías según el valor de medidas cuantitativas.

   Por generalización Jerarquía «es-un», recorrida en sentido ascendente.

2. **Equiparación heurística.** Los datos abstractos disparan hipótesis abstractas (categorías generales). La relación entre datos e hipótesis no es uno a uno y puede haber excepciones a las reglas generales.

3. **Refinamiento.** Una vez el espacio de soluciones está acotado, se evalúan las subcategorías y se refinan las verosímiles mediante **clasificación jerárquica**.

[4] Incorrectamente.
[5] La cual era horrible.
[6] Aparentemente tenemos que estudiar este término pero no lo que significa.

En el caso de un diagnóstico, se introduce una jerarquía de hipótesis diagnósticas y una de datos, se infieren datos y se definen asociaciones no jerárquicas entre datos y categorías. Se usa una taxonomía de categorías diagnósticas para guiar y focalizar el razonamiento sobre hipótesis diagnósticas.

Primero se considera como hipótesis una categoría diagnóstica general, se evalúa la hipótesis y, si es verosímil, se refina en hipótesis más específicas, repitiendo con cada hipótesis específica y parando cuando ninguna hipótesis se puede refinar más. El estado inicial es un conjunto de observaciones iniciales y una hipótesis inicial muy abstracta; el estado final es un conjunto de hipótesis concretas más plausibles, y hay dos subtareas: **evaluar** una hipótesis que no se ha evaluado anteriormente para obtener su verosimilitud y **refinar** una hipótesis verosímil que no se ha refinado anteriormente para obtener otras más concretas.

5.7. Resultados

MYCIN probó que los SBCs pueden abordar eficientemente problemas complejos en dominios específicos y sentó las bases de este tipo de sistemas: conocimiento separado de la resolución; posibilidad de justificar las conclusiones; enfoque en los procesos de evaluación, distintos a los de sistemas convencionales; declaración y organización explícita de los elementos de conocimiento usado para describir las reglas, anticipando la importancia de las ontologías, e importancia del proceso de adquisición de conocimiento para el éxito del SBC.

A partir de MYCIN surgieron muchos proyectos que profundizaron en distintos aspectos básicos de estos sistemas: **EMYCIN**, un entorno de desarrollo creado a partir del motor de MYCIN y en que se basaron muchos SBC; **TIERESIAS**, un entorno de adquisición de conocimiento que hacía de intérprete entre los expertos y MYCIN; **GUIDON**, una extensión de MYCIN para uso formativo; **ONCOCIN**, un sistema que asignaba protocolos de tratamiento a enfermos de cáncer y los monitorizaba, y **NEOMYCIN**, sistema resultante de la reorganización de la base de conocimiento de MYCIN para que GUIDON la usara de forma más efectiva.

Capítulo 6

Conjuntos borrosos

Un **conjunto borroso** F sobre un **universo de discurso** U es una función $\mu_F : U \to [0,1]$ llamada **función de pertenencia** de F, que representa el grado de pertenencia de cada elemento de U en F. Escribimos

$$F =: \sum_{x \in U} \frac{\mu_F(x)}{x} =: \int_{x \in U} \frac{\mu_F(x)}{x},$$

donde la fracción y los símbolos sumatorio e integral son solo símbolos y no se pueden simplificar. Se suele usar la primera notación cuando U es **discreto** (todos sus puntos son aislados) y la segunda cuando es **continuo** (no tiene puntos aislados). Si $U = \{x_1, \ldots, x_n\}$, escribimos

$$F =: \left\{ \frac{\mu_F(x_1)}{x_1} + \cdots + \frac{\mu_F(x_n)}{x_n} \right\}.$$

Llamamos **soporte** de F a $\mathrm{Supp}_F \coloneqq \{x \in U \mid F(x) > 0\}$, **núcleo** de F a $\ker F \coloneqq \{x \in U \mid \mu_F(x) = \sup_{x \in U} F(x)\}$, **altura** de F a $\mathrm{Height}_F \coloneqq \sup_{x \in U} F(x)$ y, para $\alpha \in [0,1)$, **α-corte** de F a $F_\alpha = \{x \in U \mid F(x) > \alpha\}$. F es **vacío**, $F = \emptyset$, si $\mathrm{Supp}_F = \emptyset$; **unitario** o ***singleton*** si $|\mathrm{Supp}_F| = 1$, y **normalizado** si $\mathrm{Height}_F = 1$. Si U es un subconjunto de un espacio vectorial E y se considera F definido sobre E con $F(x) = 0$ para $x \in E \setminus U$, F es **convexo** si $\forall x, y \in E, \forall \alpha \in [0,1], A(\alpha x + (1-\alpha)y) \geq \text{mín}\{A(x), A(y)\}$.

