Intel.ligència Artificial

Grau en Informàtica

APUNTS DE REPRESENTACIÓ DEL CONEIXEMENT

Departament de Llenguatges i Sistemes Informàtics

UPC FIB Facultat d'Informàtica de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

CURS 2011/2012 1Q

Índice general

0. Introducción

Este documento contiene las notas de clase de la asignatura Intel·ligència Artificial de la ingeniería en informática de la Facultat d'Informàtica de Barcelona.

El documento cubre todos los temas impartidos en la asignatura salvo el tema de introducción y esta pensado como complemento a las transparencias utilizadas en las clases. El objetivo es que se lea esta documentación antes de la clase correspondiente para que se obtenga un mejor aprovechamiento de las clases y se puedan realizar preguntas y dinamizar la clase.

En ningún caso se pueden tomar estos apuntes como un substituto de las clases de teoría.

Parte I

Representación del conocimiento

1. Introducción a la representación del conocimiento

1.1 Introducción

Acabamos de ver un conjunto de algoritmos que son capaces de resolver problemas mediante la exploración del espacio de soluciones. Estos algoritmos se basan en una información mínima para su resolución. No solo en lo que respecta a las características que representan al problema, sino al conocimiento involucrado en las tomas de decisión durante la exploración.

Las funciones heurísticas que se utilizan en la exploración tienen en cuenta información global que evalúa el problema en cada paso de la misma manera. Evidentemente, durante la resolución de un problema no tiene por que verse su estado siempre de la misma manera, ni tiene por que tener la misma importancia toda la información de la que disponemos.

Si observamos como nosotros mismos resolvemos problemas, podemos fijarnos en que la información que vamos teniendo en cuenta a medida que lo resolvemos va cambiando. No usamos, por ejemplo, la misma información cuando iniciamos la resolución de un problema que cuando estamos en medio de la resolución, ya que nuestra incertidumbre sobre donde está la solución va cambiando, y parte de la información puede pasar a ser crucial o irrelevante. Esta capacidad de tomar mejores decisiones puede hacer que un problema se pueda resolver de manera más eficiente.

Estas decisiones no las podemos tomar si no tenemos un conocimiento profundo de las características del problema y de como nos pueden ayudar a tomar decisiones. Esto nos lleva a pensar que en todo problema complejo en IA se debe plantear qué cocimiento nos puede ser necesario para resolverlo y como deberemos utilizarlo para hacer más eficiente la resolución.

El conocimiento que podemos usar puede ser tanto general como dependiente del dominio, pero teniendo en cuenta que el conocimiento específico del problema será el que con más probabilidad nos permitirá hacer eficiente la resolución.

Esta necesidad de introducir más conocimiento en la resolución de un problema nos lleva a plantearnos dos cuestiones. Primero, el cómo escoger el formalismo de representación que nos permita hacer una traducción fácil del mundo real a la representación. El conocimiento necesario es por lo general bastante complejo, hacer la traslación de los elementos del dominio a una representación es una tarea costosa. Segundo, el cómo ha de ser esa representación para que pueda ser utilizada de forma eficiente. Este conocimiento tendrá que utilizarse en el proceso de toma de decisiones, deberemos evaluar cuando es necesario usarlo y qué podemos obtener a partir de él. Sin un mecanismo eficiente para su uso no conseguiremos ninguna ganancia.

En este punto debemos distinguir entre *información* y *conocimiento*. Denominaremos **información** al conjunto de datos básicos, sin interpretar, que se obtienen como entrada del sistema. Por ejemplo los datos numéricos que aparecen en una analítica de sangre o los datos de los sensores de una planta química. Estos datos no nos dicen nada si no los asociamos a su significado en el problema.

Denominaremos **conocimiento** al conjunto de datos de primer orden, que modelan de forma estructurada la experiencia que se tiene sobre un cierto dominio o que surgen de interpretar los datos básicos. Por ejemplo, la interpretación de los valores de la analítica de sangre o de los sensores de la planta química para decir si son normales, altos o bajos, preocupantes, peligrosos, ..., el conjunto de estructuras de datos y métodos para diagnosticar a pacientes a partir de la interpretación del análisis de sangre, o para ayudar en la toma de decisiones de que hacer en la planta química

Los sistemas de IA necesitan diferentes tipos de conocimiento que no suelen estar disponibles en bases de datos y otras fuentes de información:

- Conocimiento sobre los objetos en un entorno y las posibles relaciones entre ellos
- Conocimiento sobre los procesos en los que interviene o que le son útiles
- Conocimiento difícil de representar como datos básicos, como la intensionalidad, la causalidad, los objetivos, información temporal, conocimiento que para los humanos es "de sentido común", etc.

Podríamos decir para un programa de inteligencia artificial el conocimiento es la combinación entre la información y la forma en la que se debe interpretar[1].

1.2 Esquema de representación

Para poder representar algo necesitamos saber entre otras cosas sus características o su estructura, que uso le dan los seres inteligentes, como adquirirlo y traspasarlo a un sistema computacional, qué estructuras de datos son adecuadas para almacenarlo y qué algoritmos y métodos permiten manipularlo.

Una posibilidad sería fijarnos en como los seres humanos representamos y manipulamos el conocimiento que poseemos. Por desgracia no hay respuestas completas para todas estas preguntas desde el punto de vista biológico o neurofisiológico, no tenemos una idea clara de como nuestro cerebro es capaz de adquirir, manipular y usar el conocimiento que utilizamos para manejarnos en nuestro entorno y resolver los problemas que este nos plantea.

Afortunadamente podemos buscar respuestas en otras áreas, principalmente la lógica. A partir de ella construiremos modelos que simulen la adquisición, estructuración y manipulación del conocimiento y que nos permitan crear sistemas artificiales inteligentes.

Como sistema para representar el conocimiento hablaremos de **esquema de representación** que para nosotros será un instrumento para codificar la realidad en un ordenador. Desde un punto de vista informático un esquema de representación puede ser descrito como una combinación de:

- Estructuras de datos que codifican el problema en curso con el que se enfrenta el agente → **Parte estática**
- Estructuras de datos que almacenan conocimiento referente al entorno en el que se desarrolla el problema y procedimientos que manipulan las estructuras de forma consistente con una interpretación plausible de las mismas → **Parte dinámica**

La parte estática estará formada por:

- Estructura de datos que codifica el problema
- Operaciones que permiten crear, modificar y destruir elementos en la estructura
- Predicados que dan un mecanismo para consultar esta estructura de datos
- Semántica de la estructura, se definirá una función que establezca una relación entre los elementos de la realidad y los elementos de la representación. Esta función es la que nos permitirá interpretar lo que hagamos en el representación en términos del mundo real.

La parte dinámica estará formada por:

[1] Se suele dar la ecuación *Conocimiento = Información + Interpretación* como visión intuitiva de este concepto parafraseando el libro de N. Wirth *Algorithms + Data Structures = Programs*

- Estructuras de datos que almacenan conocimiento referente al entorno/dominio en el que se desarrolla el problema
- Procedimientos que permiten
  - Interpretar los datos del problema (de la parte estática) a partir del conocimiento del dominio (de la parte dinámica)
  - Controlar el uso de los datos: estrategias de control, métodos que nos permiten decidir cuando usamos el conocimiento y como éste guía los procesos de decisión.
  - Adquirir nuevo conocimiento: mecanismos que nos permiten introducir nuevo conocimiento en la representación, ya sea por observación del mundo real o como deducción a partir de la manipulación del conocimiento del problema.

Es importante tener en mente que la realidad no es el esquema de representación. De hecho, podemos representar la realidad utilizando diferentes esquemas de representación. La representación es simplemente una abstracción que nos permite resolver los problemas del dominio en un entorno computacional.

Se ha de tener siempre en cuenta que nuestra representación siempre es incompleta, debido a:

- Modificaciones: el mundo es cambiante, pero nuestras representaciones son de un instante
- Volumen: mucho (demasiado) conocimiento a representar, siempre tendremos una representación parcial de la realidad
- Complejidad: La realidad tiene una gran riqueza en detalles y es imposible representarlos todos

El problema de la codificación del mundo esta ligado a los procedimientos de adquisición y mantenimiento de la representación. Deberíamos poder introducir en la representación toda aquella información que es consecuencia del cambio de la realidad, esto requiere poder representar todo lo que observamos en la realidad y obtener todas las consecuencias lógicas de ese cambio[2]. La eficiencia de los métodos de adquisición es clave, el problema es que no existe ningún mecanismo lo suficientemente eficiente como para que sea plausible desde el punto de vista computacional mantener una representación completa de la realidad.

Los problemas de volumen y complejidad de la realidad están relacionados con la granularidad de la representación. Cuanto mayor sea el nivel de detalle con el que queramos representar la realidad, mayor será el volumen de información que tendremos que representar y manipular. Esto hace que el coste computacional de manejar la representación crezca de manera exponencial, haciendo poco plausible llegar a ciertos niveles de detalle y obligando a que la representación sea una simplificación de la realidad.

1.3 Propiedades de un esquema de representación

Un buen formalismo de representación de un dominio particular debe poseer las siguientes propiedades:

- *Ligadas a la representación*
  - **Adecuación Representacional:** Habilidad para representar todas las clases de conocimiento que son necesarias en el dominio. Se puede necesitar representar diferentes tipos de conocimiento, cada tipo necesita que el esquema de representación sea capaz de describirlo, por ejemplo conocimiento general, negaciones, relaciones estructurales, conocimiento

[2] A este problema se le ha denominado el problema del *marco de referencia* (*frame problem*).

causal, ... Cada esquema de representación tendrá un lenguaje que permitirá expresar algunos de estos tipos de conocimiento, pero probablemente no todos.

- **Adecuación Inferencial:** Habilidad para manipular estructuras de representación de tal manera que devengan o generen nuevas estructuras que correspondan a nuevos conocimientos inferidos de los anteriores. El esquema de representación debe definir los algoritmos que permitan inferir nuevo conocimiento. Estos algoritmos pueden ser específicos del esquema de representación o puede ser genéricos.

- *Ligadas al uso de la representación*

  - **Eficiencia Inferencial:** Capacidad del sistema para incorporar conocimiento adicional a la estructura de representación, llamado metaconocimiento, que puede emplearse para focalizar la atención de los mecanismos de inferencia con el fin de optimizar los cómputos. El esquema de representación puede utilizar mecanismos específicos que aprovechen el conocimiento para reducir el espacio de búsqueda del problema.
  - **Eficiencia en la Adquisición:** Capacidad de incorporar fácilmente nueva información. Idealmente el sistema por sí mismo deberá ser capaz de controlar la adquisición de nueva información y su posterior representación.

Desafortunadamente no existe un esquema de representación que sea óptimo en todas estas características a la vez. Esto llevará a la necesidad de escoger la representación en función de la característica que necesitemos en el dominio de aplicación específico, o a utilizar diferentes esquemas de representación a la vez.

1.4 Tipos de conocimiento

El conocimiento que podemos representar en un dominio de aplicación lo podemos dividir en dos tipos. Por un lado esta el *conocimiento declarativo* que es un conocimiento que se representa de manera independiente a su uso, es decir, no impone un mecanismo específico de razonamiento, por lo tanto podemos decidir como usarlo dependiendo del problema que vayamos a resolver con él. Los mecanismos de razonamiento empleados en este tipo de conocimiento son de tipo general (por ejemplo resolución lineal) y la eficiencia en su uso dependerá de la incorporación de conocimiento de control que permita dirigir su utilización.

Tipos de conocimiento declarativo son el *conocimiento relacional*, que nos indica como diferentes conceptos se relacionan entre si y las propiedades que se pueden obtener a través de esas relaciones, *conocimiento heredable*, que establece las relaciones estructurales entre conceptos de manera que podamos saber propiedades de generalización/especialización, inclusión o herencia de propiedades y el *conocimiento inferible*, que establece como se pueden derivar propiedades y hechos de otros mediante relaciones de deducción.

El segundo tipo de conocimiento es el *conocimiento procedimental*. Este indica explícitamente la manera de usarlo, por lo que el mecanismo de razonamiento está ligado a la representación, con la ventaja de que es más eficiente de utilizar.

1.4.1 Conocimiento Relacional simple

La forma más simple de representar hechos declarativos es mediante un conjunto de relaciones expresables mediante tablas (como en una base de datos). La definición de cada tabla establece la

descripción de un concepto y la tabla contiene todas sus instanciaciones. Por ejemplo, podemos tener la colección de información sobre los clientes de una empresa

Cliente	Dirección	Vol Compras	...
A. Perez	Av. Diagonal	5643832	
J. Lopez	c/ Industria	430955	
.J. García	c/ Villaroel	12734	
..			

El principal problema es que tal cual no aporta conocimiento, solo es una colección de datos que necesita de una interpretación y procedimientos que permitan utilizarlo. Podríamos obtener nuevo conocimiento a partir de esta información con procedimientos que calcularan por ejemplo la media de compras en una población o el mejor cliente o la tipología de clientes.

Las bases de datos pueden proporcionar información en un dominio, pero es necesaria una definición explícita de las propiedades que se definen, las relaciones que se pueden establecer entre las diferentes tablas y las propiedades de esas relaciones.

1.4.2 Conocimiento Heredable

De entre el conocimiento que podemos expresar en un dominio suele ser muy habitual hacer una estructuración jerárquica del conocimiento (taxonomía jerárquica), de manera que se puedan establecer relaciones entre los diferentes conceptos. En este caso estamos representando mediante un árbol o grafo las relaciones entre conceptos que se basan en el principio de generalización y/o especialización (clase/subclase, clase/instancia, todo/parte, unión/intersección). Este conocimiento pretende representar definiciones de conceptos aprovechando las relaciones estructurales entre ellos.

Como se puede ver en la figura 1.1 los nodos son los conceptos/clases, y los arcos las relaciones jerárquicas. El mecanismo de inferencia se basa en la herencia de propiedades y valores (herencia simple/múltiple, valores por defecto) y en la subsumción de clases (determinar si una clase está incluida en otra).

1.4.3 Conocimiento Inferible

Es conocimiento descrito mediante el lenguaje de la lógica. En este caso las expresiones describen hechos que son ciertos en el dominio. La riqueza del lenguaje de la lógica nos permite representar multitud de hechos, estableciendo por ejemplo propiedades universales sobre los predicados del dominio, la existencia de elementos que cumplan unas propiedades o establecer relaciones de inferencia entre predicados. Podemos decir por ejemplo:

$$\forall x, y : persona(x) \wedge \neg menor(x) \wedge \neg ocupacion(x, y) \rightarrow parado(x)$$

El mecanismo de deducción en el caso de la lógica de primer orden se obtiene eligiendo de entre los métodos generales de deducción automática que existen, como por ejemplo la resolución lineal.

1.4.4 Conocimiento Procedimental

Este conocimiento incluye la especificación de los procesos para su uso. En este tipo se incluyen los procedimientos que permiten obtener conocimiento a partir de información o nuevo conocimiento que ya se tiene. Por ejemplo, si se tiene la información Fecha_nacimiento= DD-MM-AAAA, se puede calcular la edad como una función que combine esta información con la fecha actual.

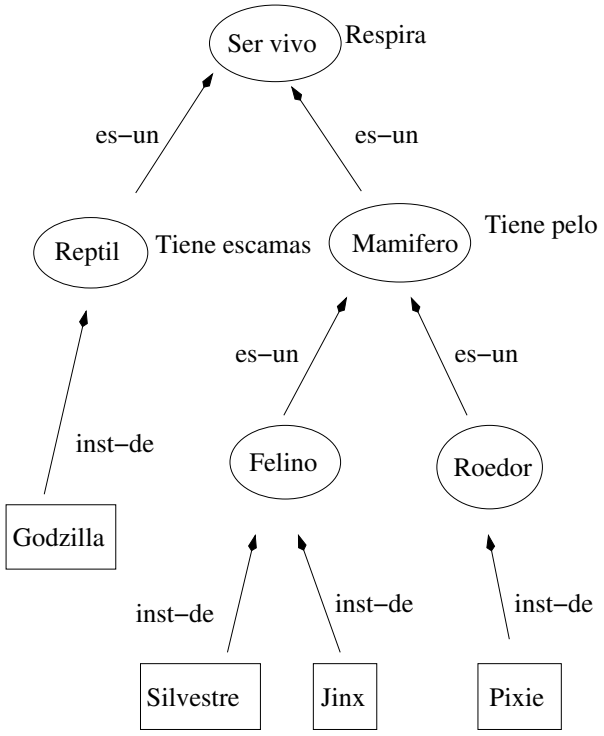


Figura 1.1: Jerarquia de conceptos

También se incluyen en este conocimiento las denominadas *reglas de producción*. Estas son expresiones condicionales que indican las condiciones en las que se deben realizar ciertas acciones al estilo `SI condición ENTONCES acción`. La combinación de estas expresiones condicionales pueden permitir resolver problemas descomponiéndolos en una cadena de acciones guiada por las condiciones de las reglas.

Este tipo de conocimiento suele ser más eficiente computacionalmente, pero hace más difícil la inferencia y la adquisición/modificación ya que se apoyan en mecanismos específicos.

2.1 Introducción

Una gran parte de los formalismos de representación del conocimiento que se utilizan en inteligencia artificial se fundamentan directamente en la lógica. De hecho la lógica es el lenguaje utilizado por todas las disciplinas científicas para describir su conocimiento de manera que pueda ser compartido sin ambigüedad y poder utilizar métodos formales para probar sus afirmaciones y obtener sus consecuencias

Es por tanto interesante ver la lógica como lenguaje de representación. La lógica provee un lenguaje formal a través de cual podemos expresar sentencias que modelan la realidad. Es un lenguaje declarativo que establece una serie de elementos sintácticos a partir de los cuales podemos representar conceptos, propiedades, relaciones, constantes y la forma de conectar todos ellos. A partir de este lenguaje podemos relacionar la semántica de las sentencias que expresamos con la semántica de la realidad que queremos representar.

Como la lógica es un lenguaje declarativo, se establecen una serie de procedimientos que permiten ejecutar la representación y obtener nuevo conocimiento a partir de ella.

Nos centraremos en la lógica clásica y en particular en la lógica proposicional y la lógica de predicados. Los conceptos que se explican en este capítulo los conocéis de la asignatura *Introducción a la lógica*, así que podéis consultar sus apuntes para tener una descripción más detallada. Este capítulo solo servirá para refrescar la memoria y no como descripción exhaustiva del formalismo de la lógica.

2.2 Lógica proposicional

Es el lenguaje lógico más simple y nos permite definir relaciones entre frases declarativas atómicas (no se pueden descomponer en elementos más simples). El lenguaje se define a partir de átomos y conectivas, siendo estas la conjunción ($\wedge$), la disyunción ($\vee$), la negación ($\neg$) y el condicional ($\rightarrow$).

Las fórmulas válidas de lógica de enunciados son las que se pueden derivar a partir de esta gramática:

1. Si P es una sentencia atómica entonces P es una fórmula.
2. Si A y B son fórmulas entonces $(A \wedge B)$ es una fórmula
3. Si A y B son fórmulas entonces $(A \vee B)$ es una fórmula
4. Si A y B son fórmulas entonces $(A \rightarrow B)$ es una fórmula
5. Si A es una fórmula entonces $\neg A$ es una fórmula

Por convención, se consideran sentencias atómicas todas las letras mayúsculas a partir de la P y fórmulas todas las letras mayúsculas a partir de la A.

Mediante este lenguaje podemos expresar sentencias que relacionan cualquier frase declarativa atómica mediante las cuatro conectivas básicas y su semántica asociada. Esta semántica viene definida

por lo que se denomina *teoría de modelos*, que establece la relación entre las fórmulas y los valores de verdad y falsedad, es lo que se denomina una interpretación. Cada conectiva tiene su tabla de verdad[1] que establece como se combinan los valores de verdad de los átomos para obtener el valor de verdad de la fórmula.

A partir de los enunciados expresados mediante este lenguaje se pueden obtener nuevos enunciados que se deduzcan de ellos o se pueden plantear enunciados y comprobar si se derivan de ellos. Para ello existen diferentes metodologías de validación de razonamientos que establecen los pasos para resolver esas preguntas.

Metodologías de validación de razonamientos hay bastantes y se pueden dividir en dos grupos. Las que se basan en la semántica, buscan demostrar que los modelos que hacen verdad un conjunto de enunciados incluyen los modelos que hacen verdad la conclusión que queremos probar. Las que se basan en sintaxis aprovechan la estructura algebraica del lenguaje de la lógica de enunciados (álgebra de boole) para demostrar que el enunciado satisfactible/insatisfactible es accesible utilizando los enunciados premisa y la conclusión.

El método más sencillo para la validación de enunciados es el *método de resolución*. Es un método que funciona por refutación, es decir, podemos saber si un enunciado se deriva de un conjunto de enunciado probando que al añadir su negación podemos obtener el enunciado insatisfactible.

Este método se basa en la aplicación de la regla de resolución:

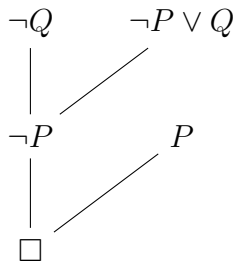
$$A \vee B, \neg A \vee C \vdash B \vee C$$

al conjunto de clausulas que se obtienen de transformar las premisas y la negación de la conclusión a forma normal conjuntiva. Existen diferentes variantes del algoritmo, la mas común es la que utiliza la denominada estrategia de conjunto de soporte, que inicia el algoritmo escogiendo alguna de las cláusulas de la negación de la conclusión. Este método es sólido y completo[2] si se demuestra la satisfactibilidad de las cláusulas de las premisas.

Ejemplo 2.1 *Podemos plantearnos un problema de lógica proposicional donde tengamos los enunciados "Está lloviendo" y "Me mojo". Asignamos a los enunciados las letras de átomo P y Q respectivamente. Podemos escribir la fórmula $P \to Q$, que se puede leer como "Si esta lloviendo entonces me mojo" y plantearnos el siguiente razonamiento:*

$$P \to Q, P \vdash Q$$

Podemos utilizar el método de resolución para validar este razonamiento obteniendo el siguiente conjunto de cláusulas con el razonamiento $\mathcal{C} = \{\neg P \vee Q, P, \neg Q\}$. Con estas cláusulas podemos obtener el siguiente árbol de resolución validando el razonamiento:



[1] Es algo que ya conocéis de la asignatura de lógica.

[2] Las propiedades de solidez y completitud (*soundness*, *completness*) son las que se piden a cualquier método de validación. La propiedad de solidez nos garantiza que el método solo nos dará conclusiones válidas y la completitud nos garantiza que podremos obtener todas las conclusiones derivables.

2.3 Lógica de predicados

La capacidad expresiva de la lógica de enunciados no es demasiado grande, por lo que habitualmente se utiliza como método de representación la lógica de predicados. Esta amplía el vocabulario de la lógica de enunciados añadiendo parámetros a los átomos, que pasan a representar propiedades o relaciones, también añade lo que denominaremos términos que son de tres tipos: variables (elementos sobre los que podremos cuantificar o substituir por valores), constantes (elementos identificables del dominio) y funciones (expresiones que pueden aplicarse a variables y constantes que denotan el valor resultante de aplicar la función a sus parámetros). A estos elementos se les añaden los cuantificadores universal ($\forall$) y existencial ($\exists$) que son extensiones infinitas de las conectivas conjunción y disyunción.

Esto amplía la gramática de fórmulas que podemos expresar, será una fórmula válida de lógica de predicados toda aquella que cumpla:

1. Si P es un predicado atómico y $t_1, t_2, ..., t_n$ son términos, entonces $P(t_1, t_2, ..., t_n)$ es una fórmula.
2. Si A y B son fórmulas entonces $(A \wedge B)$ es una fórmula
3. Si A y B son fórmulas entonces $(A \vee B)$ es una fórmula
4. Si A y B son fórmulas entonces $(A \rightarrow B)$ es una fórmula
5. Si A es una fórmula y v es una variable, entonces $(\forall vA)$ y $(\exists vA)$ son una fórmula
6. Si A es una fórmula entonces $\neg A$ es una fórmula

Por convención, se consideran sentencias atómicas todas las letras mayúsculas a partir de la P y fórmulas todas las letras mayúsculas a partir de la A. Para los términos, las constantes se denotan por letras minúsculas a partir de la a, las variables son letras minúsculas a partir de la x y las funciones letras minúsculas a partir de la f.

Mediante este lenguaje podemos expresar propiedades y relaciones entre objetos constantes de un dominio o establecer propiedades universales o existenciales entre los elementos del dominio. La visión que tenemos de la realidad está compuesta de elementos (las constantes) que podemos ligar mediante propiedades y relaciones. Podríamos asociar esta visión a una noción conjuntista de la realidad, los predicados unarios corresponderían a los conjuntos en los que podemos tener las constantes, los predicados con mayor aridad corresponderían a las relaciones que podemos establecer entre los elementos de esos conjuntos. En este caso la cuantificación nos permite establecer enunciados que pueden cumplir todos los elementos de un conjunto o enunciar la existencia en el conjunto de elementos que cumplen cierta propiedad o relación.

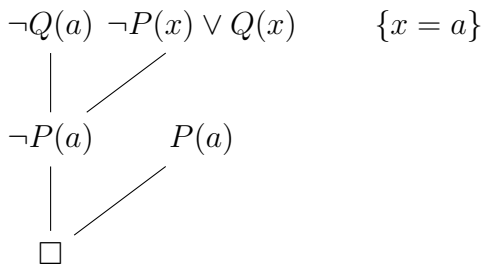
El método de resolución también se puede aplicar como método de validación en lógica de predicados, pero necesita de un conjunto de procedimientos adicionales para tener en cuenta las extensiones que hemos hecho en el lenguaje. En este caso las fórmulas son transformadas a la *forma normal de skolem*, que permite substituir todas las cuantificaciones existenciales por constantes y expresiones funcionales[3]. El método exige que para la aplicación de la regla de resolución los átomos que se quieran resolver permitan la unificación de los términos de los parámetros. La unificación lo que busca es la substitución de los valores de las variables de los términos de los átomos que permitan que las dos expresiones sean iguales.

Ejemplo 2.2 *Podemos plantearnos el problema de lógica de predicados con los siguientes predicados "ser un hombre", "ser mortal" y una constante "Socrates". Asignamos a los predicados las letras P y Q respectivamente y a la constante la letra a . Podemos escribir la fórmula $\forall x(P(x) \rightarrow Q(x))$ (todos*

[3]Tenéis los detalles en los apuntes de la asignatura de lógica.

los hombres son mortales) y la fórmula $P(a)$ (Sócrates es un hombre) y podemos intentar validar la fórmula $Q(a)$ (Sócrates es mortal).

Podemos utilizar el método de resolución para validar este razonamiento obteniendo el siguiente conjunto de cláusulas con el razonamiento $\mathcal{C} = \{\neg P(x)) \vee Q(x), P(a), \neg Q(a)\}$. Con estas cláusulas podemos obtener el siguiente árbol de resolución validando el razonamiento:



2.4 Lógica y representación del conocimiento

El lenguaje de lógica de predicados puede ser bastante incómodo para expresar cierto tipo de conocimiento y existen extensiones o modificaciones que intentan facilitar su uso. Por ejemplo, extensiones que hacen algo más cómodo el uso de lógica de predicados es incluir la igualdad como un predicado especial o permitir el utilizar tipos en las variables que se cuantifican.