Funciones de pertenencia comunes sobre el universo $\mathbb{R}$:

1. **Funciones trapezoidales:** Para $a < b \leq c < d$,

$$\mathrm{trapmf}(x; a, b, c, d) \coloneqq \begin{cases} \frac{x-a}{b-a}, & x \in [a, b]; \\ 1, & x \in [b, c]; \\ \frac{d-x}{d-c}, & x \in [c, d]; \\ 0, & \text{en otro caso.} \end{cases}$$

2. **Funciones triangulares:** Para $a < b < c$, $\mathrm{trimmf}(x; a, b, c) \coloneqq \mathrm{trapmf}(x; a, b, b, c)$.

3. **Funciones S:** Para $a < b$,

$$\mathrm{smf}(x;a,b) \coloneqq \begin{cases} 0, & x \le a; \\ 2\left(\frac{x-a}{b-a}\right)^2, & x \in \left[a, \frac{a+b}{2}\right]; \\ 1-2\left(\frac{b-x}{b-a}\right)^2, & x \in \left[\frac{a+b}{2}, b\right], \\ 1, & x \ge b. \end{cases}$$

4. **Funciones Π:** Para $a < b \le c < d$,

$$\mathrm{pimf}(x;a,b,c,d) \coloneqq \begin{cases} 2\left(\frac{x-a}{b-a}\right)^2, & x \in \left[a, \frac{a+b}{2}\right]; \\ 1-2\left(\frac{b-x}{b-a}\right)^2, & x \in \left[\frac{a+b}{2}, b\right]; \\ 1, & x \in [b,c]; \\ 1-2\left(\frac{x-c}{d-c}\right)^2, & x \in \left[c, \frac{c+d}{2}\right]; \\ 2\left(\frac{d-x}{d-c}\right)^2, & x \in \left[\frac{c+d}{2}, d\right]; \\ 0, & \text{en otro caso.} \end{cases}$$

Dados dos conjuntos borrosos A y B sobre U, $A \subseteq B$ si $\forall x \in U, A(x) \le B(x)$, y esto es un orden parcial.

6.1. T-normas

Un operador $\# : D \subseteq (\mathbb{R} \times \mathbb{R}) \to \mathbb{R}$ es **monótono** si para $(a,b),(a',b') \in D$ con $a \le a'$ y $b \le b'$ es $a\#b \le a'\#b'$. Una operador $\# : [0,1] \times [0,1] \to [0,1]$ asociativo, conmutativo y monótono es una **t-norma** si $\forall a \in [0,1], (a\#0 = 0 \wedge a\#1 = a)$, y es una **s-norma** o **t-conorma** si $\forall a \in [0,1], (a\#0 = a \wedge a\#1 = 1)$. Llamamos $*$ a una t-norma cualquiera y $\oplus$ a una s-norma cualquiera.

Algunas t-normas:

1. **Mínimo:** $a \wedge b \coloneqq \min\{a,b\}$.

2. **Producto algebraico:** $a * b \coloneqq ab$.

3. **Producto acotado:** Para $p \in [-1,\infty)$, $a * b \coloneqq \max\{0, (1+p)(a+b-1) - pab\}$. Se suele tomar $p = 0$.[1]

4. **Producto drástico:**

$$x * y \coloneqq \begin{cases} x, & y = 1; \\ y, & x = 1; \\ 0, & \text{en otro caso.} \end{cases}$$

[1] Las diapositivas dicen $p \in \mathbb{R}$, pero la monotonía y otras propiedades sólo se cumplen cuando $p \ge -1$.

5. **Familia Dubois-Prade:** Para $p \in [0,1]$, $x * y = \frac{xy}{\max\{x,y,p\}}$ (tomando límites cuando $x, y = 0$ para $p = 0$).

6. **Familia Yager:** Para $p \in \mathbb{R}^+$, $x * y \coloneqq 1 - \min\{1, \sqrt[p]{(1-x)^p + (1-y)^p}\}$.

Algunas s-normas:

1. **Máximo:** $x \vee y \coloneqq \max\{x, y\}$.

2. **Suma algebraica:** $x \oplus y \coloneqq x + y - xy$.

3. **Suma acotada:** Para $p \geq -1$, $x \oplus y \coloneqq \min(1, x + y + pxy)$. Se suele usar $p = 0$.

4. **Suma drástica:**
$$x \oplus y = \begin{cases} x, & y = 0; \\ y, & x = 0; \\ 1, & \text{en otro caso.} \end{cases}$$

5. **Familia Dubois-Prade:** Para $p \in [0,1]$, $x \oplus y \coloneqq 1 - \frac{(1-x)(1-y)}{\max\{1-x,1-y,p\}}$.

6. **Familia Yager:** Para $p \in \mathbb{R}^+$, $x \oplus y \coloneqq \min\{1, \sqrt[p]{x^p + y^p}\}$.

Un **complemento** o **negación** es una función $N : [0,1] \to [0,1]$ monótona decreciente con $N(0) = 1$ y $N(1) = 0$, y es **fuerte** si es estrictamente decreciente e involutivo ($N^2 = 1_{[0,1]}$). También se puede requerir que sea continuo. Algunos complementos:

1. **Usual:** $N(x) \coloneqq 1 - x$.

2. **Familia Sugeno:** Para $\lambda \in (-1, \infty)$, $N(x) \coloneqq \frac{1-x}{1+\lambda x}$.

3. **Familia Yager:** Para $w \in \mathbb{R}^+$, $N(x) \coloneqq \sqrt[w]{1-x}$.

Una t-norma $*$ y una s-norma $\oplus$ son **duales** o **conjugadas** respecto a un complemento N si $N(x \oplus y) = N(x) * N(y)$. Toda t-norma tiene una única s-norma dual bajo el complemento usual. La de la t-norma del mínimo es la s-norma del máximo, y la de la t-norma drástica es la s-norma drástica. Normalmente se usa la negación usual, la t-norma del mínimo y la s-norma del máximo.

6.2. Unión, intersección y complemento

Dados un complemento N, una t-norma $*$ y una s-norma $\oplus$ duales y conjuntos borrosos A y B sobre un universo U,

$$A \cup B \coloneqq \int_{x \in U} \frac{A(x) \oplus B(x)}{x}, \qquad A \cap B \coloneqq \int_{x \in U} \frac{A(x) * B(x)}{x}, \qquad \overline{A} \coloneqq \int_{x \in U} \frac{N(A(x))}{x}.$$

Propiedades:

1. La unión e intersección son conmutativas y asociativas.

2. $A \cup \emptyset = A$, $A \cap \emptyset = \emptyset$.

3. $A \cup U = A \cap U = A$, donde $U \coloneqq \int_{x \in U} \frac{1}{x}$.

Las siguientes se cumplen para las normas usuales pero no en general:

1. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

2. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

3. $A \cup A = A \cap A = A$.

Las siguientes no se cumplen tampoco para las normas del máximo y el mínimo:

1. **Ley de la contradicción:** $A \cup \overline{A} = U$.

2. **Ley del medio excluido:** $A \cap \overline{A} = \emptyset$.

6.3. Relaciones

Dados conjuntos borrosos $A_1, \ldots, A_n$ sobre universos $U_1, \ldots, U_n$, su **producto cartesiano borroso** es

$$A_1 \times \cdots \times A_n \coloneqq \int_{x \in U_1 \times \cdots \times U_n} \frac{A_1(x_1) * \cdots * A_n(x_n)}{x},$$

y una **relación borrosa** es un conjunto borroso sobre $U_1 \times \cdots \times U_n$. El **dominio** y el **rango** de una relación borrosa binaria R sobre $A \times B$ son

$$\mathrm{Dom}R \coloneqq \int_{x \in A} \frac{\text{máx}_{y \in Y}\, R(x,y)}{x}, \qquad \mathrm{Ran}R \coloneqq \int_{x \in B} \frac{\text{máx}_{x \in X}\, R(x,y)}{x}.$$

La **composición borrosa** o **sup-star** de dos relaciones borrosas $R : A \times B \to [0,1]$ y $S : B \times C \to [0,1]$ es

$$R \circ S \coloneqq \int_{(x,z) \in A \times C} \frac{\sup_{y \in B}(R(x,y) * S(y,z))}{(x,z)}.$$

Nótese que el orden es al contrario que en la composición convencional. Esta composición se llama **max-min** si la t-norma es el mínimo y **max-producto** si es el producto algebraico.