Existen otros aspectos que son difíciles de tratar directamente en lógica de predicados, como por ejemplo:

- Representación del tiempo: En lógica de predicados toda fórmula se refiere al presente. Existen diferentes lógicas para el tratamiento del tiempo que permiten expresar relaciones temporales entre los elementos del dominio.
- El conocimiento entre agentes, razonamiento sobre otros: Las fórmulas que expresamos son el conocimiento propio, pero no tratamos nuestra creencia sobre el conocimiento que puedan tener otros. Las diferentes lógicas de creencias intentan establecer métodos que permiten obtener deducciones sobre lo que creemos que otros agentes creen.
- El conocimiento incierto e impreciso: La lógica de predicados supone que podemos evaluar con total certidumbre el valor de verdad de cualquier formula, pero en la realidad eso no es siempre así. Las lógicas probabilística y posibilística intentan trabajar con esa circunstancia y permitir obtener deducciones a partir de la certidumbre que tenemos sobre nuestro conocimiento.

3. Sistemas de reglas de producción

3.1 Introducción

La lógica de predicados también se puede ver como un mecanismo para describir procedimientos. Con este tipo de visión lo que hacemos es describir un problema como si fuera un proceso de razonamiento. Los predicados que utilizamos descomponen el problema en problemas más sencillos e indican un orden total o parcial que permitirá al sistema que ejecute esta descripción resolverlo.

En este tipo de descripción utilizamos un conjunto restringido de las expresiones que se pueden representar mediante el lenguaje de la lógica. La restricción fundamental que se impone es que solo se pueden utilizar fórmulas lógicas cuantificadas universalmente que se puedan transformar a cláusulas disyuntivas[1] en las que solo haya como máximo un átomo afirmado. Este tipo de cláusulas se denominan **cláusulas de Horn**.

La justificación de esta limitación es que la representación tendrá que ser ejecutada como una demostración y el coste computacional de los procedimientos para realizarla está ligado a la expresividad empleada en su descripción. A pesar de esta restricción del lenguaje podremos describir un gran número de problemas.

Este tipo de fórmulas se corresponde con lo que denominaremos **reglas de producción**. Las reglas son fórmulas condicionales en las que puede haber cualquier número de átomos en el antecedente y solo un átomo en el consecuente, por ejemplo:

$$\forall x \forall y (Persona(x) \wedge Edad(x, y) \wedge Mayor(y, 18) \rightarrow Mayor\_de\_edad(x))$$

En la práctica los lenguajes de programación que permiten describir problemas mediante reglas son más flexibles en su sintaxis y permitir cosas mas complejas. De hecho el consecuente de una regla puede ser una acción sobre la descripción del estado (modificación de los predicados actuales), un nuevo hecho del estado, un hecho que usamos como elemento de control de la búsqueda, una interacción con el usuario, ...

3.2 Elementos de un sistema de producción

Mediante este formalismo podremos ser capaces de describir como solucionar un problema, declarando el conjunto de reglas a partir del cual se puede obtener una solución mediante un proceso de demostración. La diferencia de las reglas respecto a otros formalismos procedimentales (como un programa implementado en un lenguaje imperativo) es que no explicitamos directamente el algoritmo a utilizar para encontrar la solución del problema, sino que se indican las condiciones que esta ha de cumplir la solución. Es un mecanismo de demostración el que se encarga de encontrarla combinando los pasos indicados por las reglas, ya que lo que estamos haciendo es validar un razonamiento que se fundamenta en las reglas descritas.

Esta forma de describir procedimientos se encuadra dentro de los denominados *lenguajes declarativos* (a diferencia de los lenguajes imperativos, que son los más comunes). La ventaja de declarar las condiciones que cumplen las soluciones es que no estamos limitados a una única secuencia de ejecución, un conjunto de reglas pueden describir muchos algoritmos distintos que comparten parte

[1] Una cláusula disyuntiva es una fórmula en la que solo se usa la conectiva disyunción.

de su especificación. Este tipo de lenguajes permiten además expresar programas a un mayor nivel de abstracción ya que no tenemos que expresar exactamente como se debe conseguir la solución, solo sus condiciones, ya que hallar la solución que cumple las condiciones es labor del mecanismo de demostración que utilizaremos.

Para poner este formalismo en perspectiva, podemos hacer la analogía con los algoritmos de búsqueda que hemos visto en los primeros capítulos. Las reglas son equivalentes a los operadores de búsqueda, y el algoritmo de búsqueda utilizado es un mecanismo de validación de demostraciones. Un número relativamente reducido de operadores puede permitir hallar soluciones a problemas muy diferentes y todos ellos se generan a partir de la combinación de los operadores.

Al conjunto de reglas se le denomina la **base de conocimiento** (o de reglas). Estas reglas pueden describir la resolución de múltiples problemas, será la labor del mecanismo que resuelva el problema concreto planteado el que decida qué reglas se han de utilizar.

El planteamiento del problema se realiza mediante **hechos**. Estos hechos serán la descripción de las condiciones del problema mediante predicados primitivos, funciones, relaciones y constantes definidas en el problema. Por ejemplo:

```
Persona(juan)
Edad(juan,25)
```

Al conjunto de hechos que describen un problema se denomina **base de hechos**. Con una base de reglas y una base de hechos podemos plantear preguntas cuya respuesta permita obtener la solución del problema.

Ejemplo 3.1 *Podemos crear un conjunto de reglas que nos permitan saber cuando se puede servir una bebida a una persona en un bar. Podríamos describir el problema de la siguiente manera:*

> *"Toda persona mayor de 18 años es mayor de edad. Toda bebida que contenga alcohol es una bebida restringida a mayores de edad. Se puede servir a una persona una bebida restringida a mayores de edad si es mayor de edad."*

Y lo podríamos describir mediante reglas de como la siguiente base de reglas:

```
si Persona(X) y Edad(X,Y) y Mayor(Y,18) entonces Mayor_de_edad(X)
si Bebida(X) y Contiene(X,alcohol) entonces Restringida_a_mayores(X)
si Mayor_de_edad(X) y Restringida_a_mayores(Y) entonces Servir(X,Y)
```

Podríamos entonces plantear un problema con un conjunto de hechos:

```
Persona(juan)
Edad(juan,25)
Bebida(cubata)
Contiene(cubata,alcohol)
```

Y entonces preguntarnos si le podemos servir un cubata a juan `Servir(juan,cubata)`.

La forma en la que determinamos si existe y cual es la respuesta al problema planteado dependerá del método de demostración de razonamientos que utilicemos para validar el razonamiento que describen la base de conocimiento, la base de hechos y la pregunta que realizamos.

Este mecanismo estará implementado mediante lo que denominaremos el **motor de inferencias**.

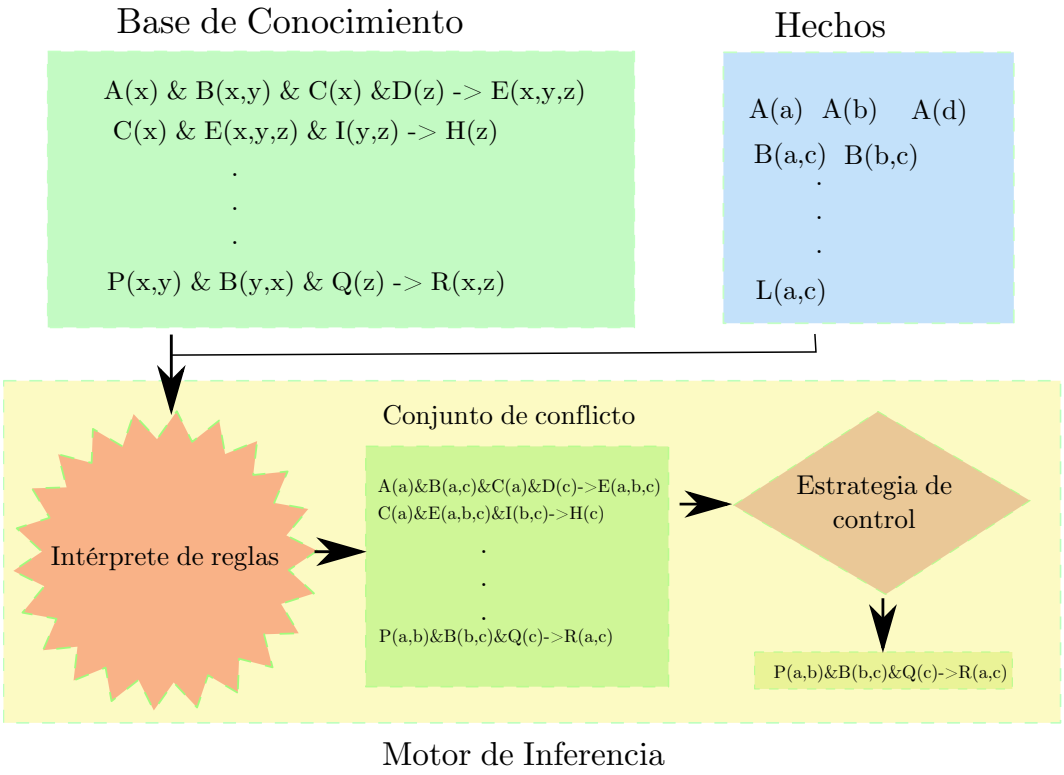


Figura 3.1: Sistema de producción

3.3 El motor de inferencias

La labor de un motor de inferencias es ejecutar la representación del problema descrita mediante la base de reglas y la base de hechos. Deberá ser por tanto capaz de demostrar razonamientos buscando la manera más eficiente de hacerlo. Para ello el motor de inferencias constará de dos elementos:

- **El intérprete de reglas:** Deberá ser capaz de ejecutar las reglas. Esto incluye interpretar el antecedente de la regla, buscar las instanciaciones adecuadas para que el antecedente se cumpla con los hechos/objetivos conocidos y generar la deducción que representa el consecuente de la regla. Una de las labores mas importantes de este componente será generar lo que se conoce como el **conjunto de conflicto**. Éste contiene todas las reglas u objetivos que se pueden utilizar en un momento dado de la resolución de un problema.

  En los lenguajes de reglas, el intérprete de reglas puede ser bastante complejo, permitiendo lo que se podría encontrar en cualquier lenguaje de programación, como por ejemplo la interacción con el usuario. También podría permitir el uso de otras lógicas mas allá de la lógica de predicados como lógicas para información incompleta o para información con incertidumbre.

- **La estrategia de control:** Será el mecanismo que permitirá al motor de inferencia elegir la regla u objetivo que deberá resolver en primer lugar y se basará en lo que denominaremos **estrategia de resolución de conflictos**. Dependiendo de la eficacia de esta estrategia, el coste final en tiempo de la resolución de un problema puede variar enormemente.

En la figura 3.1 se puede ver una representación de los elementos de un sistema de producción.

Respecto a la estrategia de resolución de conflictos, esta puede ser de muy diversas naturalezas, pudiendo utilizar información sintáctica local de las reglas o estrategias mas complejas. Por ejemplo se podría utilizar como criterio de elección:

- La primera regla según el orden establecido en la base de conocimiento. El experto conoce en que orden se deben ejecutar las reglas en caso de conflicto y lo indica en la base de reglas.

- La regla mas/menos utilizada. La regla más utilizada puede ser la correcta en la mayoría de los casos y eso le daría preferencia. También se podría dar preferencia al criterio contrario si por ejemplo las reglas menos usadas tratan excepciones y precisamente estamos resolviendo un caso excepcional.

- La regla más específica/más general. Las reglas más generales suelen utilizarse al comienzo de la resolución para plantear el problema y dirigir la búsqueda. Las reglas más específicas suelen ser útiles una vez ya hemos encaminado la resolución hacia un problema concreto.

- La regla con la instanciación de hechos mas prioritarios. Podemos indicar que hechos son mas importantes en la resolución y utilizar primero las reglas que los instancien.

- La regla con la instanciación de hechos mas antiguos/mas nuevos. Es lógico pensar que los últimos hechos deducidos deberían ser los primeros en utilizarse, pero también es posible que los hechos mas nuevos sean erróneos y que la manera mejor para redirigir la búsqueda es utilizar hechos antiguos de una manera diferente.

- Ordenadamente en anchura/en profundidad

- Aleatoriamente

Ninguno de estos criterios (en ocasiones contradictorios) garantiza por si solo hallar la solución siguiendo el camino más eficiente, por lo que por lo general se suele utilizar una combinación de criterios. Habitualmente se utiliza metaconocimiento en forma de metareglas (reglas sobre el uso de las reglas) para hacer mas eficiente la resolución y para cambiar la estrategia de resolución de conflictos dinámicamente.

3.3.1 Ciclo de ejecución

El motor de inferencias ejecuta un conjunto de fases que definen su funcionamiento. Estas fases son las siguientes:

1. **Fase de detección:** En esta fase el intérprete de reglas determinará todas las instanciaciones posibles entre los hechos u objetivos del problema y las reglas de la base de reglas. El objetivo de esta fase es calcular el **conjunto de conflicto**. Esta será la fase más costosa del ciclo de ejecución ya que el número de posibles instanciaciones de las reglas puede ser elevado.

2. **Fase de selección:** Se elegirá la regla a utilizar en este paso del ciclo de ejecución, utilizando como criterio la estrategia de resolución de conflictos. La elección que se realice determinará la forma en la que se haga la exploración de las posibles soluciones del problema.

3. **Fase de aplicación:** El intérprete de reglas propagará las instanciaciones de las condiciones de la regla a la conclusión ejecutándola y cambiando el estado del problema. Esto puede significar la obtención de nuevos hechos o el planteamiento de nuevos objetivos.

El ciclo de ejecución del motor de inferencias se acabará en el momento en el que se haya obtenido el objetivo deseado, ya no haya más reglas que aplicar o se hayan cubierto todos los objetivos que se hayan planteado durante la resolución del problema.

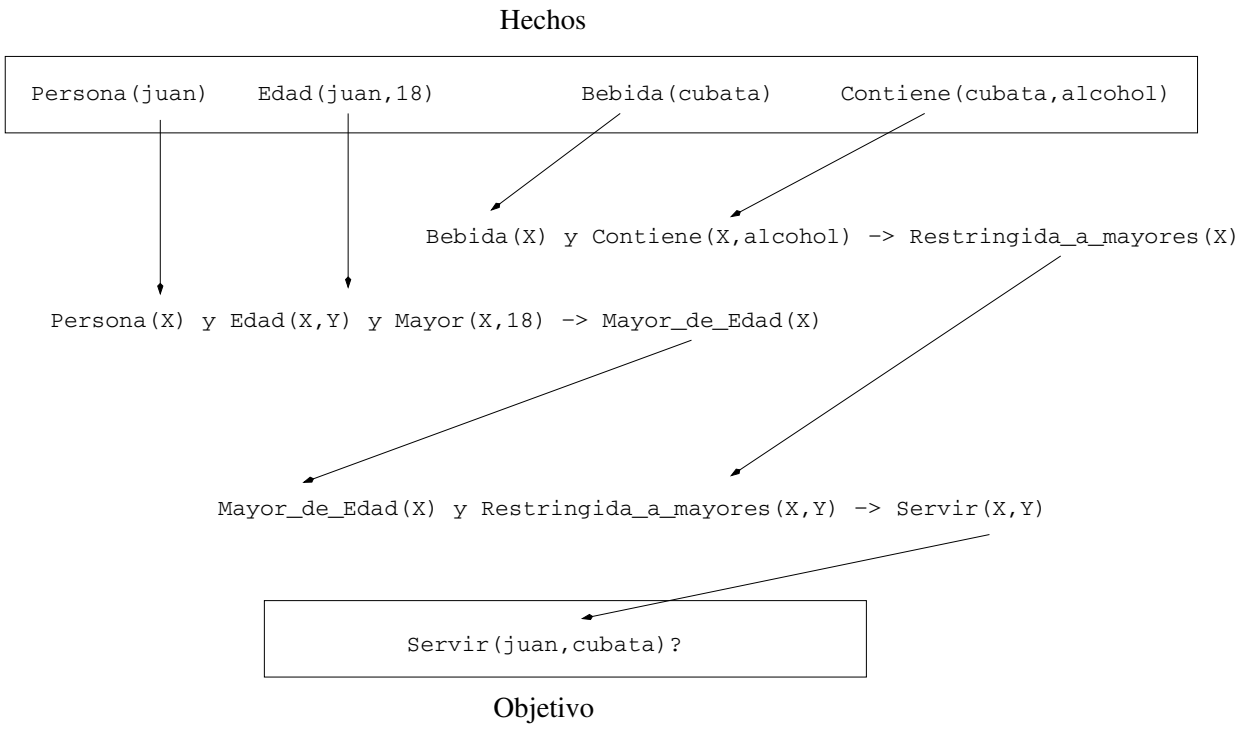


Figura 3.2: ¿Le servimos a Juan un cubata?

3.4 Tipos de razonamiento

La forma de resolver el razonamiento que plantea un conjunto de reglas y hechos depende del punto de vista que adoptemos. Si vemos el problema desde un punto de vista global, lo que se hace al validar el razonamiento es ligar lo hechos que hay en la base de hechos con el objetivo que tenemos que probar mediante las reglas de las que disponemos.

La forma en la que ligamos hechos y objetivos es simplemente encadenando las reglas una detrás de otra adecuadamente, de manera que las instanciaciones de las condiciones del antecedente y conclusiones del consecuente cuadren. Si pensamos que estamos hablando de un método de representación procedimental, de hecho lo que estamos haciendo es ensamblar el algoritmo que resuelve el problema.

Podemos ver en la figura 3.2 el planteamiento del ejemplo que vimos anteriormente. Ligamos los hechos de la base de hechos con el objetivo planteado instanciando las reglas que tenemos. En principio no hay una dirección preferente en la que debamos ligar hechos y objetivos. El problema lo podemos resolver partiendo de los hechos y deduciendo nuevos hechos a partir de las reglas hasta llegar al objetivo, este es el sentido en el que estamos habituados a utilizar las reglas. Pero también lo podríamos resolver mirando las reglas en el sentido contrario, una regla descompondría un problema (el consecuente) en problemas más pequeños (antecedente) y podríamos iniciar la búsqueda de la solución en el objetivo final hasta descomponerlo en los hechos de la base de hechos.

Estas dos estrategias se conocen como **razonamiento guiado por los datos** o **razonamiento hacia adelante** (*forward chaining*) y **razonamiento guiado por los objetivos** o **razonamiento hacia atrás** (*backward chaining*). Los explicaremos con detalle en las siguientes secciones. Como complemento a estas estrategias existen métodos de razonamiento que utilizan una combinación de las dos (*estrategias híbridas*) para intentar obtener las ventajas de cada una y reducir sus inconvenientes.

Algoritmo 3.1 Razonamiento hacia adelante

Procedure: Razonamiento Hacia Adelante

Input: Base de hechos, Base de reglas, Objetivos
Sin_alternativas ← falso
while $\exists o(o \in Objetivos \wedge o \notin Base\_de\_hechos) \wedge \neg Sin\_alternativas$ **do**
    Conjunto_Conflicto ← Interprete.Antecedentes_satisfactibles(Base_de_hechos, Base_de_reglas)
    if $Conjunto\_Conflicto \neq \emptyset$ **then**
        Regla ← Estrategia_Control.Resolucion_Conflictos(Conjunto_Conflicto)
        Interprete.Aplicar(Base_de_hechos, Regla)
    else
        Sin_alternativas ← cierto

3.4.1 Razonamiento guiado por los datos

Este tipo de razonamiento supone que son los hechos los que dirigen el ciclo de ejecución del motor de inferencias. Por lo tanto, cada vez que hacemos la fase de selección, buscamos todas aquellas reglas cuyas condiciones podamos hacer válidas con cualquier posible combinación de los hechos que tenemos en ese momento. Tras la selección de la regla más adecuada se añade a la base de hechos el consecuente de la regla. La ejecución acaba cuando el objetivo es deducido.

Este tipo de razonamiento se denomina deductivo o progresivo y se fundamenta en la aplicación de la regla denominada de la eliminación del condicional o *modus ponens*. Esta regla se puede formalizar de la siguiente manera:

$$A, A \rightarrow B \vdash B$$

Parafraseando, esta regla nos dice que cuando tenemos un conjunto de hechos y estos hechos forman parte del antedecente de una expresión condicional, podemos deducir el consecuente como un hecho mas de nuestro razonamiento. El algoritmo 3.1 es una implementación de su funcionamiento.

El mayor inconveniente de esta estrategia de razonamiento es que el razonamiento no se guía por el objetivo a conseguir, sino que se obtienen todas las deducciones posibles ya sean relevantes o no para conseguirlo. Esto implica un gran coste computacional a la hora de resolver un problema ya que se hace mas trabajo del necesario para obtener el objetivo. Esto significa que la estrategia de resolución de conflictos es bastante importante a la hora de dirigir la exploración del problema. De hecho, para hacer eficiente este método es necesario metaconocimiento y la capacidad para dirigir el flujo de razonamiento mediante las propias reglas.

Otro problema computacional es el detectar las reglas que se pueden disparar en un momento dado, pues requiere computar todas la coincidencias posibles entre hechos y antecedentes de reglas. Este problema se puede resolver de manera eficiente mediante el denominado *algoritmo de Rete*. Este algoritmo permite mantener y actualizar todas las coincidencias entre hechos y antecedentes, de manera que se puede saber eficientemente si una regla se cumple o no.

Las ventajas de este tipo de razonamiento, vienen del hecho de que, es más natural en muchos dominios expresar el razonamiento de esta manera y, por lo tanto, es más fácil plantear reglas que resuelvan problemas mediante esta estrategia.

Este tipo de razonamiento también es adecuado en problemas en los que no exista un objetivo claro a alcanzar y lo que se busque sea explorar el espacio de posibles soluciones a un problema.

Algoritmo 3.2 Razonamiento hacia atrás

```
Procedure: Razonamiento Hacia Atrás
Input: Base de hechos, Base de reglas, Objetivos
Sin_alternativas ← falso
while Objetivos ≠ ∅ ∧ ¬Sin_alternativas do
    Objetivo ← Estrategia_Control.Escoger_Objetivo(Objetivos)
    Objetivos.Quitar(Objetivo)
    Conjunto_Conflicto ← Interprete.Consecuentes_satisfactibles(Objetivo, Base_de_reglas)
    if Conjunto_Conflicto ≠ ∅ then
        Regla ← Estrategia_Control.Resolucion_Conflictos(Conjunto_Conflicto)
        Objetivos.Añadir(Regla.Extraer_antecedente_como_objetivos())
    else
        Sin_alternativas ← cierto
```

3.4.2 Razonamiento guiado por los objetivos

Este tipo de razonamiento supone que es el objetivo el que dirige el ciclo del motor de inferencias. En este caso lo que hacemos es mantener una lista de objetivos pendientes de demostrar que se inicializa con el objetivo del problema. En la fase de selección miramos si entre los hechos está el objetivo actual, si es así lo eliminamos y pasamos al objetivo siguiente. Si no es así, se seleccionan todas aquellas reglas que permitan obtener el objetivo como conclusión. Una vez elegida la regla más adecuada, los predicados del antecedente de la regla pasan a añadirse a la lista de objetivos a cubrir. La ejecución acaba cuando todos los objetivos que han aparecido acaban siendo cubiertos mediante hechos del problema.

Este tipo de razonamiento se denomina inductivo o regresivo e invierte la utilización que hace el razonamiento hacia adelante de la regla del modus ponens. Esta visión puede tomarse como una forma de descomposición de problemas, en la que se parte del objetivo inicial y se va reduciendo en problemas cada vez más simples hasta llegar a problemas primitivos (los hechos). El algoritmo 3.2 es una implementación de su funcionamiento.

La ventaja principal de este método de razonamiento es el que al estar guiado por el objetivo, todo el trabajo que se realiza se encamina a la resolución del problema y no se hace trabajo extra. Esto hace que este tipo de razonamiento sea mas eficiente.

Como desventaja, nos encontramos con que plantear un conjunto de reglas que resuelva problema de esta manera es menos intuitivo. El problema debe pensarse para poder resolverse mediante una descomposición jerárquica de subproblemas. También tenemos el inconveniente de que solo se puede aplicar a problemas en los que el objetivo sea conocido, lo cual reduce los dominios en los que puede ser aplicado.

3.5 Las reglas como lenguaje de programación

El usar reglas como mecanismo de resolución de problemas ha llevado a verlas como el fundamento de diferentes lenguajes de programación generales. Estos lenguajes entran dentro de los paradigmas de *programación declarativa* y *programación lógica*. En estos lenguajes los programas solamente definen las condiciones que debe cumplir una solución mediante un formalismo lógico y es el mecanismo de demostración el que se encarga de buscarla explorando las posibilidades.

La mayoría de estos lenguajes no son lenguajes declarativos puros e incluyen mecanismos y es-

tructuras que pueden encontrarse en otros paradigmas, pero aún así la filosofía de programación es bastante diferente y requiere un cambio en la manera de pensar.

En este tipo de lenguajes la asignación no existe y la comunicación entre reglas se realiza mediante la unificación de las variables que se utilizan. Estas variables no se pueden ver como las que se usan en programación imperativa ya que su valor lo toman durante el proceso de prueba de deducción al comparar hechos y condiciones y realizar su unificación. Por lo tanto no son lugares donde nosotros vayamos poniendo valores, los toman cuando es adecuado para demostrar el problema. En estos lenguajes la recursividad tiene un papel fundamental a la hora de programar.

Todas estas características pueden parecer una limitación expresiva, pero en realidad no lo son y la potencia expresiva de estos lenguajes es la misma que la de los lenguajes imperativos. Existen dominios de aplicación en los que estos estilos de programación son más adecuados que los lenguajes imperativos, la inteligencia artificial es uno de ellos.

Ejemplo 3.2 *El cálculo del factorial es un ejemplo clásico, podemos definir cuales son las condiciones que debe cumplir un número para ser el factorial de otro. Sabemos que hay un caso trivial que es el factorial de 1. Podemos declarar el predicado que asocie a 1 con su factorial* `fact(1,1)`.

Para calcular que un número es el factorial de otro lo único que tenemos que hacer es especificar las condiciones para que han de cumplirse. Por ejemplo:

si `=(X, X1+1)` *y* `fact(X1,Y1)` *y* `=(Y,Y1*X)` *entonces* `fact(X,Y)`

Parafraseando podríamos leer esta regla como que si hay un `X` *que sea* `X1 + 1` *y el factorial de* `X1` *es* `Y1` *y hay un* `Y` *que sea* `Y1*X` *entonces el factorial del* `X` *será* `Y`.

Supondremos que el predicado `=` *es un predicado especial que unifica el primer parámetro con el segundo.*

Podemos plantearnos la pregunta de cual es el factorial de 3, o sea si existe un valor `Z` *asociado al predicado* `fact(3,Z)`. *Mediante el mecanismo de razonamiento hacia atrás podremos averiguar cual es el valor de* `Z`.

Renombraremos las variables cada vez que usemos la regla para evitar confusiones.

`fact(3,Z)`
según la regla se puede descomponer en tres subobjetivos
`=(3,X1+1), fact(X1,Y1),=(Z,Y1*3)`
Evidentemente `X1` *se unificará con 2 en la primera condición quedándonos*
`fact(2,Y1),=(Z,Y1*3)`
Volveremos a descomponer el problema según la regla
`=(2,X1'+1), fact(X1',Y1'),=(Y1,Y1'*2),=(Z,Y1*3)`
Ahora `X1'` *se unificará con 1*
`fact(1,Y1'),=(Y',Y1'*2),=(Y,Y1*3)`
Sabemos por los hechos que `fact(1,1)`
`=(Y1,1*2),=(Z,Y1*3)`
`Y1` *se unificará con 2*
`=(Z,2*3)`
Z se unificará con 6 obteniendo la respuesta `Z=6`

Esto puede parecer la manera habitual con la que calculamos el factorial en un lenguaje imperativo, pero el paradigma declarativo tiene mayor versatilidad, porque sin cambiar el programa podríamos preguntar por ejemplo cual es el número que tiene como factorial 6 (`fact(X,6)`*) o incluso que nos calcule todos los factoriales que existen (*`fact(X,Y)`*). El mecanismo de deducción nos hallará la respuesta a todas estas preguntas. En un lenguaje imperativo deberíamos escribir un programa para responder a cada una de ellas.*

Utilizando un mecanismo de razonamiento hacia adelante también podríamos obtener el mismo cálculo, en este caso ni siquiera necesitamos de un objetivo, si ponemos en marcha el motor de inferencias obtendremos tantos factoriales como queramos:

`=(X,X1+1), fact(X1,Y1),=(Y,Y1*X)`
Instanciando `X1` *a 1 e* `Y1` *a 1 con el hecho* `fact(1,1)` *obtendremos el nuevo hecho* `fact(2,2)`
`=(X,X1+1), fact(X1,Y1),=(Y,Y1*X)`
Instanciando `X1` *a 2 e* `Y1` *a 2 con el hecho* `fact(2,2)` *obtendremos el nuevo hecho* `fact(3,6)`
...

El lenguaje de programación lógica por excelencia es **PROLOG**. Es un lenguaje en el que el mecanismo de razonamiento que se sigue es hacia atrás. Esto hace que cuando se programa en este lenguaje se deba pensar que se ha de descomponer el problema mediante el formalismo de reglas en problemas mas simples que estarán declarados como hechos.

Existen otros lenguajes de reglas que utilizan motores de inferencia hacia adelante como por ejemplo **CLIPS**. En este caso la filosofía de programación es diferente, ya que estamos explorando para encontrar la solución. Las reglas se ven como elementos reactivos que se disparan en función del estado del problema y que van construyendo los pasos que llevan a la solución. Esto obliga a que, si se quiere seguir un camino específico de resolución, haya que forzarlo añadiendo hechos que lleven el control de la ejecución.

3.6 Las reglas como parte de aplicaciones generales

Los sistemas de producción y los motores de inferencia están adquiriendo en la actualidad bastante relevancia como método adicional para desarrollar aplicaciones fuera del ámbito de la inteligencia artificial.

En muchas ocasiones las decisiones que se toman dentro de una aplicación pueden variar con el tiempo (a veces bastante rápido) y no tiene mucho sentido programarlas dentro del código del resto de la aplicación, ya que eso implicaría tener que cambiarlo cada vez que se cambia la forma de tomar esas decisiónes.

En la terminología de este tipo de aplicaciones, las reglas implementarían lo que se denominan **las reglas de negocio**, que no son mas que algoritmos de decisión invocados a partir de un conjunto de datos, algo que concuerda de manera natural con el funcionamiento de los sistemas de producción.

Las reglas son útiles en este tipo de escenarios, ya que pueden mantenerse aparte del resto de procedimientos de la aplicación y ser ejecutadas mediante un motor de inferencia. El cambio de un mecanismo de decisión solo implicará cambiar las reglas adecuadamente, manteniendo intacta el resto de la aplicación.

Este tipo de soluciones están tomando cada vez mas relevancia sobre todo con la tendencia a hacer aplicaciones distribuidas y basadas en componentes, como por ejemplo las que entran dentro del ámbito de las arquitectura orientadas a servicio (Service Oriented Arquiectures). Estas últimas están bastante ligadas a las tecnologías basadas en agentes del ámbito de la inteligencia artificial.

El uso de motores de inferencia en lenguajes de programación imperativos como por ejemplo java está estandarizado (Java Rule Engine API,JSR 94). Ejemplos de sistemas que incluyen motores de inferencia como parte de sus herramientas de desarrollo son por ejemplo SAP NetWeaver, IBM WebSphere, Oracle Business Rules, JBoss Rules o Microsoft Business Rule Engine.

4. Representaciones estructuradas

4.1 Introducción

El lenguaje de la lógica debería permitir representar cualquier conocimiento que quisiéramos utilizar en un problema. No obstante su capacidad para expresar conocimiento se ve entorpecida por la dificultad que supone utilizar su lenguaje para formalizar conocimiento. Esta dificultad la podríamos comparar con la diferencia que existe entre programar en lenguaje máquina o usar un lenguaje de programación de alto nivel.

El lenguaje de la lógica se encuentra a demasiado bajo nivel como para permitir representar de manera sencilla las grandes cantidades de conocimiento necesarias para una aplicación práctica. Es por ello que se crearon las representaciones estructuradas como lenguajes que se acercan más a como estamos nosotros acostumbrados a utilizar y describir el conocimiento.

Estas representaciones solucionan muchos problemas de representación y hacen mas fácil la adquisición de conocimiento. No obstante son restricciones del lenguaje de la lógica de predicados, por lo que no pueden representar cualquier conocimiento.

4.2 Redes semánticas

Podemos definir de forma genérica a una red semántica como un formalismo de representación del conocimiento donde éste es representado como un grafo donde los nodos corresponden a conceptos y los arcos a relaciones entre conceptos. En la figura 4.1 se puede ver un ejemplo.

La justificación del uso de este tipo de representación para formalizar conocimiento viene desde la psicología cognitiva, que es el área donde se definieron por primera vez las redes semánticas. Según esta, la representación que utilizamos las personas para almacenar y recuperar nuestro conocimiento se basa en la asociación entre conceptos mediante sus relaciones.

Este formalismo ha ido evolucionando desde su definición para permitir por un lado, una formalización de su semántica y sus capacidades (a partir de lógica de predicados), y por otro, la definición de lenguajes y formalismos que permitan su uso computacional.

A lo largo de su historia han aparecido muchos formalismos de redes semánticas, ampliando sus capacidades permitiendo formalizar diferentes aspectos del conocimiento o especializándolas para dominios concretos[1]. De hecho, hay muchos lenguajes de representación que se usan habitualmente, tanto dentro del ámbito de la inteligencia artificial, como en otros ámbitos, que caen bajo la definición de las redes semánticas.

> Por ejemplo, el UML o los diagramas de entidad relación puede verse como una notación de red semántica especializada que permiten describir entidades especializadas, ya sea qué elementos componen una aplicación y como se relaciones entre ellos, o como se describe una base de datos. Sobre estas notaciones se define de manera precisa el significado de cada uno de los elementos para permitirnos hacer diferentes inferencias a partir del diagrama.

[1] Un ejemplo de esta especialización son los múltiples formalismos de redes semánticas aparecidos para el tratamiento del lenguaje natural.

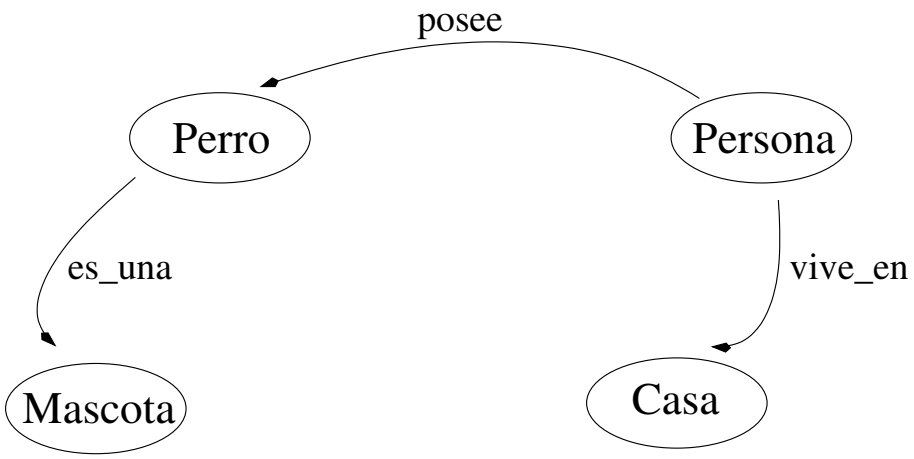


Figura 4.1: Ejemplo de red semántica

Sobre la representación básica (grafo de conceptos y relaciones) se pueden definir mecanismos de razonamiento que permiten responder a cierto tipo de preguntas, como por ejemplo:

- ¿Cierta pareja de conceptos están relacionados entre si?
- ¿Qué relaciona dos conceptos?
- ¿Cual es el concepto más cercano que relaciona dos conceptos?

La evolución de las redes semánticas ha ido añadiéndoles nuevas características que permiten una mejor estructuración del conocimiento, distinguiendo, por ejemplo, entre conceptos/instancias/valores o entre relaciones/propiedades. También han enriquecido la semántica de las relaciones para poder hacer inferencias más complejas, como, por ejemplo, creando relaciones que indiquen taxonomía (clase/subclase/instancia). Veremos todas estas características en la siguiente sección.

4.3 Frames

Los frames evolucionaron a partir de las redes semánticas y se introdujeron a partir de 1970. Estructuran el formalismo de las redes semánticas y permiten una definición detallada del comportamiento de los elementos que se definen. Por un lado un **concepto** (*frame*) es una colección de **atributos** (*slots*) que lo definen y estos tienen una serie de características y restricciones que establecen su semántica (*facets*). Una **relación** conecta conceptos entre sí y también tiene una serie de características que definen su comportamiento.

De entre las relaciones que se pueden definir se establecen como especiales las relaciones de naturaleza taxonómica (clase/subclase, clase/instancia, parte/subparte, ...). Estas permiten establecer una estructura entre los conceptos (Clases/subclases/instancias) permitiendo utilizar relaciones de generalización/especialización y herencia de atributos como sistema básico de razonamiento[2].

Estos elementos definen la parte declarativa de la representación. A partir de ellos se puede razonar sobre lo representado y hacer preguntas sobre los conceptos. El mecanismo de razonamiento se basa en la denominada **lógica de la descripción** (*Description Logics*). Esta lógica[3] es una restricción de la lógica de predicados que permite describir y trabajar con descripciones de conceptos. Su

[2]El sistema de representación es muy parecido al que se utiliza en orientación a objetos, de hecho se crearon en la misma época, aunque los elementos esenciales provienen de las redes semánticas, que son una década anteriores.

[3]De hecho son múltiples lógicas que se diferencian por el nivel de expresividad que se permite.

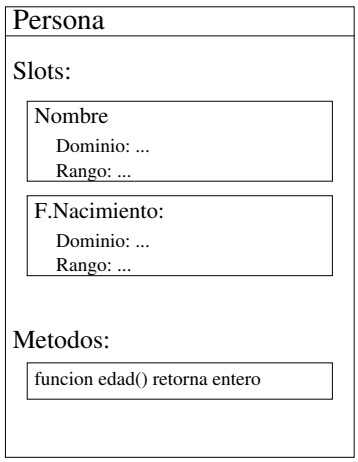


Figura 4.2: Ejemplo de frame

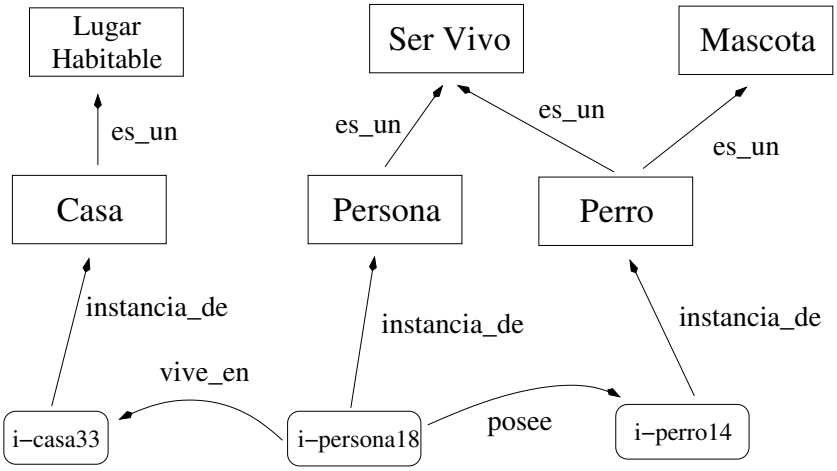


Figura 4.3: Ejemplo de red de frames

principal interés es que define algoritmos eficientes de razonamiento sobre definiciones y propiedades. Su principal hándicap es que esta eficiencia se obtiene a costa de no permitir formalizar ciertos tipos de conceptos como por ejemplo negaciones o conceptos existenciales.

Este hándicap hace que muchas implementaciones de este formalismo de representación incluyan mecanismos procedimentales que permitan obtener de manera eficiente deducciones que no se pueden obtener mediante razonamiento. Estos procedimientos permiten resolver problemas concretos en la representación de manera ad-hoc.

Bajo estos mecanismos procedimentales caen lo que se denominan **métodos** que son procedimientos o funciones que pueden invocar los conceptos o las instancias (al estilo de la orientación a objetos) y los **demons** que son procedimientos reactivos que se ejecutan cuando se dan ciertas circunstancias en la representación.

En la figura 4.2 se puede ver un ejemplo de frame. En la figura 4.3 se puede ver una red de frames representando conceptos, instancias y diferentes relaciones.

La **herencia** es el principal mecanismo de razonamiento sobre valores y propiedades que se utiliza en los frames. La herencia es un mecanismo que puede resultar problemático por la circunstancia de que un atributo o valor podría ser heredado por diferentes caminos si permitimos que un concepto pueda ser subclase de varias superclases. Esta *herencia múltiple* puede hacer que no sepamos que valor o definición de una propiedad es la correcta[4].

Este problema a veces se puede resolver razonado sobre la representación e identificando qué definiciones de una propiedad ocultan a otras y cuál es la más cercana a donde queremos obtener el atributo o su valor. Para ello se define el *algoritmo de distancia inferencial* que permite saber si existe o no una definición que oculta a las demás. El algoritmo es el siguiente:

1. Buscar el conjunto de frames que permiten heredar el valor del slot $\longrightarrow$ **Candidatos**
2. Eliminar de Candidatos todo frame que sea padre de otro de la lista
3. Si el número de candidatos es:
   a) $0 \longrightarrow$ No se puede heredar el slot
   b) $1 \longrightarrow$ Ese es el valor que buscamos
   c) $> 1 \longrightarrow$ Problema de herencia múltiple si la cardinalidad del slot no es N

En ocasiones un problema de herencia múltiple se puede resolver ad-hoc usando los mecanismos procedimentales de la representación.

[4]Estos problemas hacen que por ejemplo lenguajes como java no permitan la herencia múltiple.

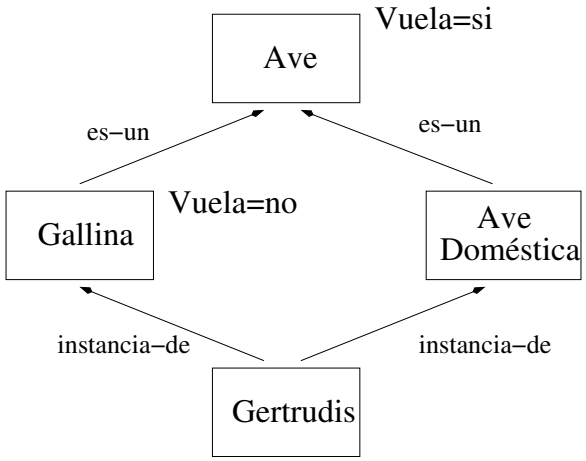


Figura 4.4: Ejemplo de frames

Ejemplo 4.1 *Podemos observar el problema de la herencia múltiple en la figura 4.4. La instancia podría tener simultáneamente el valor* ***si*** *y* ***no*** *para el atributo* ***vuela*** *ya que puede encontrar un camino de herencia para cada uno de ellos.*

En este caso el algoritmo de distancia inferencial empezaría con la lista que contendría {Gallina, Ave} como frames candidatos a dar el valor del slot. El frame Ave se eliminaría al ser ascendiente de Gallina, lo cual nos dejaría con el frame del que debemos heredar el slot.

La mayoría de los entorno para desarrollo de aplicaciones en inteligencia artificial poseen como parte del lenguaje una parte que permite representar conocimiento utilizando un mecanismo similar a los frames, aunque muchas veces aparece directamente como un lenguaje orientado a objetos con características adicionales. Veremos uno de estos lenguajes dentro del entorno de CLIPS.

También hay lenguajes de frames que no tienen parte procedimental incorporada y por lo tanto son implementaciones del lenguaje de la lógica de descripción. Un ejemplo de este tipo de lenguajes son los utilizados en la *web semántica*. Estos lenguajes se construyen sobre XML y definen los elementos básicos para crear conceptos, características y relaciones, el lenguaje que actualmente se utiliza es **OWL** (Ontology Web Language)[5].

4.3.1 Una sintaxis

Frames

Las sintaxis de los lenguajes de frames son muy variadas, en lugar de escoger una concreta, en esta sección vamos a definir un lenguaje de frames genérico que utilizaremos en la asignatura para resolver problemas.

Los conceptos se formarán a partir de propiedades y métodos, y se definirán a partir de la construcción `frame`, que tendrá la siguiente sintaxis:

Frame `<nombre>`
slot `<nombre-slot>`
slot `<nombre-slot>`
...
slot `<nombre-slot>`

[5]Los lenguajes para la web semántica han tenido una larga evolución, el primero fue **RDF**, que aún se sigue usando ámpliamente a pesar de sus limitaciones. Proyectos Europeos y Norteamericanos definieron sobre este lenguajes mas potentes (OIL,DAML) que acabaron fusionándose hasta llegar a OWL que es un estándar del W3C.

métodos
acción `<nombre-método> (parámetros) [H/noH]`
...
función `<nombre-método> (parámetros) devuelve <tipo> [H/noH]`

Slots

Un `slot` es un atributo que describe un concepto. Cada slot puede ir acompañado de modificadores (*facets*) que definirán las características del slot. Un slot puede ser redefinido en un subconcepto, por lo que puede volver a aparecer con características distintas que ocultan la definición anterior.

Estos modificadores permiten definir la semántica y el comportamiento del atributo e incluyen:

- *Dominio:* Conceptos que pueden poseer este slot
- *Rango:* Valores que puede tener el slot
- *Cardinalidad:* si se admite un único valor o múltiples valores
- *Valor por omisión:* Valor que tiene el slot si no se le ha asignado ninguno
- *Uso de demons:* Procedimientos que se ejecutarán si sucede un evento en el slot, estos procedimiento no tiene parámetros ya que no se pueden llamar explícitamente. Definiremos cuatro tipos de eventos que pueden suceder:
  - `If-needed` (al consultar el slot)
  - `if-added` (al asignar valor al slot),
  - `if-removed` (al borrar el valor)
  - `if-modified` (al modificar el valor)
- *Comportamiento en la herencia:* Si el slot se puede heredar a través de las relaciones.

La sintaxis que utilizaremos para definir un `slot` será la siguiente:

Slot `<nombre>`
++ dominio (lista de frames)
++ rango `<tipo-simple>`
++ cardinalidad (1 o N)
valor (valor o lista de valores)
demons `<tipo-demon>`
accion `<nombre-accion>` / **función**`<nombre-funcion> devuelve <tipo>*`
herencia (por rels. taxonómicas: SI/NO; por rels. usuario: SI/NO)

Los facets (propiedades) marcados con ++ son obligatorios en toda descripción de slot. Las acciones/funciones asociadas a los demons de los slots no tienen parámetros. Usan la variable `F` como referencia implícita al frame al cual pertenece el slot que activa el demon. Los demons de tipo `if-needed` solo pueden estar asociados a funciones.

Para la herencia, distinguiremos dos tipos de relaciones, las taxonómicas, que son las que definen la estructura entre los conceptos y estarán predefinidas y las relaciones de usuario que son las que podremos definir nosotros. La herencia a través de cada tipo se comporta de manera diferente. La herencia a través de las relaciones taxonómicas es de definición, de manera que si un concepto hereda un slot éste se encontrará físicamente en las instancias que creemos. La herencia a través de las relaciones de usuario es de valor, de manera que si un concepto hereda un slot solo podremos obtener

su valor en una instancia del concepto, si existe una relación con una instancia que posea ese slot y la relación nos permite heredarlo.

Consideraremos que por omisión tendremos herencia a través de la relaciones taxonómicas y no la tendremos a través de las de usuario.

Métodos

Los métodos serán procedimientos o funciones que permitirán realizar un cálculo concreto a partir de una clase o una instancia. Estos métodos de clase podrán hacer cálculos a partir de todas las instancias a las que tenga acceso a través de sus relaciones. Pueden ser heredables o no, ya que algunos casos podría tener sentido la herencia del método en sus subclases o instancias como una especialización de este. Los métodos se invocan de manera explícita, por lo que podrán tener parámetros.

Las acciones/funciones que describen los métodos usarán la variable `F` como referencia implícita al frame en el que se activa el método, y por ello no se ha de pasar como parámetro[6].

Relaciones

Las relaciones permiten conectar conceptos entre si, definiremos su comportamiento a partir de un conjunto de propiedades:

- *Dominio:* Conceptos que pueden ser origen de la relación.
- *Rango:* Conceptos que pueden ser destino de la relación.
- *Cardinalidad:* Número de instancias del rango con las que podemos relacionar una instancia del dominio.
- *Inversa:* Nombre de la relación inversa y su cardinalidad.
- *Transitividad:* Si es transitiva entre instancias.
- *Composición:* Como se puede obtener con la composición de otras relaciones.
- *Uso de demons:* Procedimientos que se ejecutarán si sucede un evento en la relación, estos procedimiento no tiene parámetros ya que no se pueden llamar explícitamente. Definiremos dos tipos de eventos que pueden suceder:
  - `If-added`: Si establecemos la relación entre instancias
  - `If-removed`: Si eliminamos la relación entre instancias
- *Slots heredables:* Slots que se pueden heredar a través de esta relación (solo el valor). Dado que las relaciones se definen bidireccionales, asumiremos que los slots se heredan en el sentido que corresponda, o sea, del frame en el que esta definido el slot al que lo debe heredar.

La sintaxis para definir relaciones es la siguiente:

Relación `<nombre>`
++ dominio (lista de frames)
++ `rango` (lista de frames)
++ `cardinalidad` (1 o N)
++ inversa `<nombre>` (cardinalidad: 1 o N)

[6]Es equivalente por ejemplo a la variable de autoreferencia `this` de java o C++.

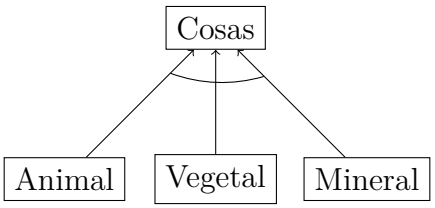


Figura 4.5: Subclasificación completa de un concepto

transitiva SI/NO [por defecto es NO]
compuesta NO/`<descripción de la composición>` [por defecto es NO]
demons (`<tipo-demon> acción <nombre-acción>`
herencia (lista de slots) [por defecto es lista vacía]

Los descriptores marcados con ++ son obligatorios en toda descripción de relación. Las acciones asociadas a los demons de las relaciones no tienen parámetros. Éstos usan las variables `D` y `R` como referencia implícita al frame origen y destino de la conexión que se está intentando añadir o eliminar entre los dos frames.

Respecto a la transitividad, para que podamos tener una relación transitiva su dominio y rango han de poder ser transportables entre instancias. Esto quiere decir que o el dominio y el rango de la relación están definidos sobre instancias de un mismo frame, o que en el rango y el dominio de la relación aparecen frames comunes. Por supuesto esto no es suficiente para que una relación sea transitiva, es necesario que la semántica de la relación lo permita.

Respecto a la composición, para que una relación sea compuesta la semántica de la relación ha de corresponder al resultado de la aplicación de dos o mas relaciones. Esto quiere decir que no es suficiente con que encontremos una cadena de relaciones entre una pareja de frames, esta cadena ha de significar lo mismo que la relación que estamos definiendo.

Como se ha comentado en el apartado sobre los slots, se distinguen dos tipos de relaciones. Por un lado están las taxonómicas, que están predefinidas y se limitan a `es-un` e `instancia-de`. La primera de ellas es una relación entre clases y la segunda entre instancias y clases. Por otro lado están las relaciones de usuario, que solo podrán ser entre instancias.

Sintaxis y elementos predefinidos

La expresión `<nombre-frame>.<nombre-relación>` nos dará el frame (si la cardinalidad es 1) o la lista de frames (si la cardinalidad es N) con los cuales esta conectado a través de la relación `<nombre-relación>`. Para consultar la cardinalidad se puede usar una funcion predefinida `card(<nombre-frame>.<nombre-relación>)`.

Tendremos predefinidas las siguientes relaciones taxonómicas:

- Relación `es-un` (inversa: `tiene-por-subclase`) transitiva SI
- Relación `instancia-de` (inversa: `tiene-por-instancia`) composición: `instancia-de` $\otimes$ `es-un`

Usaremos un arco para unir todas las relaciones `es-un` de un conjunto de subclases (ver figura 4.5) para indicar que las subclases son una partición completa del dominio, es decir, que no existen otras subclases que las definidas y que toda instancia debe pertenecer a una de esas subclases.

También supondremos que disponemos de las siguientes funciones booleanas predefinidas:

`<slot>?(<frame>)` Nos dice si `<frame>` posee este slot o no (activando la herencia si hace falta)

`<relación>?(<frame>)`

Nos dice si `<frame>` esta conectado con algún otro frame a través de la relación indicada por la función

`<relación>?(<frame-o>,<frame-d>)`

Nos dice si existe una conexión entre `<frame-o>` y `<frame-d>` etiquetada con la relación indicada por la función

5. Ontologías

5.1 Introducción

El estudio de la representación del conocimiento no es exclusivo de la inteligencia artificial, sino que es un tema que se origina desde el primer momento en el que el hombre se planteó el entender y describir el mundo en el que se encuentra.

A este área estudio se la encuadra dentro de la filosofía y fue Aristóteles quien acuñó el término **Categoría** como la palabra para describir las diferentes clases en las que se dividían las cosas del mundo. A este área de la filosofía se la conoce actualmente como *ontología*, término relativamente moderno (s. XIX) que proviene del griego *Ontos* (Ser) y *Logos* (Palabra), literalmente "las palabras para hablar de las cosas". Este término se empezó a utilizar para distinguir el estudio de la categorización del ser de la categorización que se hacia por ejemplo en biología. De hecho el trabajo de categorización surge en muchas áreas de la ciencia (filosofía, biología, medicina, lingüística, ...). La inteligencia artificial es una más de las áreas interesadas en este tema.