Propiedades:[2]

1. Asociativa.

2. $S \subseteq T \implies R \circ S \subseteq R \circ T$.

[2]Las diapositivas incluyen también $R \circ (S \cup T) = (R \cup S) \circ (R \cup T)$ y $R \circ (S \cap T) \subseteq (R \cap S) \circ (R \cap T)$, pero estas no son correctas ni siquiera cuando tienen sentido, ni siquiera con las normas y negación usuales.

Capítulo 7

Lógica borrosa

Una **variable lingüística** es una tupla $V = (N, U, T, G, M)$ donde N es el **nombre** de la variable, U es el **dominio** de valores de la variable, T es un alfabeto cuyos símbolos se llaman **etiquetas lingüísticas**, G es una gramática con alfabeto T y $M : L(G) \to (U \to [0,1])$ es una **regla semántica**. Llamamos **término lingüístico** de V a un elemento de $L(G)$.

Estas variables permiten describir términos del lenguaje natural en términos matemáticos precisos. En general G es de la forma

$$S \to P \mid H\ S \mid (S \text{ and } S) \mid (S \text{ or } S) \mid (\text{not } S),$$

donde P es un conjunto finito de **términos primarios** o **etiquetas lingüísticas** (bajo, alto, medio, etc.), H uno de **modificadores lingüísticos** o ***hedges***, y llamamos **conectores lógicos** a and, or y not. Un **término atómico** es un término primario, un *hedge* o un conector lógico. Cada término primario p lleva asociado un $C_p : U \to [0,1]$ y cada modificador h una transformación $T_h : (U \to [0,1]) \to (U \to [0,1])$, y para $p \in P$, $h \in H$ y $r, s \in L(G)$,

$$M(p) \coloneqq C_p, \qquad M(h\,r) \coloneqq T_h(M(r)), \qquad M(\text{not } r) \coloneqq N(M(r)),$$
$$M(r \text{ and } s) \coloneqq M(r) * M(s), \qquad M(r \text{ or } s) \coloneqq M(r) \oplus M(s),$$

donde N es la negación, $*$ la t-norma y $\oplus$ la s-norma.

Tipos de *hedges*:

- **De concentración:** $T_h(S)(x) \coloneqq S(x)^p$ para cierto $p > 1$.

$$T_{\text{muy}}(S)(x) \coloneqq S(x)^2, \qquad T_{\text{más}}(S)(x) \coloneqq S(x)^{\frac{5}{4}}.$$

- **De dilatación:** $T_h(S)(x) \coloneqq S(x)^p$ para cierto $p \in [0,1]$.

$$T_{\text{más o menos}}(S)(x) \coloneqq \sqrt{S(x)}, \qquad T_{\text{poco}}(S)(x) \coloneqq S(x)^{\frac{3}{4}}.$$

- **De intensificación:** Para cada x, $S(x)$ está entre $T_h(S)(x)$ y $\frac{1}{2}$.

$$T_{\text{especialmente}}(S)(x) \coloneqq T_{\text{bastante}}(S)(x) \coloneqq \begin{cases} 2S(x)^2, & x \in [0, \frac{1}{2}]; \\ 1 - 2(1 - S(x))^2, & x \in [\frac{1}{2}, 1]. \end{cases}$$

- **De difuminación:** Para cada x, $T_h(S)(x)$ está entre $S(x)$ y $\frac{1}{2}$.

$$T_{\text{cerca de}}(S)(x) \coloneqq T_{\text{casi}}(S)(x) \coloneqq \begin{cases} \sqrt{\frac{1}{2}S(x)}, & x \in [0, \frac{1}{2}]; \\ 1 - \sqrt{\frac{1}{2}(1 - S(x))}, & x \in [\frac{1}{2}, 1]. \end{cases}$$

Criterios para definir variables lingüísticas:

- **Distinguibilidad:** El significado de los términos es más claro cuanto más se puedan distinguir las funciones de pertenencia, por lo que en general no debería haber $p, q, r \in P$ distintos con $\text{Supp}_{C_p} \cap \text{Supp}_{C_q} \cap \text{Supp}_{C_r} \neq \emptyset$.
- **Normalidad:** Todos los C_p deberían ser normalizados.
- Número de etiquetas lingüísticas moderado, entre 3 y 7, pudiéndose llegar hasta 9.
- **Cubrimiento:** $\bigcup_p \text{Supp}_{C_p} = U$, y para $x \in U$, $\sum_p C_p(x)$ debería ser 1 o cercano a 1.