El objeto de estudio de la ontología son las entidades que existen en general o en un dominio y como se pueden agrupar de manera jerárquica en una categorías según sus diferencias y similaridades. El resultado de este estudio es lo que denominamos una **ontología**.

Una ontología se puede definir formalmente como:

> "Una ontología es un catálogo de los tipos de cosas que asumimos que existen en un dominio $\mathcal{D}$ desde la perspectiva de alguien que usa un lenguaje $\mathcal{L}$ con el propósito de hablar de $\mathcal{D}$. Los elementos de una ontología representan predicados, constantes, conceptos y relaciones pertenecientes a un lenguaje $\mathcal{L}$ cuando se usa para comunicar información sobre $\mathcal{D}$."
>
> (Sowa, 1999)

Otras definiciones menos formales podrían ser:

> "Una ontología es la definición de los términos básicos y relaciones que comprenden el vocabulario de un dominio y las reglas para poder combinarlos y definir extensiones a ese vocabulario."
>
> (Neches, 1991)

> "Una ontología es una especificación formal de una conceptualización compartida."
>
> (Borst, 1997)

Una ontología es pues un vocabulario, un conjunto de términos y expresiones que escogemos para representar una realidad, a través del cual comunicarnos con otras entidades que la compartan.

> Un ejemplo evidente de ontología es el diccionario de una lengua. Un diccionario es la recopilación de todas las palabras que usan los hablantes de un idioma para comunicarse. El diccionario es el conjunto de los términos que comparten los hablantes al referirse a la realidad. Cada palabra en el diccionario tiene descritas sus diferentes características de manera que los hablantes sepan como deben utilizarse y su significado. De hecho los diccionarios como descripción del conocimiento del dominio lingüístico son un elemento clave en la construcción de sistemas de tratamiento de lenguaje natural y habitualmente tienen una organización más estructurada de la que estamos acostumbrados a ver.

Lo que queda definido en una ontología se puede ver como una representación declarativa de un dominio. El uso, las maneras de combinar estos elementos y la obtención de información a partir de las expresiones formadas con los elementos de las ontologías pasa a través de la lógica.

La lógica se puede ver como un mecanismo de manipulación/ejecución que por sí misma no habla explícitamente sobre un dominio específico. Podemos ver que sus expresiones son neutras respecto al significado de sus átomos y predicados, es su combinación con una ontología lo que le da a un formalismo lógico la capacidad de expresar significados, por ejemplo:

$$\frac{\begin{array}{l} P \rightarrow Q \\ P \end{array}}{Q}$$

Este razonamiento no habla sobre nada en concreto salvo que asignemos significados a los átomos (P = llueve, Q = me mojo). A partir del momento en el que ligamos los átomos con los conceptos de la ontología estamos diciendo cosas y razonando sobre el dominio de esa ontología. El mismo razonamiento hablaría de algo distinto si cambiáramos la ontología que define el significado de los átomos.

5.2 Necesidad de las ontologías

Existen varios motivos por los que las ontologías son de utilidad en el ámbito de la inteligencia artificial:

1. *Permiten compartir la interpretación de la estructura de la información entre personas/agentes*

   Una ontología fija un conjunto de términos que denotan elementos de la realidad, el establecer una ontología sobre un dominio específico permite que dos agentes puedan entenderse sin ambigüedad y sepan a que se refieren. Eliminamos de esta manera la ambigüedad en la comunicación (al menos en lo que se refiere a los términos que se utilizan y su significado).

   Por ejemplo, un idioma es una ontología que comparten todos sus hablantes. El significado de cada término está recogido en un diccionario y cuando dos hablantes se comunican entre si, saben que tienen ese conjunto de términos y significados en común y asumen que cada uno entiende lo que el otro dice.

2. *Permiten reusar el conocimiento*

   Hacer una descripción de un dominio permite que esta pueda ser usada por otras aplicaciones que necesiten tratar con ese conocimiento. La descripción del conocimiento se puede hacer independiente de su uso y una ontología suficientemente rica puede ser utilizada en múltiples aplicaciones.

3. *Hacen que nuestras suposiciones sobre el dominio se hagan explícitas*

   Escribir una ontología exige la formalización al máximo detalle de todos los elementos del dominio y su significado y hacer explícitas todas las suposiciones que se van a utilizar. Esto facilita reflexionar sobre él y poder analizar las suposiciones realizadas para poder cambiarlas y actualizarlas de manera más sencilla. También ayuda a que otros puedan analizar y entender su descripción.

4. *Separan el conocimiento del dominio del conocimiento operacional*

   Permite hacer independientes las técnicas y algoritmos para solucionar un problema del conocimiento concreto del problema. De hecho una ontología es una descripción declarativa del dominio, de manera que no asume una metodología específica de utilización del conocimiento.

5. *Permiten analizar el conocimiento del dominio*

   Una vez tenemos una especificación del conocimiento podemos analizarlo utilizando métodos formales (para comprobar si es correcto, completo, consistente, ...)

5.3 Desarrollo de una ontología

En Inteligencia Artificial una ontología será una descripción formal explícita de los conceptos de un dominio (**Clases**). Estas clases se describirán a partir de **propiedades** que representarán las características, atributos y relaciones de las clases. Adicionalmente estas características tendrán **restricciones** (tipo, cardinalidad, ...). Finalmente tendremos **instancias** (elementos identificables) que constituirán los individuos concretos que representa la ontología.

Eso quiere decir que el desarrollo de una ontología requerirá definir las clases que forman el dominio, organizar las clases en una jerarquía taxonómica según sus afinidades, definir las propiedades de cada clase e indicar las restricciones de sus valores y asignar valores a las propiedades para crear instancias.

El cómo se realiza esto se puede encontrar en las diferentes metodologías de desarrollo de ontologías que hay descritas en la literatura de representación del conocimiento. Estas metodologías son bastante complejas, dado que en sí el desarrollo de una ontología para un dominio real es una labor compleja.

Para poder desarrollar ontologías pequeñas describiremos una metodología informal que permitirá analizar los elementos de un dominio y obtener un resultado que se puede utilizar en aplicaciones de complejidad media[1].

Antes de empezar con la metodología es necesario tener presente que:

1. No existe un modo *correcto* de modelar un dominio. La mejor solución dependerá de la aplicación/problema concreto.

2. El desarrollo de una ontología es un proceso iterativo.

3. Los objetos de la ontología deberían ser cercanos a los objetos y relaciones que se usan para describir el dominio (generalmente se corresponden a nombres y verbos que aparecen en frases que describen el dominio).

[1] Una descripción algo mas detallada de esta metodología y un ejemplo sencillo lo podéis encontrar en el artículo *"Ontology Development 101: A Guide to Creating Your First Ontology", Noy & McGuinness, (2000)* que encontrareis en la web de la asignatura.

Fases de desarrollo

1. **Determinar el dominio y la cobertura de la ontología**

   Deberemos saber qué elementos queremos que aparezcan en la ontología que vamos a desarrollar, por lo que debemos identificar qué parte del dominio nos interesa describir. Dependiendo del objetivo de uso de la ontología los términos que deberán aparecer pueden ser muy diferentes, por lo que es importante tenerlo claro.

   Una forma adecuada de saber qué elementos debemos representar es plantearnos que tipo de preguntas y respuestas deseamos de la ontología. Es lo que se denomina **preguntas de competencia**.

   También es interesante plantearse quién va a usar y mantener la ontología. No será lo mimo desarrollar una ontología restringida que vamos a utilizar en nuestra aplicación, que desarrollar una ontología de un dominio amplio que pretendemos reusar o que reusen otros.

   Ejemplo 5.1 *Supongamos que necesitamos una ontología para representar el funcionamiento de una facultad y todos los elementos que están relacionados con ella. Dependiendo del uso que vayamos a darle a la ontología necesitaremos unos conceptos u otros.*

   Si queremos utilizar la ontología para recomendar a un alumno de qué nuevas asignaturas se puede matricular el próximo cuatrimestre, seguramente necesitaremos describir que es una asignatura, como se organizan el plan de estudios y en los ciclos del plan de estudio, que temas tratan, cual es su carga de trabajo, posiblemente también nos interese guardar información histórica sobre ella. Además necesitaremos representar que es un expediente, que es una convocatoria, que es un horario, ...

   Si en cambio nos interesa hacer un programa que permita dialogar con un estudiante de bachillerato para responder sus dudas sobre como funciona la facultad tendremos que poner énfasis en otras características, como cual es la organización del plan de estudios, que órganos componen la facultad, que normativas se aplican, que temas se tratan en la carrera, que departamentos hay y cual es su función, que trámites tiene que realizar para matricularse, cual es el proceso de ese trámite, de que equipamientos dispone la facultad, ...

   Podemos formular un conjunto de cuestiones de competencia que queremos que nuestra ontología sea capaz de responder, para el primer dominio de aplicación, por ejemplo:

   - *¿Qué asignaturas son prerrequisito de otra?*
   - *¿Cuál es la carga de laboratorio de la asignatura Criptografía?*
   - *¿De qué asignaturas optativas me puedo matricular después de hacer Inteligencia Artificial?*
   - *¿Que horarios de mañana tiene la asignatura Compiladores?*
   - *¿De qué asignaturas de libre elección se puede matricular un alumno de fase de selección?*
   - *¿Cuantas asignaturas del perfil "Tècniques avançades de programació" me quedan por hacer?*

   A partir de estas preguntas podemos, por ejemplo, ver qué necesitamos definir el concepto asignatura, que tendrá diferentes especializaciones por varios criterios, estas tendrán relaciones de precedencia, estarán asociadas a perfiles, una asignatura tendrá diferentes tipos de carga de trabajo que se medirá en créditos, ...

2. **Considerar la reutilización de ontologías existentes**

   Las ontologías se construyen para comunicar conocimiento en dominios, por lo que se construyen con la idea de compartición y reutilización. Se supone que una ontología establece un vocabulario común, por lo que no es nada extraño que ya alguien haya estudiado el dominio y creado ese vocabulario. Por lo tanto, no es necesario rehacer un trabajo que ya esta hecho, si existe una ontología sobre el dominio en el que trabajamos, podemos incorporarla. En la última sección de este capítulo enumeraremos proyectos de ontologías que pueden ser el punto de partida desde el que podemos desarrollar una ontología para nuestro dominio en particular.

3. **Enumerar los términos importantes en la ontología**

   Escribir una lista de términos que podemos usar para referirnos a nuestro dominio, elaborando frases que podríamos utilizar para preguntarnos cosas sobre él o para explicar a alguien información sobre él. Deberemos pensar en:

   - ¿Qué propiedades tiene esos términos?
   - ¿Qué nos gustaría decir sobre ellos?

   El objetivo de esta fase es tener una visión informal de los conceptos y elementos que necesitaremos tener en cuenta para formalizar el dominio.

   Ejemplo 5.2 *Siguiendo con el ejemplo de la ontología de la facultad, tendríamos que recolectar todos los términos que vamos a usar en la ontología: asignatura, horario, aula, prerrequisito, perfil, crédito, tema, departamento, convocatoria, ...*

4. **Definir las clases y su jerarquía**

   Los conceptos no aparecen desvinculados entre sí y de hecho un conjunto desorganizado de conceptos no es útil, por lo que deberemos descubrir su estructura y sus relaciones de generalización y especialización. Podemos tomar diferentes aproximaciones

   - **De arriba a abajo:** Definimos los conceptos mas generales y vamos especializándolos
   - **De abajo a arriba:** Definimos las clases más específicas y vamos agrupándolas según propiedades comunes, generalizando
   - **Combinación de ambas:** Definimos los conceptos mas importantes y especializamos y generalizamos para completar la ontología

   Ninguno de estos métodos es esencialmente mejor y depende en gran medida del dominio. Existen dominios en los que es fácil descubrir los conceptos generales y se ha de trabajar para obtener las especializaciones más útiles o adecuadas. Hay dominios en los que es más fácil razonar a partir de conceptos específicos y hay que buscar una manera coherente y útil de organizarlos. Muchas veces es la experiencia en la construcción de ontologías la que nos hace decidirnos por una metodología u otra.

   Este paso y el siguiente están estrechamente relacionados y es muy difícil hacer primero uno y luego el otro. Por lo general se realizan iterativamente en varios ciclos.

   Ejemplo 5.3 *Por ejemplo podemos definir una clasificación para el concepto asignatura como el que aparece en la figura 5.1*

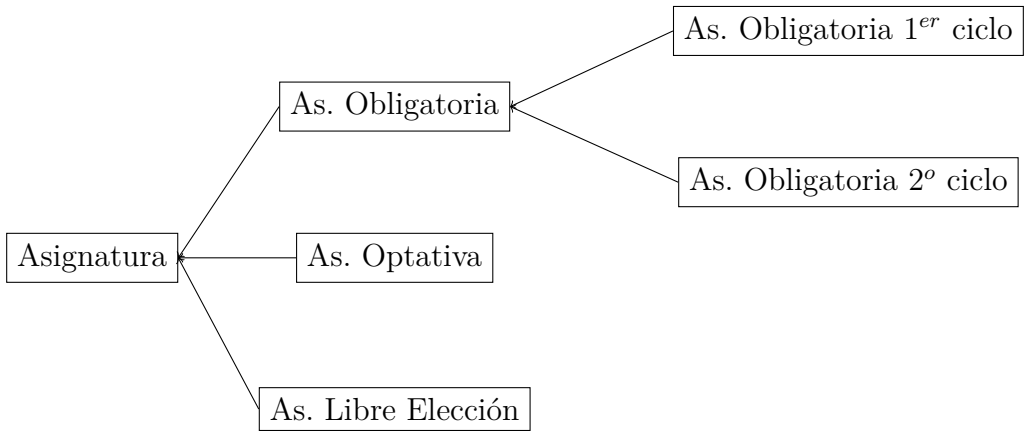


Figura 5.1: Jerarquía del concepto asignatura

5. **Definir las propiedades de las clases**

   Debemos describir la estructura interna de las clases, esta dependerá de la semántica que queremos que tenga. Deberemos determinar una lista de características que describen esa semántica y en que clases concretas debemos poner esas características. La elección de estas propiedades puede depender del dominio de aplicación, restringiéndolas a solo las necesarias o podemos desarrollar la ontología con una visión más amplia para que esta pueda ser utilizada en otros dominios de aplicación.

   En la descripción de los conceptos nos podemos encontrar muchos tipos de propiedades

   - Propiedades descriptivas, cualidades
   - Propiedades identificadoras, nombres
   - Partes u otras relaciones taxonómicas
   - Relaciones con instancias de otras clases

   Desde un punto de vista de la descripción de clases, las relaciones se pueden ver al mismo nivel que las propiedades y de hecho muchos lenguajes de descripción de ontologías no hacen la distinción. Básicamente, podríamos considerar una propiedad a aquel atributo cuyo valor es un tipo predefinido (booleano, numérico, carácter, ...) y una relación sería una propiedad en la que los elementos son de la ontología.

   Junto con la determinación de cuales son las propiedades necesarias, estas deberían asignarse a clases de la jerarquía de conceptos. Lo normal es asignar las propiedades a la clase mas general, dejando que el resto las obtengan vía herencia.

   Ejemplo 5.4 *Siguiendo con el ejemplo, podríamos definir las características y relaciones que representan una asignatura incluyendo: Nombre, Créditos_Totales, Créditos_de_Teoría, ..., con_tema, impartida_por_departamento, perteneciente_al_perfil, ...*

6. **Definir las características de las propiedades**

   Para describir las propiedades deberemos identificar también sus características y restricciones, entre estas podemos tener:

   - Cardinalidad (número de valores permitidos)
   - Tipo, valores

- Valores por defecto
- Obligatoriedad
- Si es una relación hay que definir su cardinalidad y su rango y si tiene inversa.

Ejemplo 5.5 *Podemos definir las características de las propiedades de asignatura, por ejemplo el Nombre es una cadena de caracteres y es un atributo obligatorio, los Créditos_Totales es un número real, impartida_por_departamento sería una relación entre asignatura y el departamento que la imparte con cardinalidad 1 y con una relación inversa con cardinalidad N, ...*

7. **Crear instancias**

   La ontología supone un lenguaje de definición que utilizaremos para hablar de elementos concretos. En este caso los elementos concretos son las instancias. Es posible que una ontología tenga como parte de su definición un conjunto de instancias que aparecen en cualquier uso que podamos hacer de ella. En este caso este sería el momento de decidir cuáles son esos individuos y crearlos a partir de las definiciones que hemos determinado.

   En el momento de usar la ontología tendremos que crear otras instancias específicas para el problema que vayamos a resolver. Para ello utilizaremos los términos de la ontología que hemos desarrollado para crear una representación del problema. Obviamente, la ontología nos impone los límites de lo que podremos representar, pero si hemos desarrollado correctamente la ontología tendremos el vocabulario necesario.

Consejos prácticos

Estos son unos consejos prácticos a tener en cuenta a la hora de desarrollar una ontología:

- No incluir versiones singulares y plurales de un término (la mejor política es usar solamente nombres en singular o plural). Estamos creando una terminología, por lo que es de sentido común usar un criterio uniforme a la hora de dar nombres a los conceptos que utilizaremos. Es posible que queramos representar individuos y agrupaciones de individuos en nuestro problema, tenemos que tener en cuenta que semánticamente son cosas diferentes y no deberíamos usar las mismas palabras.

- Los nombres no son las clases, debemos distinguir la clase del nombre que le damos. Podemos tener sinónimos, pero todos representan a la misma clase. Es a veces sencillo confundir el nombre de una cosa con la cosa en sí. Estamos estableciendo los conceptos que existen en nuestro dominio, los nombres no son entidades independientes.

- Asegurarnos de que la jerarquía está correctamente construida. Una jerarquía incorrecta afecta a nuestra capacidad para deducir a partir de la ontología y obtener respuestas correctas a nuestras preguntas. Hemos de pensar que la forma principal de razonamiento en esta representación es la herencia.

- Observar las relaciones de transitividad y comprobar si son correctas. La transitividad es un mecanismo importante de deducción y además ahorra explicitar en la ontología muchas relaciones que se pueden deducir vía este mecanismo, establecer relaciones transitivas que no lo son solo puede dar problemas. También es una relación que puede hacer muy ineficiente el razonamiento y puede dar lugar fácilmente a una explosión combinatoria, no conviene utilizarla más de lo necesario.

- Evitar ciclos en la jerarquía. Obviamente es algo difícil de hacer en una ontología sencilla, pero si el número de clases es grande y permitimos herencia múltiple es algo que hay que vigilar.
- Todas las subclases de una clase deben estar al mismo nivel de generalidad. Cada especialización debe llevar a conceptos que tengan el mismo nivel de descripción. Mezclar conceptos generales y específicos en un mismo nivel hace la ontología confusa y difícil de interpretar. También puede llevar a deducciones incoherentes.
- No hay un criterio respecto al número de clases que debe tener un nivel de la jerarquía, la experiencia dice que un número entre dos y doce es habitual, más clases indicaría que tenemos que estructurarlas añadiendo más niveles
- ¿Cuando introducir nuevas clases? Suele ser incómodo navegar por jerarquías o muy planas o muy profundas, se debería elegir un punto intermedio, unas indicaciones serían:
  - Las nuevas clases tienen propiedades adicionales que no tiene la superclase
  - Tienen restricciones diferentes
  - Participan en relaciones diferentes

  No obstante nos puede interesar crear clases porque existen en el dominio aunque no tengan atributos o relaciones distintas, o porque hacen más clara la comprensión de la ontología.
- Decidir si hemos de usar una propiedad o crear una clase. A veces un atributo es suficientemente importante como para considerar que sus valores diferentes corresponden a objetos diferentes.
- Decidir donde está el nivel de las instancias. Pensar cual es nivel mínimo de granularidad que necesitamos.
- Limitar el ámbito de la ontología a las necesidades de representación.
  - La ontología no necesita incluir todas las clases posibles del dominio, solo las necesarias para la aplicación que se va a desarrollar.
  - Tampoco necesitamos incluir todos los atributos/restricciones/relaciones posibles.

5.4 Proyectos de ontologías

Como se comentó al principio del capítulo, una ontología se crea con el propósito de que sea compartida y sirva como un lenguaje común para que diferentes agentes puedan intercambiar conocimiento. Esto ha hecho que se hayan desarrollado múltiples proyectos de investigación encaminados a escribir ontologías tanto en dominios específicos como generales.

Uno de los proyectos de ontologías mas ambiciosos es **CYC**. Este proyecto comenzó en los años 1980 y su pretensión es crear una base de conocimiento que describa todo el conocimiento de sentido común que puede ser necesario para escribir una aplicación de inteligencia artificial. Este proyecto ha dado lugar a una ontología de dominio público OpenCYC que contiene aproximadamente cientos de miles de conceptos y millones de aserciones sobre ellos. El lenguaje en el que esta escrita esta ontología es CYCL que esta basado en la lógica de predicados. Esta ontología se puede utilizar tal cual para escribir aplicaciones o se pueden coger subconjuntos con conceptos para dominios específicos (microteorías). Se puede ver la estructura de conceptos más generales en la figura 5.2.

Como comentamos al principio de esta capítulo, el diccionario de un idioma puede verse como una ontología. Esto ha llevado al desarrollo de múltiples ontología especializadas en el dominio del

tratamiento del lenguaje natural. Un ejemplo de este tipo de ontologías es **Wordnet** que es una ontología léxica para el idioma inglés. Esta ontología está organizada según categorías semánticas y etiquetado con las categorías sintácticas correspondientes a cada palabra. En la actualidad contiene 150.000 palabras inglesas organizadas en 115.000 sentidos. La ontología está estructurada jerárquicamente mediante la relación de hiperonimia/hiponimia. esta ontología esta pensada para aplicaciones de lenguaje natural, pero también se pueden utilizar los conceptos y la estructuración para otras aplicaciones. Se puede ver la estructura de conceptos más generales en la figura 5.3. En la actualidad existen versiones de esta ontología para muchos idiomas y se utiliza ampliamente en muchas aplicaciones que involucran la comprensión del lenguaje natural.

Dentro de los proyectos de ontologías existen algunos que intentan crear una ontología de los conceptos mas generales que se deberían utilizar en la construcción de ontologías. El objetivo de estas ontologías no es pues recolectar todos los conceptos de un dominio, sino dar los conceptos bajo los cuales estos deberían estar organizados. Este tipo de ontologías se denominan *Upper Model* y no existe de momento ninguna ontología de este tipo que sea ampliamente aceptada. De hecho desde el punto de vista teórico hay argumentos a favor y en contra de que exista o se pueda construir una ontología de este tipo.

Dentro de este tipo de proyectos cae **Generalized Upper Model**. Es una ontología léxica pensada para tratamiento del lenguaje natural que posee alrededor de 250 conceptos. Se puede ver la estructura de conceptos en la figura 5.4.

También existen multitud de ontologías para dominios específicos. Por ejemplo, en comercio electrónico existen ontologías estándar definidas por organismos internacionales para la clasificación y nomenclatura de productos y servicios que se utilizan para el intercambio de información. Otro dominio con múltiples ontologías es la medicina, área que destaca por aplicaciones donde hay grandes necesidades de almacenamiento e intercambio de información, y que necesita nomenclaturas muy específicas. Otras áreas con ontologías muy utilizadas incluyen la ingeniería, el mundo empresarial o la química.

El interés por construir ontologías se ha visto impulsado en la actualidad por el proyecto **Semantic Web**. La ambición de este proyecto es el desarrollo de un lenguaje de ontologías que permita describir el contenido de las páginas de web para que puedan ser procesadas de manera automática. Actualmente el contenido de la web está pensado para ser accedido por personas, el contenido está poco estructurado, está en lenguaje natural y el lenguaje de representación está mas pensado para el formato del contenido que para su descripción.

Los lenguajes de descripción de contenido para la web se construyen a partir del lenguaje XML. El primero de ellos fue `RDF/RDFS` (Resource Description Format/Resource Description Format Schema) que permite definir categorías y relaciones aunque está bastante limitado como lenguaje de representación. Sobre este lenguaje se construyeron `DAML` (Darpa Agent Markup Language) y `OIL` (Ontology Inference Layer), resultado de proyectos sobre web semántica en Estados Unidos y Europa respectivamente.

Estos lenguajes definen las características necesarias para tener un lenguaje de descripción de ontologías y, de hecho, se pueden ver como ontologías de dominio que definen el vocabulario necesario para representar conocimiento. Estos dos lenguajes se fusionaron y el lenguaje resultante se llama `OWL` (Ontology Web Language) que es un estándar del W3C. Este lenguaje se basa en lógica de descripción y tiene diferentes versiones: OWL lite, OWL DL y OWL full. La primera es la menos expresiva, pero garantiza que se puede razonar sobre el en tiempo finito. La segunda se permite representar expresiones en lógica de descripción sin limitaciones, pero no asegura poder resolver cualquier expresión en tiempo finito. Para la última versión no hay implementaciones de razonadores que puedan tratarla.

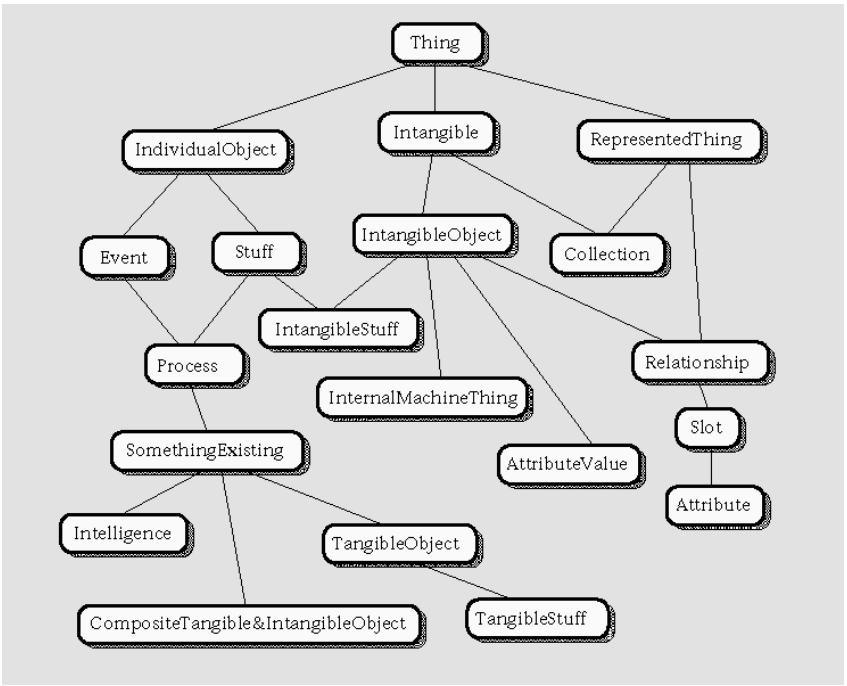


Figura 5.2: Ontología de CYC

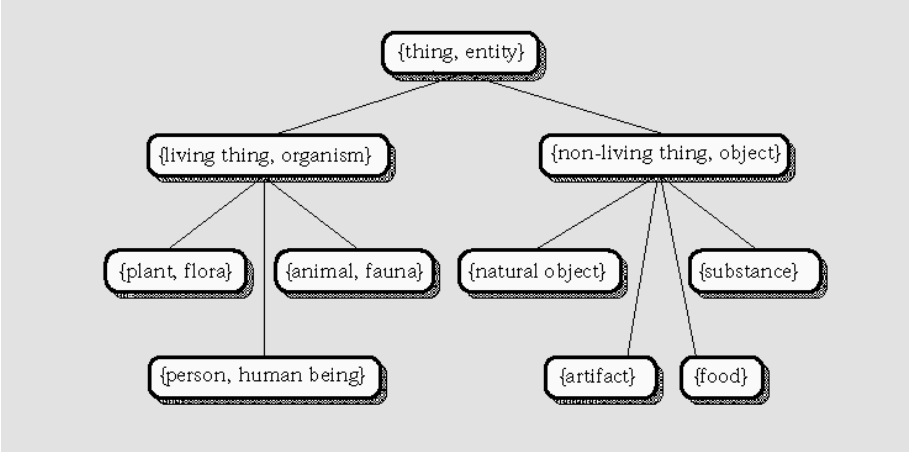


Figura 5.3: Ontología de Wordnet

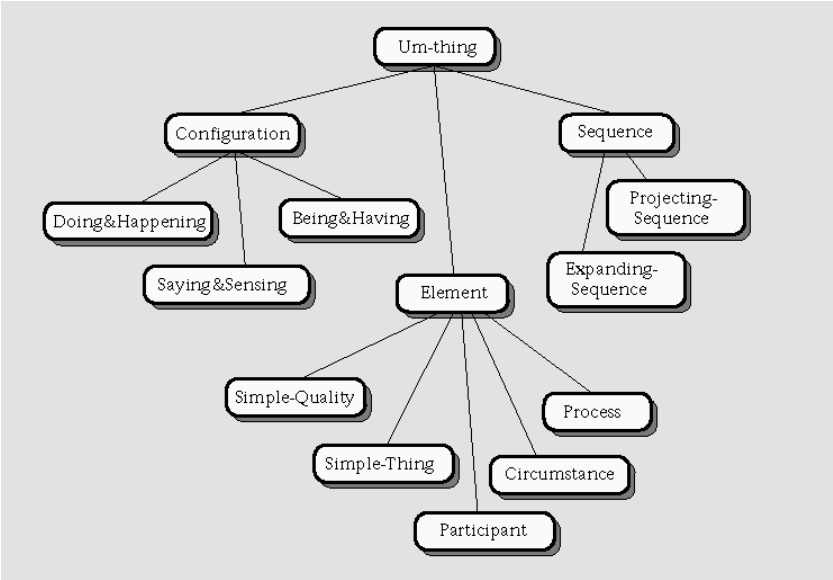


Figura 5.4: Ontología de Generalized Upper Model

Parte II

Sistemas basados en el conocimiento

6. Introducción a los SBC

6.1 Introducción

Un tema esencial en el área de Inteligencia Artificial es la resolución de problemas. Ya hemos visto en temas anteriores de la asignatura métodos generales que a partir de una representación del problema basada por ejemplo en el espacio de estados o en el de soluciones y un conjunto de operadores específicos, se exploraba este espacio en busca de una solución al problema.

Este tipo de métodos no utilizaban apenas conocimiento del dominio del problema que intentaban solucionar y se guiaban por funciones heurísticas generales que se basaban en unas pocas características del problema. Estas funciones heurísticas intentaban guiar la elección del camino solución ordenando los pasos accesibles durante la exploración. La capacidad para generar el orden adecuado en la exploración es lo que permitía reducir en la práctica su coste computacional.

Evidentemente, las decisiones que se pueden tomar a partir de los valores de una función heurística a veces no son suficientemente buenas para reducir sensiblemente el espacio de búsqueda o, simplemente, es imposible encontrar una función heurística adecuada.

Este tipo de métodos generales se utilizaron como métodos básicos de resolución de problemas en los primeros años de la Inteligencia Artificial ya que permitían su aplicación de forma relativamente sencilla a cualquier dominio. Se pudo comprobar que el coste computacional de este tipo de métodos era prohibitivo en muchos problemas reales y que hacían falta otras metodologías que mejoraran su eficiencia. El elemento a introducir para conseguir esta eficiencia es el conocimiento específico del dominio del problema a resolver. Este conocimiento puede acelerar el proceso de resolución al mejorar las decisiones que se toman.

Los métodos generales son denominados *métodos débiles* frente a los que denominaremos *métodos fuertes*, que explotarán el conocimiento del dominio. El uso de conocimiento particular hará que estos métodos necesiten más trabajo de desarrollo a la hora de aplicarlos y que la experiencia de usarlos en un problema no sea exportable directamente a otros problemas, ya que estará orientada al problema concreto que se resuelve.

El principal exponente de estos métodos que se basan en conocimiento específico del dominio son los que se denominaron en su origen *sistemas expertos* (*expert systems*) y que actualmente son conocidos también como *sistemas basados en el conocimiento* (*knowledge based systems*).

Originalmente, el término sistema experto intentaba reflejar que el objetivo que se tenía al construir este tipo de sistemas era emular la forma de resolver problemas de un experto humano en un área específica de conocimiento.

Todavía se siguen utilizando de manera indistinta las dos denominaciones, pero el término sistemas expertos se suele asociar más a sistemas construidos a partir de conocimiento de expertos humanos, basados fundamentalmente en sistemas de reglas de producción y suelen ser sistemas cerrados donde, en el dominio del problema, el aprendizaje y la adaptación no son prioritarios. El término sistemas basados en el conocimiento pretende ser mas general, incluyendo cualquier sistema en el que se utilice conocimiento independientemente de su procedencia (procesos de ingeniería del conocimiento, aprendizaje automático) y la metodología o arquitectura de diseño que se utilice (razonamiento basado en reglas, razonamiento basado en casos, modelos cualitativos, agentes inteligentes, redes neuronales, ...), las aplicaciones pueden incluir la necesidad de adaptación y aprendizaje.

6.2 Características de los SBC

Los sistemas basados en el conocimiento están pensados para la resolución de problemas complejos en los que las técnicas de software convencionales no son aplicables.

Dado el tipo de problemas con los que se tendrán que enfrentar, hay un conjunto características que se considera que un sistema de este tipo debería tener y que le diferencian de los sistemas de software tradicionales. Estas características son:

- Flexibilidad, ya que los tipo de problemas que se pueden presentar en un dominio pueden ser bastante variados
- Emular un comportamiento racional, ya que uno de los objetivos es utilizar los mecanismos de resolución que usan los expertos humanos tanto para elaborar la solución como para explicar sus razones.
- Operar en un entorno rico y con mucha información, ya que este tipo de problemas involucra grandes cantidades de conocimiento, con naturaleza bastante diversa y con características especiales, como la incertidumbre o la incompletitud.
- Uso de información simbólica, ya que las decisiones que toman los expertos humanos se basan principalmente en apreciaciones sobre la información descritas a partir de símbolos y no información numérica sin interpretar.
- Uso del lenguaje natural, ya que la forma de interactuar y de recibir respuesta del sistema debería ser lo mas cercana al experto que está emulando.
- Capacidad de aprender, ya que puede ser necesario incorporar nuevo conocimiento para extender las capacidades del sistema a partir de la observación y la experiencia.

Para obtener sistemas que tengan las características mencionadas se ha de tener en cuenta una serie de restricciones y características a la hora de diseñarlos, de manera que deberemos:

1. **Separar el conocimiento y el control de la resolución**

   Ésta es una característica inspirada en la forma en la que los expertos humanos trabajan y es importante sobre todo por la gran cantidad de conocimiento que interviene en la resolución de un problema. Un experto tiene, por un lado, conocimiento específico sobre los elementos que forman parte de su dominio y, por otro, conocimiento sobre las estrategias de resolución de problemas válidas en el dominio que le permiten obtener soluciones a problemas.

   Esta separación además incluye un conjunto de ventajas:

   - Naturalidad para expresar el conocimiento, ya que su representación es independiente de los métodos de resolución.
   - Modularización del conocimiento, ya que se puede agrupar por afinidad, permitiendo un desarrollo más sencillo y comprensible.
   - Extensibilidad, ya que es más sencillo ampliar el conocimiento del sistema si podemos estructurarlo de manera independiente.
   - Modificabilidad, ya que es más fácil detectar errores y modificar el conocimiento, permitiendo probar cada módulo de manera independiente del mecanismo de resolución.
   - Independencia, ya que es posible utilizar diferentes estrategias de resolución para un mismo conocimiento.

2. **Incorporar conocimiento heurístico**

   El conocimiento que se introduce en un SBC está basado en el conocimiento de un experto. Este conocimiento es de naturaleza heurística (incompleto, aproximado, no sistemático) en contraste con el conocimiento algorítmico, en el que tenemos a priori determinado el curso de la resolución y el resultado esperado. Esta resolución basada en el conocimiento experto hace que se parezca más a como los expertos humanos resuelven problemas. Ésta es también una diferencia fundamental respecto a los sistemas software convencionales en los que no existe esta componente heurística.

3. **Permitir Interactividad a diferentes niveles**

   Este tipo de sistemas necesitarán interactuar tanco con los expertos humanos, como con otros sistemas basados en el conocimiento o como con su entorno. Esta interacción ayudará en el curso de la resolución, obteniendo la información que el sistema encuentre que es necesaria para continuar. A la vez, también deberá ser capaz de explicar su línea de razonamiento y revelar el porqué de las elecciones que realiza durante la resolución. Esto último juega un doble papel, primero el justificar las decisiones tomadas (permite dar confianza en los resultados) y segundo permitir el detectar problemas en el conocimiento que tiene el sistema.

Todas estas características hacen de los sistemas basados en el conocimiento un paradigma de la combinación de las diferentes áreas de la Inteligencia Artificial. En estos sistemas necesitaremos:

- **Representación del conocimiento:** Este conocimiento incluirá tanto la descripción de las características del dominio, como el conocimiento de resolución de problemas, la representación del control de la resolución, las heurísticas, ...
- **Razonamiento e inferencia:** La combinación de los datos del problema para llegar a la resolución se realizará como un proceso de razonamiento. Este razonamiento no solo se basará en la lógica clásica, sino que tendrá que utilizar otro tipo de lógicas que permitan, por ejemplo, el razonamiento bajo premisas incompletas, razonamiento ante incertidumbre en el conocimiento y en los pasos de resolución, razonamiento temporal, sobre otros, ...
- **Búsqueda y resolución de problemas:** Las técnicas de resolución basadas en razonamiento tendrán que complementarse con técnicas de búsqueda heurística.
- **Interacción con el usuario:** Mediante lenguaje natural para preguntar, generar explicaciones.
- **Aprendizaje:** Para adquirir el conocimiento del dominio de forma automática, adquisición de nuevo conocimiento o capacidades, aprendizaje de los errores, ...

6.3 Necesidad de los SBC

La principal razón del desarrollo de sistemas basados en el conocimiento viene directamente del interés en automatizar la resolución de problemas que requieren un conocimiento especializado. Este interés viene del alto coste que supone acceder a personal experto ya que por lo general es escaso y su formación es cara. Detallando las razones, podemos decir que la necesidad proviene de:

- Poder disponer del conocimiento de expertos altamente cualificados. Los expertos son caros y no siempre los hay disponibles. La formación de nuevos expertos solo sirve a largo plazo. Es vital disponer de un sistema que posea el conocimiento del experto y permita usarlo.

- Poder ayudar a otros expertos/no expertos. A veces es suficiente con tener personas que puedan utilizar estos sistemas como soporte a sus decisiones o estos sistemas pueden servir para entrenar y completar la formación de otros expertos.
- Poder preservar el conocimiento de los expertos. El poder preservar de alguna manera el conocimiento que han adquirido los expertos es importante. Por un lado por el conocimiento en si, ya que puede seguir utilizándose para tomar decisiones, por otro, por que ese conocimiento ayudará a otros expertos en su formación.
- Poder combinar el conocimiento de varios expertos. En la construcción de estos sistemas se combina el conocimiento de diferentes fuentes de información, entre ellas, grupos de expertos. De esta manera el sistema tiene una visión más amplia y se beneficia de la combinación de conocimientos.
- Poder disponer de sistemas que permitan dar soluciones rápidas y justificadas. La automatización del proceso de resolución permite que la respuesta a los problemas pueda ser más rápida. El proceso de razonamiento se puede estudiar y justificar de una manera más fiable.
- Poder tratar grandes volúmenes de información. En muchas áreas el volumen de información involucrada en los problemas a solucionar supera la capacidad de los expertos, de manera que la automatización del proceso es necesaria para su resolución efectiva.
- Poder disponer de sistemas que tomen decisiones autónomas. Ciertos problemas no necesitan constantemente una supervisión por parte de expertos y algunas decisiones se pueden dejar al sistema para reducir su dependencia.

6.4 Problemas que se pueden resolver mediante SBC

No cualquier problema es susceptible de ser resuelto mediante un SBC, antes de comenzar a trabajar nos hemos de plantear ciertas preguntas:

- ¿La necesidad de la solución justifica el desarrollo? El desarrollo de un SBC tiene un coste importante tanto económicamente, como en tiempo. Se ha de adquirir el conocimiento del experto, se ha de construir el sistema, se ha de validar. El problema ha de tener entidad suficiente.
- ¿Es posible acceder al experto necesario? Si es difícil acceder a un experto cuando es necesario, siempre es mas sencillo tener disponible un sistema capaz de ayudar a otras personas no tan expertas.
- ¿El problema puede plantearse como un proceso de razonamiento simbólico? Los SBC aplicarán métodos de razonamiento simbólico para solucionar el problema, si no puede plantearse en esos términos no tiene sentido desarrollar un SBC para el problema.
- ¿El problema está bien estructurado? Para que sea posible tratar el problema hemos de poder identificar sus elementos y hemos de poder formalizar tanto el conocimiento necesario, como los procedimientos de resolución. De hecho, el proceso de construcción del SBC requerirá una formalización de todos los elementos del problema.
- ¿Puede resolverse el problema por métodos tradicionales? Si el problema tiene una solución algorítmica o matemática y no involucra ningún proceso de razonamiento cualitativo o heurístico es innecesario desarrollar un SBC para resolverlo.

- ¿Existen expertos disponibles y cooperativos? El construir este tipo de sistemas necesita la colaboración completa de expertos en el problema. Estos han de comprender la tarea y el objetivo del SBC y cooperar en la labor de formalización del conocimiento. Al ser el conocimiento el elemento fundamental del sistema, la dificultad de acceder a él hace imposible la labor de desarrollo.

- ¿Está el problema bien dimensionado? Esta restricción es clave en cualquier proyecto informático. La complejidad de la tarea no ha de ser tal que se necesiten formalizar grandes cantidades conocimiento para resolverlo, ha de ser suficientemente restringido para ser manejable y poderse terminar la tarea con éxito.

6.5 Problemas de los SBC

Existen todo un conjunto de problemas asociados a los SBC, algunos son inherentes a sus características, otros pueden depender de las limitaciones que imponen el coste de su desarrollo. Podemos enumerar los siguientes:

Fragilidad: Su capacidad se degrada bruscamente cuando los problemas que se le plantean están fuera de su ámbito de experiencia. En el caso de un experto humano, éste podría dar respuestas mas o menos ajustadas a pesar de que los problemas vayan saliendo de su ámbito de conocimiento, estos sistemas simplemente serán incapaces de dar una respuesta.

Dificultades con el control del razonamiento: Los expertos, además del conocimiento del dominio, tienen conocimiento sobre como plantear y estructurar la resolución de un problema. Este conocimiento, denominado *metaconocimiento* es difícil de explicitar y suele ir mezclado con el conocimiento del dominio. De la posibilidad de codificar este conocimiento en el sistema dependerá la eficiencia a la hora de encontrar soluciones.

Baja reusabilidad de las bases de conocimiento: Por lo general el conocimiento de un problema depende fuertemente del dominio, sobre todo la parte de resolución. No obstante ciertas partes del conocimiento mas general puede ser reaprovechado como ontologías de dominio.

Es difícil integrar el aprendizaje: Muchas veces estos sistemas no incluyen capacidad de aprendizaje por la complejidad de integrar nuevo conocimiento con el ya existente en el sistema. No obstante hay tareas en las que la capacidad de aprendizaje es esencial para la resolución.

Problemas en la adquisición del conocimiento: Obtener tanto la descripción del dominio, como el conocimiento de resolución es el principal cuello de botella de los SBC. Esto es debido a la dificultad que tienen los expertos en explicitar su conocimiento. Se ha denominado *paradoja del experto* a la observación de que, cuanta más experiencia tiene una persona en un problema, más difícil le es explicar como lo resuelve.

Problemática de la validación: El proceso de validación de una base de conocimiento es muy costoso. El mayor problema reside en la posibilidad de que haya inconsistencias en el sistema[1]. También es muy costoso probar la completitud del sistema.

[1]Probar que un conjunto de axiomas es inconsistente es semidecidible.

6.6 Áreas de aplicación de los SBC

Los sistemas basados en el conocimiento se aplican a gran variedad de áreas. Se pueden encontrar en cualquier dominio en el que se necesite un conocimiento especializado. Áreas tan diferentes como la medicina, la biología, el diseño, la concesión de créditos o la predicción meteorológica se cuentan entre sus aplicaciones.

Su sofisticación depende del tipo de problema concreto que se desee resolver, pero básicamente se aplican en dos tareas genéricas, el análisis de un problema para identificar su naturaleza o la construcción de una solución a partir de sus elementos esenciales.

Problemas que involucran identificación pueden ser por ejemplo la interpretación de los elementos de un problema, el diagnóstico a partir de unos síntomas, la supervisión o control de un proceso para mantenerlo dentro de los parámetros correctos o la predicción de resultados a partir de evidencias.

Problemas que involucran la construcción de una solución pueden ser por ejemplo la planificación de un conjunto de tareas, el diseño a partir de un conjunto de elementos y restricciones o la configuración de un conjunto de elementos.

6.7 Breve historia de los SBC

La historia de los SBC comienza con la construcción de los primeros sistemas expertos desarrollados durante la década de 1960. Estos sistemas intentaban resolver tareas medianamente complejas dentro de un ámbito bastante específico. En esa época los diferentes lenguajes y entornos para la construcción de estos sistemas estaban poco desarrollados, por lo que estos sistemas eran esfuerzos bastante individuales y muchos de ellos tenían la intención de probar la viabilidad de la construcción de sistemas capaces de resolver problemas complejos.

El primer sistema experto fue DENDRAL, desarrollado en 1965. El problema que se quería resolver era el de la identificación de la estructura de moléculas orgánicas a través de espectrografía de masas. En esa época este tipo de problema necesitaba a una persona con bastante experiencia y llevaba un tiempo considerable. Este sistema fue capaz de reducir drásticamente el tiempo necesario para resolver el problema permitiendo su automatización. Se puede considerar que la mayoría de los sistemas que se construyeron en la siguiente década derivan de este trabajo.

Otro hito en la historia de los sistemas expertos fue MYCIN (1970). Se encuadraba en el área de la medicina y era capaz de identificar infecciones en sangre. Su principal aportación fue la introducción de tratamiento del conocimiento incierto mediante el sistema denominado de factores de certeza. De este trabajo surgió EMYCIN que es considerado el primer entorno de desarrollo de sistemas expertos.

El sistema PROSPECTOR (1974) trataba el dominio de las prospecciones minerales. Su mérito consiste en haber introducido un nuevo método para el tratamiento de la incertidumbre[2] que es el origen de algunos de los métodos actuales.

El primer sistema experto de un tamaño respetable que tuvo éxito comercial fue XCON (1980), con alrededor de 2500 reglas de producción. El sistema asistía a la compra de sistemas de computación VAX (de la desaparecida compañía Digital Equipment Corporation[3]) configurándolos según las necesidades del cliente. Este sistema evitaba errores en los envíos de los sistemas asegurándose de que todos los elementos requeridos fueran enviados reduciendo las posibilidades de error, lo cual ahorró una buena cantidad de dinero a la compañía.

La década de 1980 marca el inicio de la carrera de la construcción de sistemas expertos en prácticamente cualquier dominio de aplicación. Su capacidad para resolver problemas tanto de diagnóstico,

[2]Dice la leyenda que el sistema superó a los expertos prediciendo vetas de mineral no descubiertas por los expertos.

[3]Digital Equipment Corporation fue una de las empresas pioneras en la fabricación de ordenadores, fue absorbida por Compaq que posteriormente se fusionó con HP.

como de clasificación, monitorización, supervisión, predicción, diseño, ... han hecho que su construcción se cuente por miles.

Durante la década de 1990 estos sistemas se han ido diversificando también en las metodologías aplicadas para su construcción pasando de ser construidos básicamente mediante sistemas de producción a utilizar técnicas provenientes de la inteligencia artificial distribuida como los *agentes inteligentes*, usar técnicas de *razonamiento basado en casos* (Case Based Reasoning) o técnicas de aprendizaje automático como las *redes neuronales*. Esta diversificación ha generalizado su concepción y se ha rebautizado a estos sistemas como sistemas basados en el conocimiento (Knowledge Based Systems).

También se han desarrollado nuevas técnicas para el tratamiento de la incertidumbre que se han incorporado en estos sistemas como por ejemplo las redes bayesianas y el razonamiento difuso. El aprendizaje está empezando a tener gran importancia como por ejemplo en la fase de adquisición del conocimiento, de manera que se pueden construir sistemas que resuelvan problemas a partir de ejemplos presentados por los expertos y el sistema puede construir su programación a partir de ellos.

En la actualidad los sistemas basados en el conocimiento desempeñan multitud de tareas tanto especializadas, como cotidianas.

7. Arquitectura de los SBC

7.1 Introducción

En este capítulo vamos a describir los diferentes elementos de los que se compone la arquitectura de un sistema basado en el conocimiento. Esta arquitectura puede variar dependiendo de las técnicas de inteligencia artificial que se utilicen, pero existen un conjunto de bloques generales que aparecerán siempre. La arquitectura de estos sistemas está pensada para poder cumplir las capacidades que se esperan de un sistema basado en el conocimiento, principalmente:

- Que sea capaz de resolver problemas a partir de información simbólica
- Que aplique métodos de razonamiento y resolución heurísticos
- Que sea capaz de explicar los resultados y el proceso de razonamiento
- Que sea capaz de interactuar con el entorno y pueda ser interrogado

Para cumplir estas características necesitamos al menos los siguientes elementos en un SBC:

- Un subsistema de *almacenamiento del conocimiento*
- Un subsistema de *uso e interpretación del conocimiento*
- Un subsistema de *almacenamiento del estado del problema*
- Un subsistema de *justificación e inspección de las soluciones*
- Un interfaz de *comunicación con el entorno y/o el usuario*

Adicionalmente se puede incorporar capacidad de aprendizaje mediante observación del entorno, que daría lugar a la necesidad de un subsistema de *aprendizaje*.

En la figura 7.1 se puede ver una representación de sus relaciones.

Los tres primeros subsistemas se pueden agrupar en el *subsistema de razonamiento*, que será el encargado de construir la solución del problema.

El conocimiento y la forma en la que funcionan estos subsistemas dependen bastante de las técnicas que se usen para la representación del conocimiento y la resolución de problemas. Nosotros nos centraremos en dos tipos:

- Sistemas basados en reglas de producción (Rule Based Reasoning)
- Sistemas basados en casos (Case Based Reasoning)

7.2 Arquitectura de los sistemas basados en reglas

Los sistemas basados en reglas utilizan como elemento principal a los sistemas de reglas de producción. La resolución de un problema se realiza mediante el proceso de razonamiento generado por el ciclo de funcionamiento de un motor de inferencia. El conocimiento del dominio está expresado como una ontología de dominio y el conocimiento de resolución está representado como un conjunto de reglas de producción.

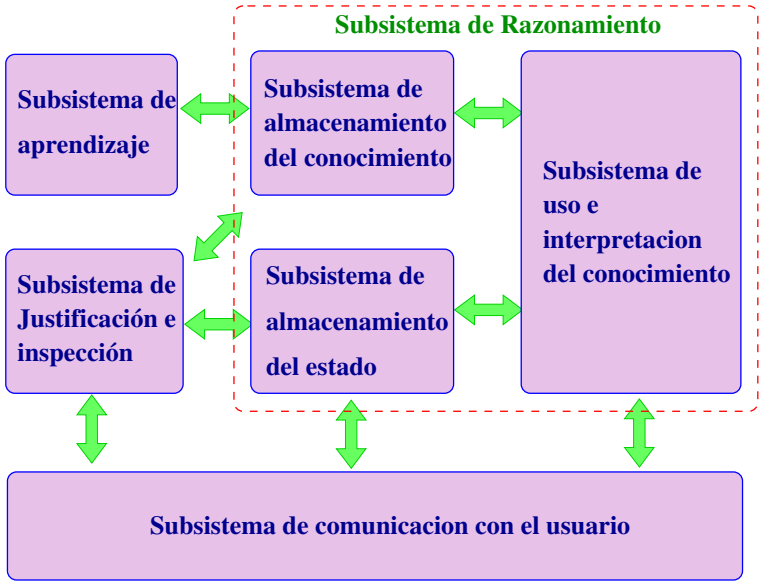


Figura 7.1: Arquitectura genérica de un SBC

7.2.1 Almacenamiento del conocimiento

Este subsistema contiene el conocimiento necesario para resolver problemas en el dominio del SBC y se habrá construido a partir de un proceso previo de extracción de conocimiento. También se le conoce como *base de conocimiento*.

El conocimiento almacenado es de naturalezas diferentes y puede por lo tanto estar representado utilizando distintos formalismos. Centrados en los sistemas basados en reglas de producción tendremos tres tipos de conocimiento en una base de conocimiento:

1. Conocimiento factual (objetos del dominio y sus características)
2. Conocimiento relacional (relaciones entre los objetos del dominio, por ejemplo jerárquicas)
3. Conocimiento condicional (reglas de producción que expresan conocimiento deductivo sobre el dominio)

Los dos primeros conocimientos suelen estar integrados en una ontología de dominio y habitualmente están representados mediante formalismos estructurados como frames o redes semánticas.

El tercer conocimiento es el más voluminoso y mas importante desde el punto de vista de la resolución de problemas. Está representado habitualmente mediante un formalismo de reglas de producción, aunque existen otros formalismos. La complejidad y expresividad de estas reglas dependerá del dominio, ya que puede ser necesario trabajar con formalismos lógicos que traten sus características particulares. Puede ser suficiente con un formalismo basado en lógica de primer orden, necesitar hacer tratamiento de la incertidumbre, trabajar con conocimiento incompleto, razonamiento temporal, ...