Una **proposición borrosa atómica** es una expresión de la forma «x es P» o «x es P es C», donde x es una variable libre, P un conjunto borroso que representa la propiedad y C un **cualificador de certeza**, una función $[0,1] \to [0,1]$ de entre las siguientes:

$$\begin{aligned} \text{cierto}(x) &\coloneqq x, & \text{falso}(x) &\coloneqq 1 - x, \\ \text{muy cierto}(x) &\coloneqq x^2, & \text{muy falso}(x) &\coloneqq (1-x)^2, \\ \text{bastante cierto}(x) &\coloneqq \sqrt{x}, & \text{bastante falso}(x) &\coloneqq \sqrt{1-x}. \end{aligned}$$

Una **proposición borrosa compuesta** es una expresión de la forma «p and q», «p or q» o «not p», donde p y q son proposiciones borrosas (simples o compuestas).

La **función de pertenencia** de una proposición borrosa es

$$\mu_{x \text{ es } P} \coloneqq P, \qquad \mu_{x \text{ es } P \text{ es } C} \coloneqq C \circ P, \qquad \mu_{\text{not } p} \coloneqq N \circ P,$$
$$\mu_{p \text{ and } q}(x,y) \coloneqq p(x) * q(y), \qquad \mu_{p \text{ or q}}(x,y) \coloneqq p(x) \oplus q(y),$$

tomando $*$ y $\oplus$ conjugadas respecto a N. Las proposiciones borrosas son sentencias sobre un concepto sin definición precisa, permitiendo expresar ideas subjetivas con distintas interpretaciones. Lo que en lógica clásica son paradojas, en lógica difusa es una **media verdad** o **media falsedad** (grado de pertenencia $\frac{1}{2}$).

El **razonamiento aproximado** permite razonar sobre proposiciones imprecisas. Una **regla IF-THEN borrosa** es una expresión «IF u es A THEN v es B». Si A se define sobre el universo U y B sobre V, esta regla se entiende como una relación entre U y V, la **implicación borrosa** $R_{A \to B} : U \times V \to [0,1]$ dada por $R_{A \to B}(u,v) \coloneqq I(A(u), B(v))$, donde $I : [0,1] \times [0,1] \to [0,1]$ es una **función de implicación**. Algunas funciones de implicación:

1. **Interpretación clásica:** $I(x,y) \coloneqq N(x) \oplus y$.
2. **Dienes-Rescher:** $I_D(x,y) \coloneqq \text{máx}\{1-x, y\}$ (interpretación clásica con las normas usuales).
3. **Lukasiewicz:** $I_{Lu}(x,y) \coloneqq \text{mín}\{1, 1-x+y\}$.

4. **Zadeh:** $I_Z(x,y) \coloneqq \text{máx}\{\text{mín}\{x,y\}, 1-x\}$.

5. **Gödel:** $I_G(x,y) \coloneqq \begin{cases} 1, & x \leq y; \\ y, & \text{en otro caso.} \end{cases}$

Se tiene $I_Z \leq I_D \leq I_{Lu}$. Definiendo $I(x,y) \coloneqq x * y$ obtenemos:

1. **Mandami:** $I_M(x,y) \coloneqq \text{mín}\{x,y\}$.

2. **Larsen:** $I_L(x,y) \coloneqq xy$.

Estas funciones se basan en una interpretación «local» de la implicación, en la que $P \to Q$ equivale a $P \wedge Q$ si se cumple P y se ignora en otro caso. Si hace falta una interpretación global como en la lógica clásica, hay que usar una como las de la primera lista.

Reglas lógicas:

- **Modus ponens generalizado:** Dados los conjuntos borrosos A y A' sobre U y B sobre V, si «x es A'» e «IF x es A THEN y es B», entonces «y es $A' \circ R_{A \to B}$».

- **Modus tollens generalizado:** Dados los conjuntos borrosos A sobre U y B y B' sobre V, si «y es B'» e «IF x es A THEN y es B», entonces «x es $R_{A \to B} \circ B'$».