La expresividad del formalismo de reglas de producción nos permitirá representar conocimiento de diferente naturaleza que puede ser necesario para un dominio, principalmente:

- **Conocimiento deductivo (estructural):** Este conocimiento nos permitirá describir los procesos de resolución de problemas del dominio a partir de cadenas de deducción.
- **Conocimiento sobre objetivos (estratégico):** Este conocimiento orientará la resolución del problema y permitirá plantear la forma de resolver problemas a alto nivel.

- **Conocimiento causal (de soporte):** Este conocimiento permite añadir la posibilidad de realizar explicaciones sobre los procesos de deducción llevados a cabo para solucionar un problema o interrogar al sistema sobre la consecuencia de suposiciones (what if)

Habitualmente los lenguajes de reglas que se utilizan para implementar este conocimiento permiten organizarlo de manera estructurada para facilitar el desarrollo. Entre las facilidades mas comunes se encuentra la división del conocimiento en **módulos**. Estos permiten agrupar las reglas en bloques de alto nivel que tengan sentido en el dominio del problema.

Esta posibilidad de estructurar las reglas permite una mejor organización y encapsulamiento del conocimiento, facilitando el desarrollo, prueba y depuración.

Precisamente esta organización en módulos de las reglas permite la expresión del conocimiento sobre objetivos ya mencionado. Este conocimiento se describe mediante reglas denominadas **meta-reglas**. Estas reglas describen conocimiento de alto nivel que no se refiere a problemas específicos del dominio, sino a estrategias o metodologías de resolución de problemas propias del dominio. Mediante las meta-reglas se pueden desarrollar mecanismos de resolución mas potentes de lo que permiten las reglas que expresan conocimiento deductivo y reducir el coste computacional de la resolución de problemas.

El mayor problema de las meta-reglas es que son mas difíciles de adquirir a partir del experto ya que representan conocimiento de un nivel mayor de abstracción. Las meta-reglas se pueden clasificar dependiendo de su nivel de intervención en el proceso de resolución:

Meta-reglas sobre reglas: Se usan para activar o desactivar reglas dentro de un módulo o dar prioridad a las reglas que se activan

Metarreglas sobre módulos: Afectan a la forma en la que se hace la búsqueda dentro del módulo, a la forma de tratamiento de la incertidumbre, al tipo de conclusiones, ...

Metarreglas sobre estrategias: Deciden sobre la ordenación de activación de los módulos de reglas, tratamiento de excepciones, ...

Metarreglas sobre planes de actuación: Cambian las diferentes estrategias de resolución de problemas de alto nivel que pueda haber

7.2.2 Uso e interpretación del conocimiento

Si la implementación del SBC se realiza utilizando reglas de producción este subsistema es habitualmente un *motor de inferencia*. Ya conocemos el funcionamiento de un motor de inferencia del tema de sistemas de reglas de producción.

Este subsistema utilizará la base de conocimiento y la información del problema particular a resolver, para obtener nuevas inferencias que lleven a plantear los pasos necesarios para llegar a la solución.

La complejidad del motor de inferencia dependerá de la expresividad de las reglas de la base de conocimiento. Este tendrá que interpretar las condiciones de las reglas, realizar las instanciaciones adecuadas y escoger la regla a aplicar en cada paso de la resolución. Esta elección dependerá directamente de la estrategia de resolución de conflictos que tenga implementada o del conocimiento de control expresado mediante las meta-reglas.

Al ser el motor de inferencia el mecanismo básico de resolución del sistema la capacidad y eficiencia en la resolución de problemas dependerá totalmente de él.

7.2.3 Almacenamiento del estado del problema

A este subsistema se le conoce como *memoria de trabajo*. Su complejidad dependerá de la expresividad del mecanismo de resolución. Habitualmente almacena los datos iniciales del problema descritos mediante los objetos de la ontología del dominio y las deducciones intermedias que se van obteniendo durante el proceso de resolución.

En sistemas mas complejos la memoria de trabajo puede almacenar otras cosas, como información de control que apoye al mecanismo de resolución (orden de deducción de los hechos, preferencias sobre hechos, reglas, módulos, reglas activadas recientemente, caminos alternativos, ...)

7.2.4 Justificación e inspección de las soluciones

Para que un sistema sea creíble ha de poder justificar y mostrar sus decisiones, para ello el experto ha de ser capaz de añadir a las reglas la justificación de su aplicación y de las condiciones que tienen. También puede incluir reglas que permitan interrogar al sistema sobre las deducciones que se han obtenido y las cadenas de deducción que han llevado a ellas.

Esta justificación se apoyará en la información incluida por el experto en el sistema y en la traza de la resolución que deberá almacenar el motor de inferencia en la memoria de trabajo para su posterior inspección.

Dos preguntas típicas que debería poder responder un SBC en cualquier punto de la resolución son:

- **Porqué:** Que objetivos globales se desean resolver.
- **Cómo:** Que cadena de razonamiento ha llevado a ese punto.

Podemos distinguir dos niveles de explicación:

Muestra: Traza que describe los pasos que se han realizado en la cadena de razonamiento, las reglas que se han utilizado y los hechos deducidos.

Justificación: Razones por las que se ha escogido una línea de razonamiento, porqué se preguntan ciertas cosas al usuario, tipo de subproblema en el que se ha clasificado la solución, razones de las preferencias utilizadas, ...

Las explicaciones pueden consistir en textos prefijados añadidos a las reglas o pueden generarse explicaciones en lenguaje natural que dependan del contexto.

7.2.5 Aprendizaje

Habitualmente las aplicaciones que se crean para solucionar problemas en un dominio están acotadas y delimitan el conjunto de problemas que se pueden resolver. Esto es adecuado en muchos dominios, pero otros requieren que el sistema sea capaz de aprender a resolver nuevos problemas o adaptar su comportamiento al entorno.

Hay diversas maneras de abordar el problema del aprendizaje en un SBC. Si partimos de un sistema basado en reglas, una posibilidad es el poder crear o corregir reglas del sistema cuando se detectan fallos en las soluciones. Para ello es necesario un proceso de razonamiento que detecte en que parte de la solución aparece el problema. Otra posibilidad es que el sistema sea capaz de crear sus propias reglas de control analizando las trazas de las resoluciones que ha obtenido. En estas trazas se puede detectar en qué puntos el mecanismo de resolución perdió el objetivo y fue necesario

replantear la estrategia de resolución. Creando nuevas reglas de control se puede evitar esta pérdida de tiempo en futuras resoluciones. Este tipo de aprendizaje se conoce como *aprendizaje basado en explicaciones*.

Otras posibilidades de aprendizaje aparecen en dominios en los que es difícil obtener directamente reglas de producción por la complejidad de la tarea a resolver. En estos dominios se intenta aprender un modelo de la tarea a resolver. Este modelo por lo general se construye por observación de ejemplos de soluciones de la tarea. A partir de estos ejemplos se construye una generalización que permitirá resolver los tipos de problemas que describen los ejemplos. Este modelo puede consistir en un conjunto de reglas que describe la tarea u otro formalismo equivalente. Las técnicas utilizadas caen dentro de lo que se denomina *aprendizaje inductivo*

7.3 Arquitectura de los sistemas basados en casos

El *razonamiento basado en casos* (*Case Based Reasoning*) supone que la solución de un problema se puede obtener identificando una solución anterior similar y adaptándola a las características particulares del problema.

Este método de resolución está apoyado por estudios en psicología cognitiva que indican que muchas veces esa es la manera en la que personas expertas en un problema llegan a obtener sus soluciones. De este modo, la necesidad de extraer conocimiento encaminado a la resolución se reduciría en gran medida y esta se centraría en la identificación y adaptación de casos similares.

Existen muchos problemas que pueden ser resueltos mediante esta metodología, presentando algunas ventajas:

- Reducen en gran medida el proceso de explicitación y extracción del conocimiento y, sobre todo, reducen la necesidad de formalizarlo mediante un formalismo procedimental, por ejemplo, reglas de producción.
- Mejoran la capacidad de mantenimiento del conocimiento al permitir añadir o corregir fácilmente los casos
- Hacen más eficiente la resolución de problemas al reducir la necesidad de utilizar mecanismos de razonamiento
- Es cercano a la experiencia de los usuarios, ya que permite la explicación de las soluciones a partir de casos conocidos

El ciclo de resolución de los sistemas basados en casos es diferente del de los sistemas de producción. Estos se basan en un ciclo que incluye cuatro fases:

1. **Fase de recuperación:** A partir del problema se deben identificar los casos almacenados que tengan mayor similaridad
2. **Fase de reuso:** Escogemos como solución inicial la que aparece en el caso recuperado
3. **Fase de revisión:** Evaluamos la solución recuperada y la adaptamos al caso particular, evaluando su resultado. Esta revisión puede necesitar un proceso de razonamiento.
4. **Fase de retención:** Evaluamos si es útil guardar la información de la nueva solución obtenida para poder solucionar nuevos problemas.

La figura 7.2 muestra las fases gráficamente. Este ciclo de resolución encaja con los elementos de la arquitectura general que hemos visto.

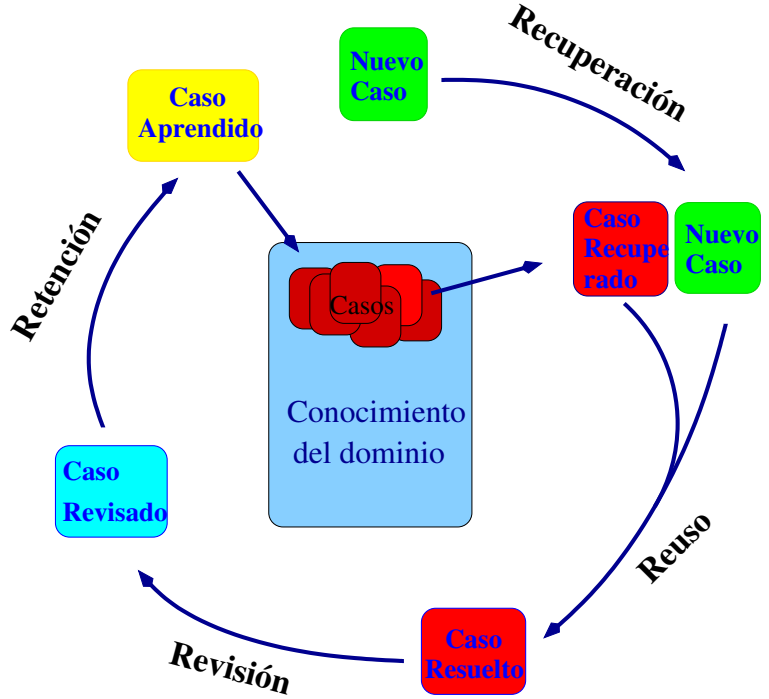


Figura 7.2: Ciclo de ejecución del razonamiento basado en casos

7.3.1 Almacenamiento del conocimiento

En esta arquitectura el sistema de almacenamiento del conocimiento tendrá dos elementos. Por un lado tendremos el sistema que almacenará los casos, denominado *base de casos*. Esta almacenará un conjunto de soluciones que se hayan escogido como representativas y que se podrán usar en resoluciones de nuevos problemas. Por otro lado estará el conocimiento específico del dominio que por lo general se expresa a partir de reglas o procedimientos. Estos representarán la información necesaria para determinar la similaridad de los casos, la adaptación de la solución recuperada al caso actual y la evaluación de la solución.

La información que describe un caso puede ser bastante compleja. Será no solo necesaria información descriptiva que permita la recuperación de los casos almacenados a partir de casos nuevos, sino también información sobre la solución o justificaciones que permitan explicar la solución.

Otro punto importante será la organización de la base de casos para la recuperación de los casos similares. La eficiencia computacional exige que exista alguna organización que ayude a la recuperación. Por lo general esta organización es jerárquica y tiene algún tipo de indexación. Hay que tener en cuenta que la recuperación no tiene por que basarse directamente en las características que describen el caso, sino que puede ser necesario cierto tipo de cálculo o razonamiento para evaluar la similaridad. También puede haber cierta noción de distancia que determine cómo de similares son los casos que se recuperan.

7.3.2 Uso e interpretación del conocimiento

El mecanismo de uso del conocimiento se basa directamente en el ciclo de funcionamiento de un sistema basado en casos.

El sistema será capaz de identificar el conjunto de casos más similares al problema actual utilizando la base de casos. Este proceso de recuperación dependerá de la información que contengan los casos y la estructura en la que esté organizada la base de casos. Se recorrerá esta estructura evaluando la similaridad de los casos con el caso actual de la manera que esté definida. Por lo general

se utilizará una función de similaridad o un proceso de razonamiento sencillo para puntuar los casos.

La solución se construirá a partir del caso mas similar o la combinación de soluciones de los casos más similares. El sistema deberá instanciar la solución obtenida para el caso actual y comprobar si es válida tal como está. Esto puede requerir otro proceso de razonamiento.

Si la solución no es válida se puede utilizar el conocimiento del dominio del que se dispone para intentar adaptar la solución. Durante esta adaptación el conocimiento del dominio deberá determinar que elementos son incorrectos en la solución y proponer alternativas.

El resultado de este proceso será una solución completa.

7.3.3 Almacenamiento del estado del problema

En este caso el estado del problema está formado únicamente por el caso actual, el caso recuperado y la información que se infiera en los procesos de recuperación del caso mas similar, la evaluación de la solución y su adaptación.

7.3.4 Justificación e inspección de las soluciones

La justificación en los sistemas de razonamiento basado en casos está almacenada en los propios casos. Esta información deberá ser complementada por el proceso de razonamiento de la adaptación cuando se necesite adaptar la solución.

En estos sistemas la justificación es más natural ya que la solución se ha definido a priori a partir de un caso concreto que es mas sencillo de entender por un usuario.

7.3.5 Aprendizaje

El proceso de aprendizaje en estos sistemas es mas sencillo y natural, ya que simplemente se ha de añadir a la base de casos el nuevo caso y su solución. La ventaja respecto a los sistemas basados en reglas es que no se ha de tocar el conocimiento que ya tiene el sistema.

La decisión de añadir un nuevo caso debe ser evaluada por el sistema teniendo en cuenta lo similar que sea a casos ya existentes. El objetivo es completar la visión que se tiene del dominio, así que nuevos casos suficientemente diferentes de los conocidos son importantes.

También es posible plantearse la posibilidad de olvidar casos que tengan poca utilidad eliminándolos de la base de casos.

7.4 Comunicación con el entorno y/o el usuario

Dada la necesidad de interacción del sistema con el usuario y/o el entorno deberá existir un interfaz de comunicación. Este puede ser relativamente simple (preguntas cerradas, menús) o permitir una interacción mas natural con el usuario (interfaz en lenguaje natural).

Debería permitir:

- Introducir los datos del problema a resolver
- Preguntar sobre el estado interno: Sobre la resolución (explicaciones, reglas utilizadas, hechos), sobre suposiciones alternativas (what if), sobre el estado de la memoria de trabajo
- Preguntar al usuario: sobre hechos, petición de confirmaciones, ...

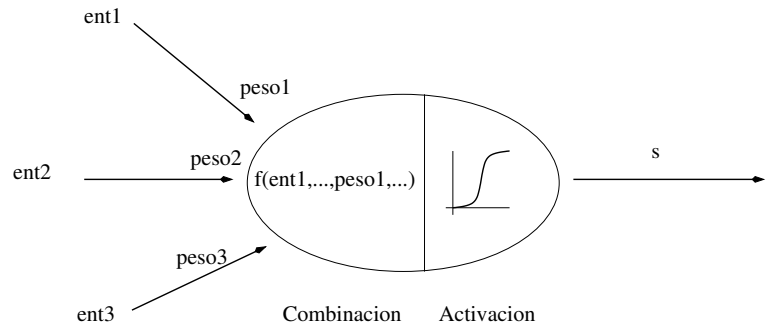


Figura 7.3: Esquema de una neuronal artificial

En sistemas complejos un SBC no tiene por que interactuar solamente con usuarios humanos sino que puede colaborar con otros SBC para realizar su tarea. Esto hace necesario establecer primitivas de comunicación a través de las cuales se puedan hacer preguntas. Existen lenguajes de comunicación entre SBC estandarizados que definen las comunicaciones que se pueden realizar y su semántica. Estos lenguajes suponen que existe una ontología de dominio común entre los SBC que interactúan que permiten que el contenido de las comunicaciones sea comprendido por ambas partes.

7.5 Otras Metodologías

Los sistemas basados en reglas y el razonamiento basado en casos son arquitecturas habituales sistemas basados en el conocimiento. No obstante existen otras aproximaciones que a veces se adaptan mejor a los problemas que se pretenden resolver.

Comentaremos sucintamente algunas de ellas.

7.5.1 Redes neuronales

Las redes neuronales (o redes de neuronas artificiales) forman parte de una visión no simbólica de la inteligencia artificial y entran dentro del denominado modelo conexionista. Las arquitecturas descritas hasta ahora basan su funcionamiento en una descripción explícita del conocimiento y de sus mecanismos de resolución. El modelo conexionista se basa en la utilización de conjuntos de elementos simples de computación denominados neuronas (ver figura 7.3).

Las neuronas se describen a partir de unas entradas, una salida, un estado y un conjunto de funciones que se encargan de combinar las entradas, cambiar el estado de la neurona y dar el valor de salida. El funcionamiento de una neurona es muy sencillo, los valores de la entrada y el estado de la neurona se combinan y se obtiene un valor de salida, esta salida sería la respuesta de la neurona a los estímulos de entrada. La interconexión de estos elementos entre si permite realizar cálculos complejos.

Las neuronas se agrupan en lo que se denomina redes de neuronas (ver figura 7.4). Estas redes están organizadas en capas interconectadas. Por lo general se distingue **la capa de entrada**, que es la que recibe la información del exterior, **la capa de salida**, que es la que da la respuesta y **las capas ocultas**, que son las que realizan los cálculos.

Las redes neuronales asocian un conjunto de entradas con un conjunto de respuestas. Por lo general las entradas correspondientes a la capa de entrada suelen corresponder con la información del problema y los valores que corresponden a las salidas de la capa de salida codifican la respuesta al problema.

En este caso para construir un sistema basado en el conocimiento a partir de redes neuronales

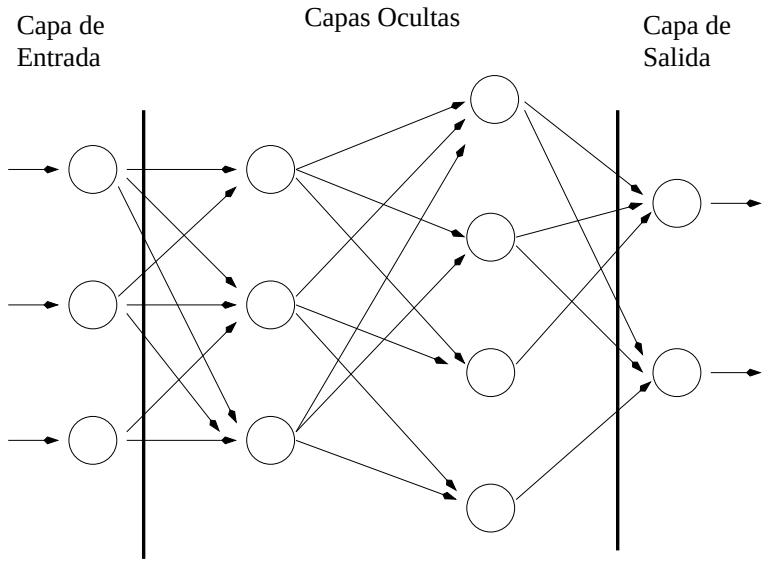


Figura 7.4: Red neuronal

se debe decidir como se han de codificar los datos de entrada y la salida, el número de neuronas en cada una de las capas, el número de capas ocultas y la interconexión entre cada capa de neuronas.

La asociación entre entradas y salidas se realiza mediante técnicas de aprendizaje automático. A la red de neuronas se le van presentando ejemplos solucionados y, mediante algoritmos adecuados, las neuronas de la red van quedando en un estado que describe la asociación entre problemas y soluciones. Este periodo de aprendizaje, también llamado entrenamiento es la parte mas compleja y requiere escoger adecuadamente el conjunto de ejemplos y el número de veces que se presentan a la red.

Los principales inconvenientes de las redes neuronales son el no disponer de una descripción explícita de la resolución y en consecuencia no poder dar una justificación en términos que pueda comprender un experto. No obstante, los métodos conexionistas se usan con gran éxito en muchas aplicaciones.

7.5.2 **Razonamiento basado en modelos**

El *razonamiento basado en modelos*, junto al razonamiento cualitativo, pretende resolver problemas a partir de construir un modelo del sistema sobre el que se ha de trabajar. A diferencia de los modelos utilizados en simulación, que se basan fundamentalmente en información y métodos numéricos, estos modelos serían adecuados para problemas en los que estos métodos no son aplicables por la complejidad del sistema.

La descripción del sistema se realizaría a partir de información cualitativa y reglas y ecuaciones cualitativas que describirían sus relaciones. Este modelo permitiría obtener deducciones sobre las consecuencias de las acciones que se pueden realizar en el problema y razonar sobre el comportamiento del sistema.

Este tipo de modelado se acercaría a lo que se considera razonamiento del sentido común y que permitiría obtener soluciones a problemas sin tener que entrar la complejidad del funcionamiento real del problema.

Un ejemplo de este tipo de metodología es la denominada *física naïf* (*naïve physics*), en la que se pretende incluir información del sentido común en el razonamiento sobre los fenómenos físicos.

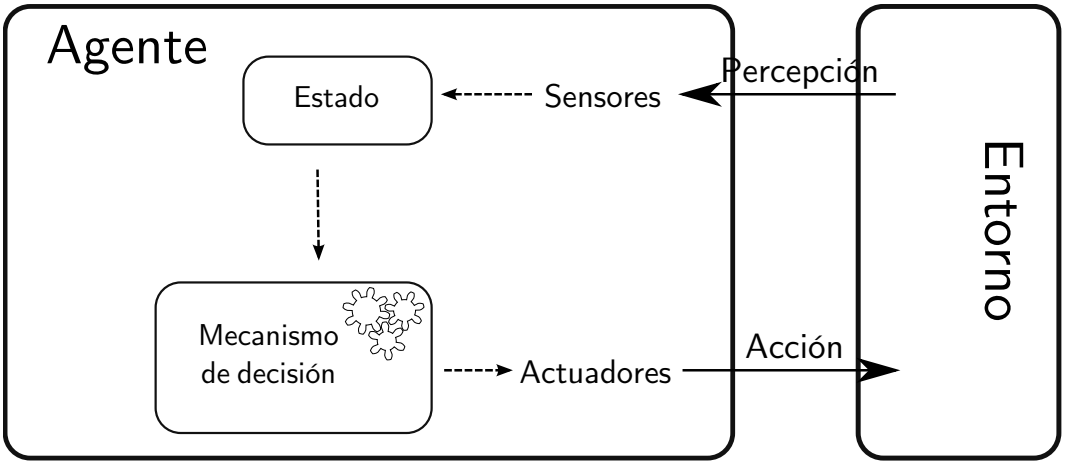


Figura 7.5: Agente inteligente

7.5.3 **Agentes Inteligentes/Sistemas multiagente**

La visión de todas las metodologías mencionadas de los sistemas basados en el conocimiento es una visión monolítica. La tarea a realizar es muy compleja y existe un único sistema que tiene todo el conocimiento y realiza todas las acciones.

El área de agentes inteligentes intenta ampliar esta visión, considerando que tener múltiples agentes que resuelve problemas más sencillos y que se encuentran en un entorno en colaboración/competición entre sí, permite también resolver esos problemas complejos. Podríamos decir que las otras arquitecturas tendría una visión de agente único y el área de agentes inteligentes tendría una visión de colectividad.

Un agente sería un sistema que obtiene información del entorno (percepción) elabora una decisión basada en estas percepciones y su representación interna del estado (razonamiento) y realiza una acción mediante sus actuadores (actuación) (ver figura 7.5). El mecanismo de decisión es el elemento más complejo y se puede resolver de múltiples maneras, dando así más o menos capacidades al agente.

Los agentes inteligentes pueden considerarse más como un paradigma de desarrollo de sistemas basados en el conocimiento que una arquitectura en sí, ya que las metodologías utilizadas para construir los agentes individuales son las que ya se han descrito.

La diferencia esencial es la división de un trabajo complejo entre pequeños sistemas que tienen capacidad de razonamiento y que para resolver el problema global deben comunicarse y coordinarse. Es una visión social de la inteligencia que podemos ver en la resolución de muchos problemas.

Ejemplo 7.1 *Un ejemplo evidente de sociedad de agentes es el comportamiento de los insectos sociales. Cada individuo es relativamente simple y tiene una serie de capacidades que le permite resolver un conjunto de tareas, la organización de estos y su división del trabajo les permite llegar a resolver problemas complejos.*

Esta visión social hace necesario incluir técnicas que no están presentes en las otras metodologías, entre ellas:

- Organización y estructura de la comunidad de agentes (reglas, normas, prohibiciones)
- Asignación y compartición de recursos, resolución de conflictos y negociación
- Cooperación, división del trabajo, planificación multiagente
- Lenguajes de comunicación

- Razonamiento sobre los otros agentes, introspección

Los agentes inteligentes aportan una flexibilidad al desarrollo de sistemas basados en el conocimiento que no permiten otras metodologías. Cada uno de los elementos que forma parte de la comunidad se puede reorganizar para resolver otros problemas, de manera que se puede reaprovechar lo desarrollado a la manera de la programación a partir de componentes, pero con la ventaja de la capacidad de razonamiento que incluye cada agente. Esto se puede llevar todavía mas allá si es el propio agente el que busca y recluta a otros agentes que le permitan resolver partes de su problema que él por sí solo no puede resolver.

El desarrollo de internet está íntimamente relacionado con los agentes inteligentes y tiene áreas de interés común con por ejemplo los sistemas Grid o los servicios web.

8.1 Ingeniería de sistemas basados en el conocimiento

El punto clave del desarrollo de un sistema basado en el conocimiento es el momento de traspasar el conocimiento que posee el experto a un sistema real. En este proceso no sólo se han de captar los elementos que componen el dominio del experto, sino que también se han de adquirir las metodologías de resolución que utilizan éstos.

Este trabajo de extracción del conocimiento se realiza durante la interacción entre dos personajes, el *ingeniero del conocimiento* (persona que conoce el formalismo de representación que utilizará el SBC) y el experto (persona que posee el conocimiento, pero que no tiene por que usar un formalismo para representarlo).

Todo este proceso ha de encajarse en las metodologías de desarrollo de software y adaptarse a las características específicas que tienen los sistemas basados en el conocimiento.

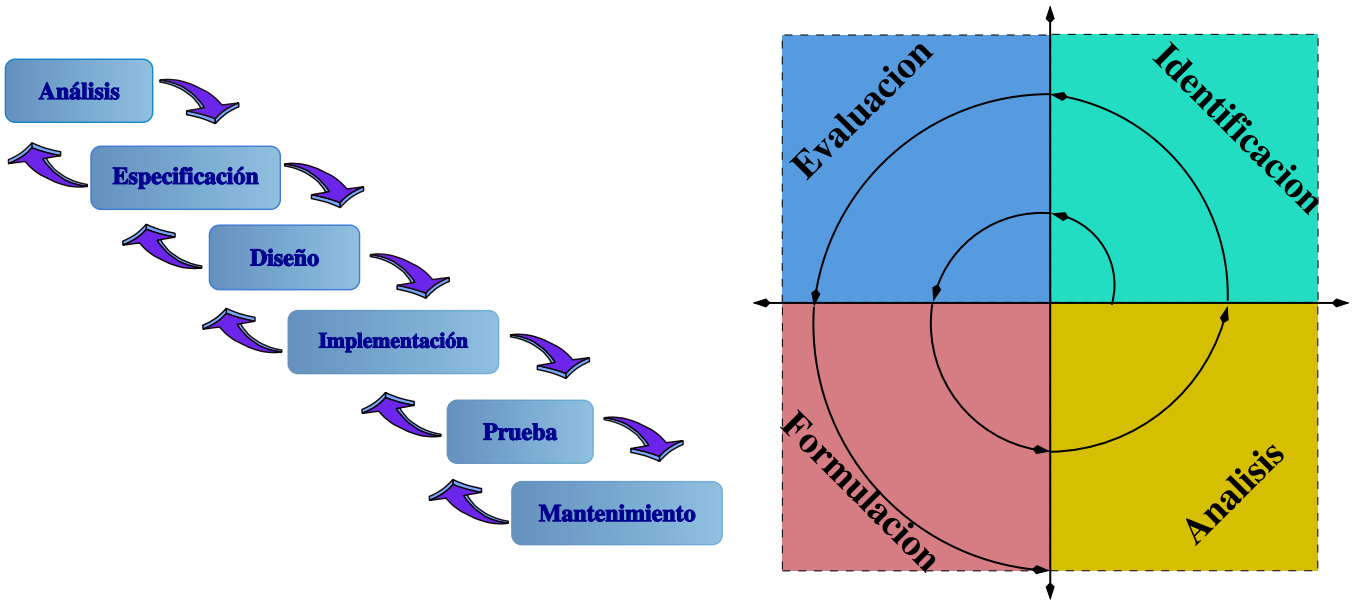
Como cualquier otro desarrollo de software el desarrollo de sistemas basados en el conocimiento puede hacerse siguiendo las metodologías existentes de ingeniería de software, adaptándolas a las diferentes particularidades que estos tienen. En las siguientes secciones describiremos brevemente diferentes metodologías de ingeniería de software que se pueden utilizar.

8.1.1 Modelo en cascada

Esta metodología es utilizable en desarrollos de tamaño pequeño o mediano. Se divide en seis fases que describiremos brevemente:

- **Análisis del problema:** Se determina la factibilidad de resolver el problema a partir de la información disponible y su viabilidad económica.
- **Especificación de requerimientos:** Se redacta un documento de especificación que describe los objetivos y los elementos que deberá poseer el sistema.
- **Diseño:** Se determina la arquitectura del sistema desde el nivel de mayor abstracción hasta el nivel más específico, describiendo las interacciones entre los diferentes elementos que formarán el sistema
- **Implementación:** Se programará cada uno de los módulos del sistema, integrándolos en un único sistema.
- **Prueba:** Se determinará si el sistema implementado cumple con las especificaciones iniciales
- **Mantenimiento:** Se corregirán los posibles errores que se descubran durante el funcionamiento del sistema y se añadirán las modificaciones necesarias para cumplir nuevas capacidades que se pidan al sistema

Las desventajas de esta metodología son que ni el desarrollador, ni el usuario final pueden hacerse una idea del resultado final hasta que el sistema sea completado y que el usuario final no puede utilizar ningún elemento del sistema hasta el final de proyecto.



(a) Ciclo de vida en cascada (b) Ciclo de vida en espiral

Figura 8.4: Metodologías de ingeniería de software

8.1.2 Modelo en espiral

Este modelo combina las ideas del modelo en cascada con las de prototipado y análisis de riesgos, viendo el desarrollo como la repetición de un ciclo de pasos. Estos pasos son:

- **Identificación:** Se determinan los objetivos del ciclo, las alternativas para cumplirlos y las diferentes restricciones de estas alternativas.
- **Evaluación:** Se examinan los objetivos y restricciones para descubrir posibles incertidumbres y riesgos.
- **Formulación:** Se desarrolla una estrategia para resolver las incertidumbres y riesgos.
- **Análisis:** Se evalúa lo realizado para determinar los riesgos remanentes y se decide si se debe continuar con el elemento actual o se puede pasar al siguiente elemento o paso de desarrollo.

El punto clave de esta metodología es saber identificar lo antes posible los riesgos involucrados en el desarrollo para que estos no tengan un impacto negativo en el proceso.

8.1.3 Diferencias de los sistemas basados en el conocimiento

Al igual que otros sistemas software, los sistemas basados en el conocimiento tiene el objetivo de crear soluciones computacionales a problemas, por ello el desarrollo de estos sistemas es parecido al del software convencional. Sin embargo, hay diferencias que hay que tener en cuenta.

La primera de ellas es el tipo de conocimiento que se representa. En los sistemas software habituales se implementan procedimientos algorítmicos bien conocidos y de uso común, al contrario que en los sistemas basados en el conocimiento, que involucran conocimiento incompleto, impreciso y de naturaleza heurística que es conocido únicamente por un limitado conjunto de expertos. Este conocimiento debe ser transferido de estos expertos a una solución software, este proceso es denominado **adquisición del conocimiento** (*Knowledge Elicitation*).

Por lo general, el sistema que utiliza el ingeniero del conocimiento para obtener el conocimiento del experto es el de entrevistas. Durante estas entrevistas el ingeniero ha de ayudar al experto a sistematizarlo, consiguiendo que vaya explicitando el conocimiento del dominio y las diferentes técnicas que utiliza para resolver los problemas que deberíamos incluir en nuestro sistema. Con esto pretendemos transformar este conocimiento de manera que se puedan representar en un formalismo computable. Este proceso de extracción del conocimiento es bastante lento y costoso.

Varias son las dificultades que hacen que este proceso sea tan costoso:

- La naturaleza especializada del dominio hace que el ingeniero del conocimiento deba aprender unas nociones básicas de éste para que pueda establecerse una comunicación efectiva con el experto (vocabulario básico, elementos que intervienen en el dominio, formalismos que utilizan los expertos, etc.).

- La búsqueda de un formalismo de representación que se adapte adecuadamente al problema y que sea fácil de interpretar y adoptar por el experto. Este formalismo ha de ser susceptible de ser transformado en algo computable.

- Los expertos se encuentran más cómodos pensando en términos de ejemplos típicos que razonando en términos generales, que son de los que realmente se podría hacer una mejor abstracción.

- Por lo general, a los expertos les es muy difícil explicitar los pasos que utilizan para resolver los problemas. Es lo que se ha denominado *paradoja del experto*. Cuanta más experiencia, menos explícitos son los razonamientos del experto y más ocultos los métodos de resolución.

  Si observamos como un experto resuelve un problema, éste omite muchas cadenas de razonamiento e información que da por supuestas, y a las que no asigna importancia dentro de la resolución, pero que son necesarias si se quiere abordar el problema de manera sistemática.

Con todas estas circunstancias, podemos observar que la auténtica dificultad de la extracción del conocimiento estriba en descubrir los métodos mediante los que se usa el conocimiento en la resolución y no tanto en la adquisición del conocimiento estático del problema (elementos del problema y relaciones)

Sobre la adquisición de los elementos básicos del dominio, existen bastantes herramientas automáticas, encuadradas dentro del área del aprendizaje automático (*Machine Learning*), que permiten reducir el esfuerzo.

La segunda diferencia es la naturaleza y la cantidad del conocimiento. En los sistemas software tradicionales es posible estimar estas dos cualidades de manera sencilla, en los sistemas basados en el conocimiento ésto es mucho más difícil. Incluso para los expertos en el dominio es difícil hacer esta evaluación, lo que complica el calcular el esfuerzo necesario para desarrollar el sistema.

Esta dificultad puede hacer que sea complicado obtener un diseño adecuado en las fases iniciales del proyecto pudiendo suceder que durante el proceso de desarrollo se descubra que alguno de los elementos decididos no son realmente adecuados para resolver el problema. Esto puede llevar a tener que decidir entre continuar el desarrollo sabiendo que ese problema tendrá consecuencias en las fases restante o empezar de nuevo el proyecto. La manera mas adecuada para minimizar este problema es el uso de técnicas de desarrollo incremental y de prototipado rápido (*rapid prototyping).*

Estás técnicas son bastante populares en el desarrollo de sistemas basados en el conocimiento y están favorecidas por los lenguajes utilizados en este tipo de problemas (PROLOG, LISP, CLIPS, ...) que permiten la construcción rápida de un prototipo funcional del sistema final. Este prototipo recogerá un subconjunto de las funcionalidades del sistema y permitirá observar la idoneidad de las decisiones.

En el ciclo de vida de este tipo de desarrollo, las fases de análisis y especificación deben realizarse teniendo en cuenta el sistema completo, pero el diseño e implementación se realizan de una manera

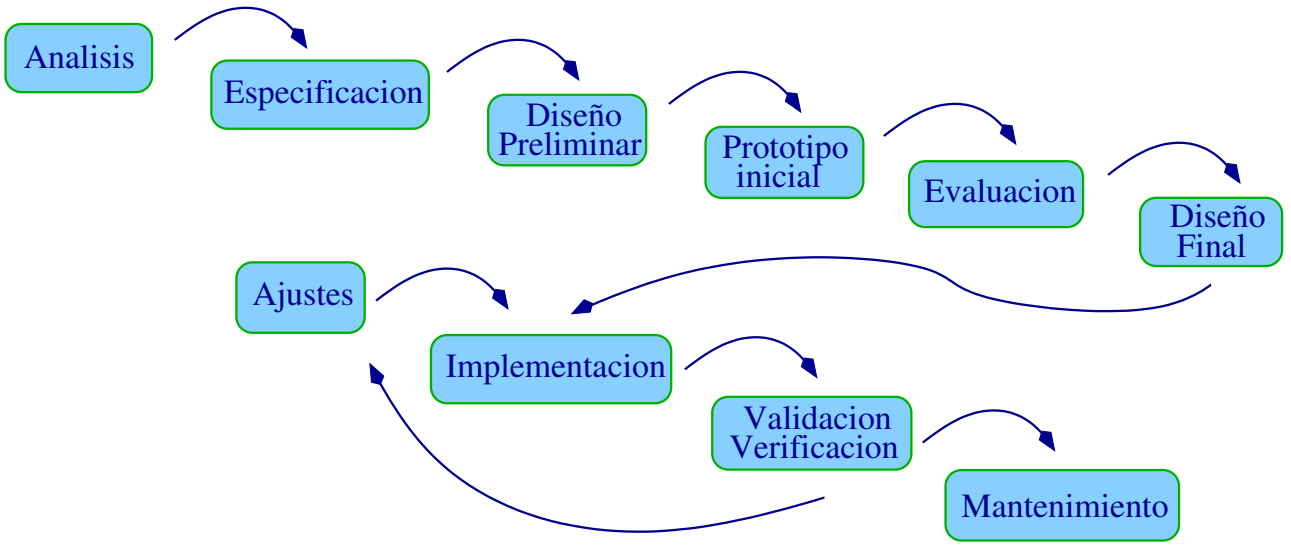


Figura 8.5: Ciclo de vida de un SBC

mas preliminar y reduciendo las necesidades. Esto permite disponer de un sistema funcional que puede ser evaluado para obtener una valoración de las decisiones tomadas.

A partir del prototipo inicial se puede aplicar la metodología de desarrollo incremental. Esta metodología se basa en la división de problemas y el desarrollo iterativo, de manera que se va completando el sistema añadiendo nuevos módulos y funcionalidades. Esto permite tener un sistema funcional durante todo el desarrollo.

8.1.4 Ciclo de vida de un sistema basado en el conocimiento

Podemos modificar el ciclo de vida convencional del desarrollo de software para adaptarlo a las necesidades del desarrollo de sistemas basados en el conocimiento, estas serían las fases en las que dividiríamos el proceso (ver figura 8.5):

1. **Análisis del problema:** Se han de evaluar el problema y los recursos disponibles para determinar si es necesario y viable el desarrollo de un sistema basado en el conocimiento para resolverlo.

2. **Especificación de requerimientos:** Formalizar lo que se ha obtenido en la fase de análisis, fijando los objetivos y los medios necesarios para su cumplimiento.

3. **Diseño preliminar:** Se toman decisiones a alto nivel para crear el prototipo inicial. Estas decisiones deben incluir la forma de representar el conocimiento y las herramientas que se van a utilizar para el desarrollo del prototipo. En esta fase se hará necesario obtener información general sobre el problema a partir de las fuentes disponibles para poder tomar las decisiones adecuadas.

4. **Prototipo inicial y evaluación:** El prototipo inicial debe tener las características de la aplicación completa pero su cobertura del problema debe estar limitada. Este prototipo debe permitirnos poder evaluar las decisiones tomadas. Esto hace necesario obtener una cantidad de conocimiento suficiente para evaluar la forma escogida de representarlo y utilizarlo. El objetivo es poder detectar los errores que hayamos podido cometer en el diseño preliminar para poder corregirlos en esta fase.

5. **Diseño final:** En esta fase debemos tomar las decisiones finales con las que vamos a continuar el desarrollo del sistema, las herramientas y recursos a usar y, sobre todo, la forma de representar el conocimiento. Se debe también hacer un diseño de la estructura de la aplicación que nos permita un desarrollo incremental.

6. **Implementación:** En esta fase debemos completar la adquisición del conocimiento necesaria para desarrollar el sistema completo. En esta fase debemos aplicar la metodología de desarrollo incremental implementando cada uno de los módulos de la aplicación.

7. **Validación y verificación:** Se ha de validar el sistema construido y verificar que cumpla las especificaciones del problema. La validación de estos sistemas es mas compleja que en los sistemas software convencionales dada la naturaleza de los problemas que se resuelven.

8. **Ajustes de diseño:** Puede ser necesaria cierta realimentación del proceso y la revisión y ajuste de algunas decisiones tomadas en sus fases anteriores, pero no debería suponer grandes cambios de diseño, ya que el prototipo inicial nos debería haber permitido eliminar los posibles errores en el diseño inicial.

9. **Mantenimiento:** Esta fase es similar a la que se realizaría en el desarrollo de un sistema de software convencional.

8.1.5 Metodologías especializadas

Se han desarrollado bastantes metodologías para el desarrollo de sistemas basados en el conocimiento, algunas genéricas, otras más especializadas, éstas son un ejemplo:

CommonKADS

Una de las metodologías mas conocidas es KADS (Knowledge Acquisition and Documentation Structuring), cuya evolución se denomina CommonKADS.

CommonKADS utiliza un ciclo de vida en espiral y aborda el desarrollo de sistemas basados en el conocimiento como un proceso de modelado utilizando técnicas y notación parecidas a UML.

El desarrollo se basa en la creación de un conjunto de seis modelos: organización, tareas, agentes, comunicación, conocimiento y diseño. Estos seis modelos están interrelacionados y se desarrollan a partir de un conjunto de plantillas que provee la metodología.

Los tres primeros modelos permiten determinar si el problema puede ser resuelto mediante sistemas basados en el conocimiento. Modelan la estructura de la aplicación, las capacidades que ha de tener y los elementos que son necesarios para la resolución. En el modelo de conocimiento se ha de representar el conocimiento que es necesario para resolver el problema, tanto la descomposición en subproblemas, el conocimiento del dominio (ontología), como los procesos de inferencia involucrados. El modelo de comunicación se encarga de describir la interacción entre los elementos de la aplicación y la información que se intercambia. El último modelo describe el diseño de la aplicación.

MIKE

MIKE (Model-Based and Incremental Knowledge Engineering) es otra metodología también basada en el ciclo de vida en espiral. Este ciclo esta compuesto por cuatro fases que se repiten: adquisición del conocimiento, diseño, implementación y evaluación.

La fase de adquisición del conocimiento comienza con un proceso de extracción del conocimiento del que se obtiene un modelo semiformal del conocimiento necesario a partir de entrevistas con los expertos, este modelo es denominado *modelo de adquisición*. A partir de esta descripción semiformal del conocimiento se crea un modelo formal que describe todos los aspectos funcionales y no

funcionales del sistema, este modelo es denominado *modelo de estructura*. A partir de este modelo se realiza un proceso de formalización y operacionalización que utiliza un lenguaje especializado (KARL, Knowledge acquisition representation language) que combina una descripción conceptual del sistema con una especificación formal.

Este último modelo es la entrada de la fase de diseño en la que se tienen en cuenta los requisitos no funcionales y se genera un nuevo modelo mediante otro lenguaje especializado (DesignKARL) en el que se describe el sistema de manera mas detallada. Este diseño es el que pasará a la fase de implementación en el que se plasmará el sistema. Finalmente se realizará la fase de evaluación.

8.2 Una Metodología sencilla para el desarrollo de SBC

En esta sección describiremos una metodología sencilla para el desarrollo de SBC de tamaño pequeño o mediano, basada en un conjunto de fases en las que realizaremos un conjunto de tareas específicas. Está basado en un ciclo de vida en cascada, pero podríamos fácilmente encajarlo en una metodología basada en prototipado rápido y diseño incremental como el descrito en la sección 8.1.4.

Las fases que consideraremos serán las siguientes:

1. **Identificación del problema:** Determinar la viabilidad de la construcción de un SBC para solucionar el problema y la disponibilidad de fuentes de conocimiento.
2. **Conceptualización:** Descripción semiformal del conocimiento del dominio del problema, descomposición del problema en subproblemas.
3. **Formalización:** Formalización del conocimiento y construcción de las ontologías necesarias, identificación de las metodologías de resolución de problemas adecuadas. Decisión sobre las técnicas de representación y herramientas adecuadas.
4. **Implementación:** Construcción del sistema
5. **Prueba:** Validación del sistema respecto a las especificaciones.

8.2.1 Identificación

En esta fase se ha de determinar, en primer lugar, si el problema se puede o se debe abordar mediante las técnicas de los SBC. Para que un problema sea adecuado no ha de poder solucionarse de manera algorítmica, ya que si se pudiera hacer de ese modo, no tendría sentido iniciar una labor tan costosa. También ha de ser necesario tener acceso a las fuentes de conocimiento suficientes para completar la tarea. Por último, el problema a tratar ha de tener un tamaño adecuado para que no constituya una tarea inabordable por su complejidad.

El siguiente paso consiste en buscar las fuentes de conocimiento que serán necesarias para el desarrollo del sistema, las más comunes son:

- Expertos humanos en el dominio del problema.
- Libros y manuales que expliciten el problema y técnicas de resolución.
- Ejemplos de casos resueltos.

Estos últimos serán importantes sobre todo en la última fase de validación, pero se pueden usar también para utilizar técnicas de adquisición automática del conocimiento y obtener de esta manera los elementos básicos que intervienen y sus relaciones.

Con estas fuentes de información se podrán determinar los datos necesarios para la resolución del problema y los criterios que determinen la solución, tanto los pasos que permiten la resolución como su posterior evaluación.

En este momento el ingeniero del conocimiento y el experto podrán realizar una primera descripción del problema, en ésta se especificarán:

- Los objetivos
- Las motivaciones
- Las estrategias de resolución y su justificación
- Las fuentes de conocimiento
- Los tipos de tareas que son necesarias

Este esquema será el punto de partida para plantear las siguientes fases.

8.2.2 Conceptualización

Esta fase pretende obtener una visión del problema desde el punto de vista del experto. Ha de poder permitirnos decidir el conocimiento que es necesario para resolver el sistema y las tareas involucradas para obtener una idea informal del alcance del sistema.

Necesitamos conocer cuales son los conceptos que ha de manejar el sistema y sus características de cara a poder elaborar en la fase siguiente una ontología que los formalice. Hemos de identificar también sus características y necesidades de inferencia para poder elegir el método de representación del conocimiento mas adecuado. Esto significa que deberemos aplicar técnicas de desarrollo de ontologías como un proceso paralelo al desarrollo del sistema basado en el conocimiento.

Hay también que obtener una descomposición del problema en subproblemas, realizando un análisis por refinamientos sucesivos hasta que nos podamos hacer una idea de la relación jerárquica de las diferentes fases de resolución hasta los operadores de razonamiento más elementales.

Otro elemento necesario es descubrir el flujo del razonamiento en la resolución del problema y especificar cuando y como son necesarios los elementos de conocimiento.

Con esta descomposición jerárquica y el flujo del razonamiento se pueden caracterizar los bloques de razonamiento superiores y los principales conceptos que definen el problema. Hará falta distinguir entre evidencias, hipótesis y acciones necesarias en cada uno de los bloques y determinar la dificultad de cada una de las subtareas de resolución. De esta manera se conseguirá captar la estructura del dominio y las diferentes relaciones entre sus elementos.

La manera habitual de realizar estas tareas es la interacción con el experto. En particular, es necesario observar como resuelve problemas típicos y abstrae de ellos principios generales que pueden ser aplicados en diferentes contextos.

Toda esta labor debería darnos un modelo semiformal del dominio y de los problemas y métodos de resolución que se deberán incluir en el sistema.

8.2.3 Formalización

Esta fase ha de darnos una visión del problema desde el punto de vista del ingeniero del conocimiento. Se han de considerar los diferentes esquemas de razonamiento que se pueden utilizar para modelizar las diferentes necesidades de representación del conocimiento y de resolución de problemas identificadas en las fases anteriores. Se deberá decidir el mecanismo de representación adecuado y formalizar el conocimiento de manera que podamos crear la ontología que represente el conocimiento

del dominio. En esta fase se detallarán los elementos que forman parte de la ontología y se iniciará el proceso de su formalización.

Deberemos decidir también la manera más adecuada de representar los procesos de resolución identificados. En este punto, se ha de poder comprender la naturaleza del espacio de búsqueda y el tipo de búsqueda que habrá que hacer. En este momento se pueden comparar los métodos de resolución que aplica el experto en un problema con diferentes mecanismos prototípicos de resolución de problemas como la clasificación, abstracción de datos, razonamiento temporal, estructuras causales, etc. De esta manera decidiremos cuales son los mas adecuados en nuestras circunstancias.

En esta etapa también tendrá que analizarse la certidumbre y completitud de la información disponible, dependencias temporales, o la fiabilidad y consistencia de la información. Se deberá descubrir qué partes del conocimiento constituyen hechos seguros y cuales no. Para estos últimos deberá elegirse alguna metodología de tratamiento de la incertidumbre, de manera que ésta pueda ser modelizada dentro del sistema.

8.2.4 Implementación

En esta fase ya se conocerán la formas más adecuadas para la representación del conocimiento y los métodos que se adaptan a los mecanismos de resolución de problemas. Ahora se podrán decidir las herramientas y lenguajes más adecuados para llevar a cabo la implementación.

Durante esta fase se implementará la ontología que ha de servir de base para la resolución del problema el el formalismo de representación del conocimiento elegido. Decidiremos también como encajar la descomposición de subproblemas de las tareas involucradas en el sistema con las metodologías de resolución de problemas genéricas que habremos identificado en la fase anterior.

Una vez validadas o corregidas las decisiones que se han tomado continuaremos con la construcción del sistema añadiendo incrementalmente nuevos módulos y nuevas funcionalidades. Si procedemos con una metodología basada en prototipado rápido y desarrollo incremental, deberemos construir un prototipo inicial a partir de una restricción de las capacidades del sistema. Con las decisiones iniciales podremos construir el conjunto de módulos necesarios para obtener un prototipo inicial capaz de resolver un subconjunto de problemas y plantearnos las fases del desarrollo incremental.

8.2.5 Prueba

Se ha de elegir un conjunto de casos resueltos representativos y se ha de comprobar el funcionamiento del sistema con estos. Estos casos deberían incluir los utilizados para el diseño del sistema, pero también casos nuevos.

La validación de un sistema basado en el conocimiento es más compleja que en otros sistemas software, dado que estamos implementado resoluciones de problemas de naturaleza heurística, por lo que es muy difícil determinar la completitud y correctitud del sistema. Esto obligará a realizar futuros mantenimientos o ampliaciones de la aplicación a medida que se descubran casos en los que el sistema no funcione correctamente. Estos fallos pueden deberse no solo a errores en la implementación o el diseño, sino también a problemas en la formalización del conocimiento y de los métodos de resolución.

9. Resolución de problemas en los SBC

9.1 Clasificación de los SBC

El abordar la construcción de un SBC en cualquier dominio es una tarea difícil, y sería deseable disponer de un conjunto de metodologías de resolución de problemas que permitieran aproximar soluciones a diferentes tipos de SBC según sus características.

Con esta idea en mente se han realizado clasificaciones de los SBC según las tareas que realizan, para intentar descubrir metodologías comunes y así extraer directrices de análisis en los distintos tipos de dominios.

De esta manera, dada una clase de problema dispondríamos de:

1. Un conjunto de tareas usuales para cada tipo fáciles de identificar.
2. Un conjunto de metodologías de resolución de problemas específicas para cada tipo.
3. Métodos de representación del conocimiento e inferencia adecuados para cada tipo.

Originalmente se realizó una primera clasificación de los SBC atendiendo a las tareas que realizan[1], ésta es:

Sistemas de Interpretación: Infieren descripciones de situaciones a partir de observaciones.

Sistemas de predicción: Infieren consecuencias previsibles de situaciones o eventos.

Sistemas de diagnóstico: Infieren fallos a partir de síntomas.

Sistemas de diseño: Desarrollan configuraciones de objetos que satisfacen ciertas restricciones.

Sistemas de planificación: Generan secuencias de acciones que obtienen un objetivo.

Sistemas de monitorización: Estudian el comportamiento de un sistema en el tiempo y procuran que siga unas especificaciones.

Sistemas de corrección: Genera soluciones para fallos en un sistema.

Sistemas de control: Gobiernan el comportamiento de un sistema anticipando problemas, planeando soluciones.

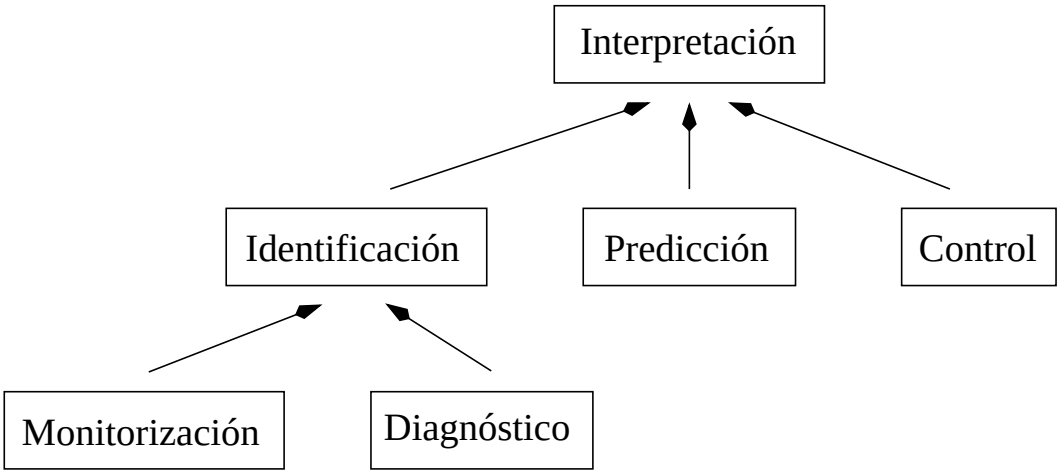
Esta primera clasificación, que es la que se utilizó como punto de partida para la identificación de necesidades para el desarrollo de SBC, plantea varios problemas ya que varias categorías se superponen o están incluidas en otras. No obstante, da una idea inicial de los rasgos comunes que aparecen entre los distintos dominios en los que tratan los sistemas.