Si la función de implicación es la t-norma, al aplicar modus ponens sobre «x es A'» e «IF x es A THEN y es B», se obtiene «y es B'» con

$$B'(y) = \sup_{x \in U}(A'(x) * I(A(x), B(y))) = \sup_{x \in U}(A'(x) * A(x)) * B(y),$$

y llamamos ***degree of fulfillment*** a

$$\text{DOF}(A', A) \coloneqq \sup_{x \in U}(A'(x) * A(x)) = \sup_{x \in U}(A' \cap A)(x),$$

con lo que $B'(y) = \text{DOF}(A', A) * B(y)$. Por ello las implicaciones de Mandami y Larsen son muy eficientes y son las más usadas. Además hay una interpretación gráfica intuitiva: para calcular la implicación de Mandami, se toma el supremo del mínimo de A y A', que será el DOF, y se corta B por este punto (tomando el mínimo) para obtener B', y para la de Larsen se toma el supremo del producto de A y A' y se «achata» B multiplicando por este número.

Capítulo 8

Sistemas de inferencia borrosa

Una **base de reglas borrosas** es una familia finita de reglas IF-THEN de la forma

$$R_k : \text{IF } x_1 \text{ es } A_{k1} \text{ and } \dots \text{ and } x_n \text{ es } A_{kn} \text{ THEN } y \text{ es } B_k,$$

donde cada A_{ki} es un conjunto borroso sobre un universo U_i y B_k sobre un universo V, y llamamos $U \coloneqq U_1 \times \cdots \times U_n$. La base de reglas es **completa** si $\forall x \in U_1 \times \cdots \times U_n, \exists k : \forall j, A_{kj}(x_j) > 0$, **consistente** si no existen reglas con los mismos antecedentes pero distintos consecuentes, y **continua** si no existen dos reglas adyacentes[1] cuyos consecuentes tienen intersección vacía, asegurando un comportamiento suave.

Una **regla parcial** es una de la forma

$$\text{IF } x_{p_1} \text{ es } A_{k1} \text{ and } \dots \text{ and } x_{p_m} \text{ es } A_{km} \text{ THEN } y \text{ es } B_k,$$

con $1 \leq p_1 < \cdots < p_m \leq n$, $m < n$ y cada A_{ki} definido sobre U_{p_i}, y equivale a una regla completa que en cada $j \neq p_1, \dots p_m$ incluye en el antecedente «x_j es U_j», donde $U_j \coloneqq \int_{x \in U_j} \frac{1}{x}$. Una **regla OR** es una de la forma

$$\begin{aligned} \text{IF } (x_{t_{11}} \text{ es } A_{11} \text{ and } \dots \text{ and } x_{t_{1s_1}} \text{ es } A_{1s_1}) \text{ or } \dots \text{ or} \\ (x_{t_{m1}} \text{ es } A_{m1} \text{ and } \dots \text{ and } x_{t_{ms_m}} \text{ es } A_{ms_m}) \text{ THEN } B, \end{aligned}$$

y equivale a m reglas

$$\text{IF } x_{t_{i1}} \text{ es } A_{i1} \text{ and } \dots \text{ and } x_{t_{is_i}} \text{ es } A_{is_i} \text{ THEN } B.$$

Una **función de agregación** es una que toma una familia finita de conjuntos borrosos sobre V y devuelve otro. Las más comunes son la unión y la intersección, normalmente con las normas del máximo y el mínimo.

Un **motor de inferencia borrosa** es un sistema que toma como entrada conjuntos borrosos A'_i sobre los U_i, aplica modus ponens entre los elementos y cada una de las reglas de una base de reglas y aplica una función de agregación a los resultados. Aunque es preferible que la base de reglas sea consistente, puede haber reglas inconsistentes y los resultados se agregan, lo que no es posible en un sistema de inferencia clásico.

[1] No sé qué significa adyacentes aquí.

Para $D \subseteq \mathbb{R}$, un **fuzzificador** es una función $f : D \to (D \to [0,1])$ tal que $\forall x \in D, f(x)(x) = \max_{y \in D} f(x)(y)$, y un **defuzzificador** es una función $g : (D \to [0,1]) \to D$.

Un **sistema de inferencia borroso** (**SIB**) está formado por una base de reglas borrosas $(U = U_1 \times \cdots \times U_n) \to V$ con cada $U_i, V \subseteq \mathbb{R}$, fuzzificadores sobre cada U_i y un defuzzificador sobre V, y tiene asociada una función $U \to V$ que asigna a cada $x \in U$ el resultado de pasar cada x_i por su fuzzificador, aplicar el motor de inferencia a los resultados y pasar el conjunto borroso devuelto por el defuzzificador.

Un fuzzificador debe ayudar a eliminar el ruido de las variables de entrada, si existe, y a simplificar los cálculos implicados en el motor de inferencia. Algunos fuzzificadores:

- **Unitario:** $f(x)(y) = \delta_{xy}$.
- **Triangular:** Para un $b > 0$, $f(x)(y) = \max\left\{0, 1 - \frac{|y-x|}{b}\right\}$.
- **Gaussiano:** Para un $b > 0$, $f(x)(y) = \mathrm{e}^{-\left(\frac{x-y}{b}\right)^2}$.

El fuzzificador unitario simplifica mucho los cálculos ya que $\mathrm{DOF}(A, f(x)) = A(x)$. Los fuzzificadores triangulares y gaussianos sólo simplifican los cálculos si las funciones de pertenencia en los antecedentes son triangulares o gaussianas respectivamente, pero permiten eliminar ruido en la entrada.

Un defuzzificador debería ser continuo; **creíble** o **intuitivamente plausible**, con una salida que represente intuitivamente el conjunto de entrada, y con poca complejidad computacional, especialmente en controladores borrosos ya que operan en tiempo real. Algunos defuzzificadores:

- **Centro de gravedad** o **centroide:**

$$g(B) = \frac{\int_V yB(y)\,\mathrm{d}y}{\int_V B(y)\,\mathrm{d}y},$$

  cambiando las integrales por sumatorios si el denominador es 0.

- **Media ponderada de los centros:** En vez de agregar y luego defuzzificar, se defuzzifica cada resultado B_k por centro de gravedad obteniendo centros y_k y el resultado es

$$\frac{\sum_k y_k B_k(y_k)}{\sum_k B_k(y_k)}.$$

  Es el más usado ya que es continuo y creíble y el cálculo es sencillo cuando las funciones son simétricas, aunque también se puede aplicar a funciones no simétricas.

- **Máximo:** Si $\mathrm{hgt}(B) \coloneqq \{y \in V \mid B(y) = \sup_{y \in V} B(y)\} \neq \emptyset$, se puede tomar:
  - **Máximo más pequeño:** $\inf \mathrm{hgt}(B)$.
  - **Máximo más grande:** $\sup \mathrm{hgt}(B)$.
  - **Media de los máximos:**

$$\frac{\int_{\mathrm{hgt}B} y\,\mathrm{d}y}{\int_{\mathrm{hgt}B}\mathrm{d}y},$$

    cambiando las integrales por sumatorios si el denominador es 0.

Es creíble y computacionalmente simple, pero no continuo, y si B es no convexo, $B(f(B))$ puede ser pequeño.

Como **teorema**, dado un SIB con reglas

$$\text{IF } x_1 \text{ es } A_{k1} \text{ and } \ldots \text{ and } x_n \text{ es } A_{kn} \text{ THEN } y \text{ es } B_k,$$

fuzzificador unitario, t-norma producto, implicación de Larsen y defuzzificador media de los centros, si cada B_k es normalizado con centro y_k, la función asociada al SIB es

$$f(x) \coloneqq \frac{\sum_k y_k \prod_i A_{ki}(x_i)}{\sum_k \prod_i A_{ki}(x_i)}.$$

Teorema universal de aproximación: Para $U \subseteq \mathbb{R}^n$ compacto, $g : U \to \mathbb{R}$ continua y $\varepsilon > 0$, existe un SIB del tipo del teorema anterior con los A_{ki} y B_k de la forma

$$A_{ki}(x) = a_{ki} \mathrm{e}^{-\left(\frac{x-\overline{x}_{ki}}{\sigma_{ki}}\right)^2}, \qquad B_k(y) = \mathrm{e}^{-(y-\overline{y})^2},$$

para la que la función asociada $f : U \to \mathbb{R}$ cumple $\|f - g\|_\infty < \varepsilon$, y se dice entonces que un SIB de este tipo es un **aproximador universal**. Otros aproximadores universales son las redes neuronales y los controladores convencionales.