Un análisis alternativo, posterior a éste, permite un tratamiento más sistemático de las necesidades de un SBC, éste se basa en las operaciones genéricas que puede hacer un SBC respecto al entorno. Se distinguen dos operaciones genéricas:

[1]Esta clasificación apareció en: F. Hayes-Roth, D. A. Waterman, D. B. Lenat, *Building Expert Systems*, Addison Wesley, Reading, MA, 1983.

- Operaciones de análisis, que interpretan un sistema.
- Operaciones de síntesis, que construyen un sistema.

Estas operaciones se pueden especializar en otras más concretas dando lugar a una jerarquía de operaciones. Para el caso del **análisis** tenemos:



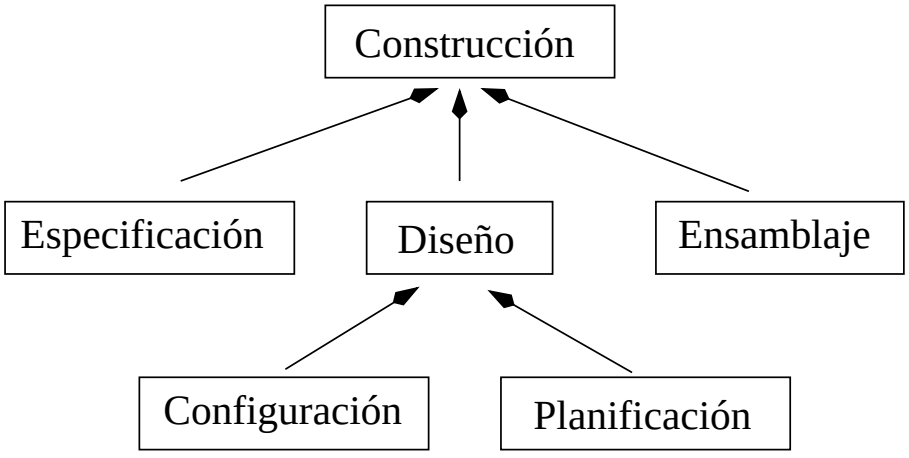
En este caso, la *interpretación* se podría especializar según la relación entre los elementos de entrada/salida de un sistema:

- Identificación, nos dice que tipo de sistema tenemos.
- Predicción, nos dice que tipo de resultado podemos esperar.
- Control, determina que entradas permiten conseguir la salida deseada.

La *identificación* se puede especializar para sistemas con fallos en:

- Monitorización, detecta discrepancias de comportamiento.
- Diagnóstico, explica discrepancias.

Para el caso de las operaciones de *síntesis* tenemos:



La especialización de la *construcción* se puede realizar en:

- Especificación, busca que restricciones debe satisfacer un sistema.

- Diseño, genera una configuración de elementos que satisfacen las restricciones.
  - Configuración, como es la estructura actual del sistema.
  - Planificación, pasos a realizar para ensamblar la estructura.
- Ensamblaje, construye un sistema juntando las diferentes piezas.

Obteniendo una clasificación de las diferentes tareas y operaciones que realiza un SBC podemos establecer una correspondencia entre estos y los métodos de resolución, y de esta manera facilitar la tarea de análisis de los dominios.

9.2 Métodos de resolución de problemas

Diferentes son las técnicas de resolución de problemas que se pueden utilizar para las tareas que debe realizar un SBC. Existen ciertas técnicas generales que se pueden aplicar a diferentes tipos de dominios y tareas. De ellas destacaremos las dos más utilizadas:

- Clasificación Heurística (*Heuristic Classification*)
- Resolución Constructiva (*Constructive Problem Solving*)

9.2.1 Clasificación Heurística

La clasificación es un método utilizado en muchos dominios. El elemento esencial de ésta consiste en que el experto escoge una categoría de un conjunto de soluciones previamente enumerado.

En dominios simples, el disponer de las características esenciales de cada una de las categorías es suficiente para establecer la clase del problema y su solución. Esto no ocurre así cuando la complejidad del problema aumenta, pues las características esenciales son cada vez más difíciles de identificar. El objetivo de la técnica de clasificación heurística será obtener y representar el conocimiento necesario para que la asociación problema-solución se pueda realizar.

Se define como *clasificación heurística* a toda asociación no jerárquica entre datos y categorías que requiere de inferencias intermedias. Es decir, el establecer la clase de un problema requiere realizar inferencias y transformaciones sobre éste, para poder asociarlo con la descripción de la clase. El esquema de razonamiento para hacer estas inferencias se ha de adquirir del experto.

La clasificación heurística se divide en tres etapas:

1. **Abstracción de los datos**

   Por lo general, se hace una abstracción del caso concreto para acercarlo a las soluciones que se poseen.

2. **Asociación heurística**

   Se busca la mayor coincidencia entre el caso abstraído y las soluciones. Esta asociación es de naturaleza heurística, es decir, depende de conocimiento basado en la experiencia, y, por lo general, la correspondencia entre caso y soluciones no será uno a uno, existirán excepciones, y las coincidencias no serán exactas.

   La solución corresponderá con la que mejor coincida con la abstracción de los datos.

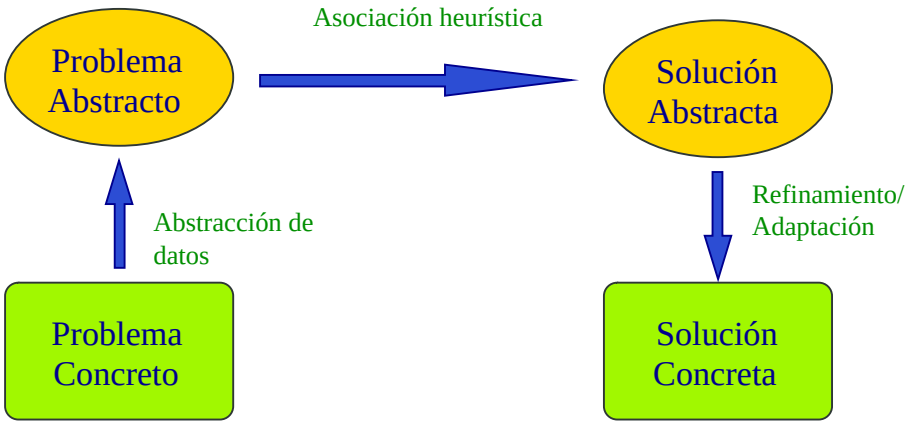


Figura 9.1: Pasos de la clasificación heurística

3. **Refinamiento de la solución**

   Haber identificado la abstracción de la solución reducirá el espacio de búsqueda, ahora será necesario buscar la mejor solución determinada por la solución abstracta. Esto puede necesitar de más deducciones, o de la utilización de más información. De esta manera se debe reducir el espacio de búsqueda hasta encontrar la mejor solución.

En la figura 9.1 se puede ver un esquema del proceso.

Dentro de este proceso, un punto importante es la abstracción de los datos. Tres son las más utilizadas:

Abstracción definicional: Se deben extraer las características definitorias del problema y focalizar la búsqueda con éstas. Le corresponde al experto decidir cuales son esas características.

Cualitativa: Supone abstraer sobre valores cuantitativos, convirtiéndolos en cualitativos (e.g.: Fiebre $= 39^oC \Longrightarrow$ Fiebre = alta).

Generalización: Se realiza abstracción sobre una jerarquía de conceptos (e.g.: forma = pentágono $\Longrightarrow$ forma = polígono).

Se puede ver que esta metodología de resolución de problemas capta una gran cantidad de dominios, siendo adecuada para cualquier problema en el que se pueda hacer una enumeración del espacio de soluciones. Es válida para todas las tareas de análisis.

Clasificación heurística en los sistemas de reglas

Por lo general, la construcción de un sistema mediante clasificación heurística basado en reglas es una labor iterativa. A los expertos a veces les es difícil dar las reglas que son capaces de realizar la labor de clasificación, y además encuentran difícil el formalismo de las reglas.

Es proceso de refinamiento del sistema ha de hacerse paso a paso, añadiendo nuevas reglas que cubran nuevos casos y vigilando las interacciones. La metodología que se suele seguir es la siguiente:

1. El experto da las nuevas reglas al IC.
2. El IC cambia la base de conocimiento.
3. El IC prueba casos ya resueltos para comprobar inconsistencias.
4. Si aparecen errores, se comprueba el nuevo conocimiento con el experto y se empieza de nuevo.

5. Se prueban nuevos casos.

6. Si no hay problemas se para, sino se retorna al principio.

Esta labor iterativa se puede realizar de manera independiente para cada uno de los módulos que componen el sistema, reduciendo de esta manera las interacciones entre diferentes partes del conocimiento.

Estrategias de adquisición del conocimiento con clasificación heurística

La aplicación de la clasificación heurística a diferentes problemas ha llevado a métodos que permiten dirigir la explicitación del conocimiento por parte del experto de una manera más sistemática, enfocando la labor de extracción en cada uno de los elementos que componen las reglas (hipótesis, síntomas, causas, cadenas de inferencia, hechos intermedios, confianza en las evidencias y las asociaciones evidencia-conclusión).

El conjunto de conceptos del problema se puede dividir en tres:

- **Las hipótesis:** Soluciones posibles a nuestro problema.
- **Los síntomas:** Características que describen las hipótesis
- **Las causas originales:** Información del problema que lleva a los síntomas

Las causas y los síntomas se relacionarán mediante las *reglas de abstracción*. De los datos originales obtendremos los valores para el conjunto de características que describen a los problemas. Los síntomas y las hipótesis se relacionarán mediante las *reglas de asociación heurística*. De las características que tiene el problema a solucionar deberemos identificar las hipótesis más probables.

En cada conjunto de reglas (reglas de abstracción, reglas de asociación heurística) deberemos observar qué antecedentes están asociados con que consecuentes. El objetivo es escoger aquellos antecedentes que permitan distinguir mejor los consecuentes, ya que estos pueden tener características comunes. El objetivo es escribir las reglas que permitan diferenciar mejor a los consecuentes.

En este proceso es posible que sea necesario introducir conceptos nuevos (**conceptos intermedios**) que pueden determinar una cadena de deducciones entre las premisas originales y las conclusiones. Estas cadenas de deducciones pueden ser complejas, dependiendo del problema.

Durante el proceso de construcción de las reglas podemos observar también la certidumbre de esas asociaciones (confianza con la que los antecedentes permiten deducir los consecuentes), de manera que podamos hacer el tratamiento adecuado.

En el caso de las soluciones, si estas corresponden a abstracciones será necesario determinar las reglas que permiten especializarlas en soluciones concretas.

Aplicación de la clasificación heurística

Como ejemplo de la técnica de clasificación heurística, vamos a plantear un pequeño SBC para la concesión de créditos bancarios para creación de empresas. El propósito de este sistema será examinar las solicitudes de créditos de clientes con pretensiones de crear una empresa, para determinar si se les debe conceder y que cuantía es la recomendable respecto a la que solicitan.

El problema que se nos plantea tiene por lo tanto una labor de análisis que nos ha de predecir la fiabilidad de si cierta persona, en ciertas condiciones, será capaz de devolver un crédito si se lo concedemos. El número de soluciones a las que podemos llegar es evidentemente finito, el crédito se concede, o no se concede, y en el caso de que se conceda, se decidirá si la cuantía solicitada es adecuada o si sólo se puede llegar hasta cierto límite.

Todas estas características indican que la metodología de resolución que mejor encaja es la clasificación heurística, por lo tanto dirigiremos el planteamiento con las fases que necesita.

Deberemos plantear cuatro tipos de elementos que definen el proceso de clasificación heurística y los mecanismos para transformar unos en otros. Primero definiremos como se plantearán los problemas al sistema, es decir, qué elementos se corresponderán con los datos específicos, las solicitudes de crédito.

Esta información ha de definir el estado financiero del solicitante, el motivo por el que pide el crédito, cuanto dinero solicita, etc. Supongamos que una solicitud contiene la siguiente información:

- Si tiene avales bancarios.
- Si tiene familiares que puedan responder por él.
- Si tiene cuentas corrientes, casas, coches, fincas, etc. y su valoración.
- Si tiene antecedentes de morosidad.
- Si ha firmado cheques sin fondos.
- Si tiene créditos anteriores concedidos.
- Tipo de empresa que quiere crear.
- Cantidad de dinero que solicita.

Esta información deberá convertirse mediante el proceso de abstracción de datos en los problemas abstractos a partir de los cuales se hará el razonamiento. Podríamos decidir que nuestras soluciones abstractas quedan definidas por los siguientes atributos:

- Apoyo financiero: Valoración de la capacidad económica para responder al valor del crédito que solicita. Este apoyo se puede evaluar con la información sobre avales y personas allegadas que puedan responder por él.
- Bienes: Dinero o propiedades que puedan usarse para responder por el crédito o que se puedan embargar en caso de no devolución.
- Fiabilidad de devolución: Información sobre si el cliente tiene antecedentes económicos positivos o negativos.
- Compromiso: Información sobre si ya se tienen compromisos económicos con esa persona o si se tienen intereses especiales con ella.
- Viabilidad de la empresa: Tipo de empresa que se quiere crear y su posible futuro.

Supondremos que estos cinco atributos pueden tomar valores cualitativos que estarán dentro de este conjunto: muy bueno, bueno, normal, regular, malo, muy malo.

Para realizar la abstracción de datos se podrían dar un conjunto de reglas que harían la transformación, como por ejemplo:

- si *avales* > un millón euros o tío rico entonces *apoyo financiero* bueno
- si *avales* entre 100000 euros y un millón entonces *apoyo financiero* normal
- si *avales* < 100000 euros entonces *apoyo financiero* malo
- si suma *bienes* < un millón entonces *bienes* malo
- si suma *bienes* entre uno y dos millones entonces *bienes* normal

- si suma *bienes* > dos millones entonces *bienes* bien
- si *cheques sin fondos* o *moroso* entonces *fiabilidad* muy mala
- si *fábrica de agujeros* entonces *viabilidad* muy mala
- si *hamburguesería* o *heladería* entonces *viabilidad* normal
- si *grandes almacenes* o *proveedor de internet* entonces *viabilidad* muy buena
- si *concedido crédito* < 100000 euros entonces *compromiso* regular
- si *concedido crédito* > un millón o *hermano del director* entonces *compromiso* bueno

El conjunto de soluciones abstractas a las que podría dar el análisis de las solicitudes podría ser el siguiente:

- Denegación, no hay crédito para el cliente.
- Aceptación, se acepta el crédito tal como se solicita.
- Aceptación con rebaja, se acepta el crédito, pero se rebaja la cantidad solicitada, harán falta reglas para crear la solución concreta indicando la cantidad final que se concede.
- Aceptación con interés preferente, se concede la cantidad solicitada, pero además se rebaja el interés que normalmente se pone al crédito, en este caso también hará falta generar una solución concreta.

Ahora nos faltan las reglas que nos harán la asociación heurística entre los problemas abstractos y las soluciones abstractas. Un conjunto de reglas que cubre una pequeña parte del espacio de soluciones podría ser:

- si *apoyo financiero*=regular y *bienes*=malo entonces *denegar*
- si *fiabilidad*={mala, muy mala} entonces *denegar*
- si *apoyo financiero*=normal y *bienes*=normal y *viabilidad*=buena entonces *aceptar con rebaja*
- si *apoyo financiero*=bueno y *bienes*=normal y *compromiso*=normal y *viabilidad*=buena entonces *aceptar*
- si *apoyo financiero*=bueno y *bienes*=bueno y *compromiso*=muy bueno y *viabilidad*=muy buena entonces *aceptar con interés preferente*

Por último, nos hacen falta reglas para poder generar soluciones concretas en los casos que son necesarias, algunas reglas podrían ser:

- si *aceptación con rebaja* y *petición* > 500000 euros y *bienes* = 500000 euros entonces *rebaja* a 500000 euros
- si *aceptación con interés preferente* y *petición* > un millón y *bienes* > un millón entonces *rebaja* de un 1 % de interés
- si *aceptación con interés preferente* y *hermano del director* entonces *rebaja* de un 2 % de interés

9.2.2 **Resolución Constructiva**

En contraste con la clasificación heurística, hay dominios en los que las soluciones no se pueden enumerar a priori, sino que la solución ha de construirse. Por ejemplo, en problemas de diseño, de planificación, y, por lo general, en todos los problemas que incluyen tareas de síntesis.

Este tipo de problemas se pueden atacar mediante métodos no guiados por conocimiento, pero obtener una solución satisfactoria es computacionalmente prohibitivo.

Construir una solución necesita que exista un modelo de la estructura y el comportamiento del objeto que se desea construir, modelo que debe contener conocimiento acerca de las restricciones que se deben satisfacer. Este conocimiento debe incluir:

1. Restricciones en la configuración de los componentes (físicas, temporales, ...).
2. Restricciones respecto a las entradas y salidas/precondiciones y postcondiciones de los operadores de construcción.
3. Interacciones entre estos dos tipos de restricciones.

Es necesario también conocimiento que permita evaluar las decisiones que se toman y la solución actual (total o parcial). Este conocimiento es más elaborado que el que se utiliza en los algoritmos de búsqueda heurística o búsqueda local. En estos algoritmos la información heurística es de carácter más general. En la resolución constructiva se debe poder evaluar la bondad de cada paso en las circunstancias particulares de su aplicación.

Dos son las estrategias generales que se siguen para la resolución de este tipo de problemas:

- Proponer y aplicar (*Propose and apply*).
- Mínimo compromiso (*Least commitment*).

Proponer y aplicar

En principio, el experto debe tener una idea clara de la descomposición en tareas del problema y de las relaciones espacio temporales entre éstas para, de esta manera, plantear las restricciones que se tienen que cumplir. Se han de definir también las operaciones que se pueden efectuar en cada estado de la resolución, cuándo se pueden aplicar y cuáles son sus efectos. Los pasos que se siguen en esta metodología son los siguientes:

1. Inicializar el objetivo: Se crea el elemento que define la solución actual
2. Proponer un operador: Se seleccionan operaciones plausibles sobre la solución actual.
3. Podar operadores: Se eliminan operadores de acuerdo con criterios globales. Estos criterios globales consistirán en criterios de consistencia generales que permiten descartar operadores que, aun siendo aplicables, se ve claramente que no mejorarán la solución (e.g.: no tiene sentido escoger el operador que deshaga el efecto del último operador aplicado).
4. Evaluar operadores: Se comparan los efectos de los operadores sobre la solución y se evalúa su resultado. Es en este punto donde interviene el conocimiento del experto para realizar la evaluación de los operadores. Si ninguno de los operadores es considerado adecuado se pasa a reconsiderar pasos anteriores.
5. Seleccionar un operador: Se escoge el operador mejor evaluado.
6. Aplicar el operador: Se aplica el operador al estado actual.

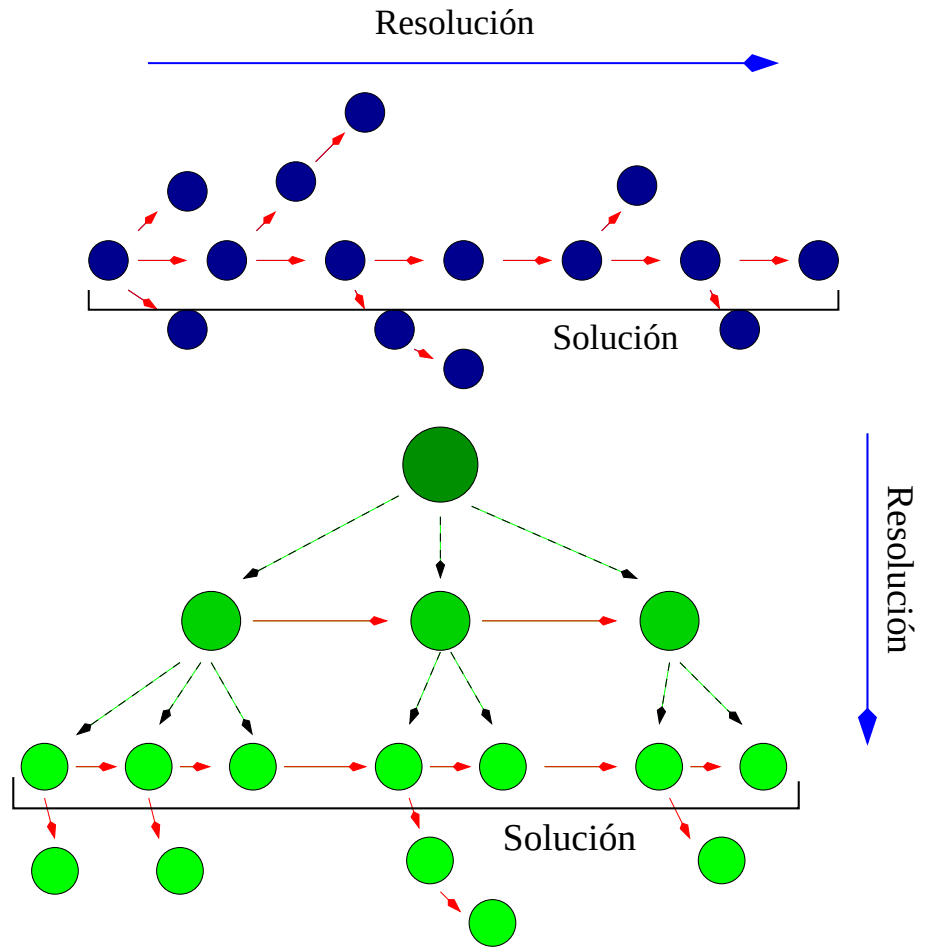


Figura 9.2: Resolución secuencial/resolución por descomposición de tareas

7. Evaluar el objetivo: Se para si se ha llegado al objetivo final o se reinicia el proceso.

La forma de plantear la solución del problema puede ser esencial para la eficiencia del proceso de resolución. El planteamiento más sencillo es definir el problema como una búsqueda en el espacio de soluciones parciales construyendo la solución paso a paso (secuencialmente). Esta aproximación puede ser viable si se dispone de un conocimiento exhaustivo sobre como evaluar cada uno de los pasos de la resolución y se puede dirigir la exploración rápidamente hacia el camino solución.

Esta aproximación puede ser poco eficiente si el conocimiento para la evaluación de los pasos de resolución no es suficientemente buena. Una alternativa es plantear el problema como una descomposición jerárquica de tareas. El conjunto de operadores permite dividir el problema inicial en problemas de complejidad decreciente hasta llegar al nivel de operaciones primitivas. Esto significa que el conjunto de operadores que permiten solucionar el problema no se ha de limitar a las acciones primitivas, sino que debe incluir aquellos que permiten hacer la descomposición del problema. Evidentemente esto aumenta el trabajo de adquisición del conocimiento, pero redunda en un aumento considerable de la eficiencia.

Mínimo compromiso

Un planteamiento alternativo consiste en partir de soluciones completas no óptimas e ir mejorándolas hasta llegar a una solución mejor, aunque veces se puede partir de una no solución y corregirla hasta llegar al espacio de soluciones. La búsqueda la realizaríamos en el espacio de soluciones completas. Para poder aplicar esta estrategia el problema nos debería permitir hallar fácilmente una

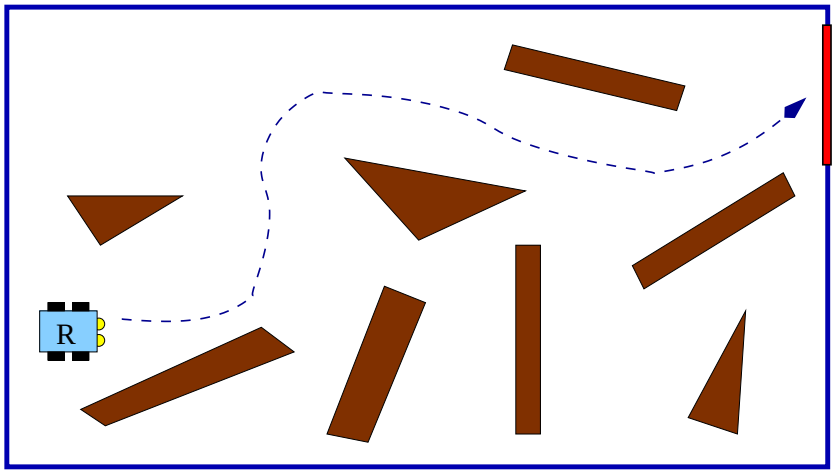


Figura 9.3: Planificación de la trayectoria de un robot

solución inicial completa. La elección del operador a aplicar en cada paso de la resolución la define la estrategia de mínimo compromiso: mínima modificación que imponga menos restricciones futuras.

La estrategia de resolución sería la siguiente:

1. Partir de una solución inicial no óptima, usualmente que satisface las restricciones.
2. Hacer una modificación sobre la solución. Esta modificación ha de hacerse de acuerdo con la heurística de mínimo compromiso, es decir, escoger la acción que menos restricciones imponga sobre la solución y, por lo tanto, menos restricciones imponga sobre el próximo paso.
3. Si la modificación viola alguna de las restricciones, se intenta deshacer alguno de los pasos anteriores, procurando que las modificaciones sean las mínimas. Esta modificación no tiene por que ser precisamente deshacer el último paso que se realizó.

El conocimiento del experto ha de aparecer en la evaluación de los efectos de los operadores sobre las restricciones, de manera que se pueda escoger siempre el operador con menos efecto sobre éstas y que permita más libertad de movimientos.

Aplicación de la resolución constructiva

Queremos planificar la mejor trayectoria de un robot en una habitación de manera que de un punto de salida llegue a la puerta de la habitación sin colisionar sin ninguno de los obstáculos que hay en la habitación. Supondremos que el robot puede ver lo que tiene delante de él y con ello calcular la distancia a los objetos que tiene a su alrededor.

Disponemos de un conjunto de operadores que le permiten:

- Moverse hacia adelante o hacia atrás cierta distancia a cierta velocidad
- Girar cierto número de grados

En este caso las restricciones globales son que se debe llegar a la puerta de salida y que el recorrido de la trayectoria y el tiempo que se tarde debe ser el menor posible. Las restricciones que se aplican a la elección de los operadores serán que el robot no choque con ninguno de los obstáculos o la pared, esto incluye que la distancia para pasar entre dos obstáculos debe ser la adecuada para que el robot pueda maniobrar y no se quede atascado.

La evaluación de la bondad de los operadores dependerá de cada movimiento. El operador mover será mejor cuando su aplicación nos acerque más al objetivo y más rápidamente. El operador girar

será mejor cuando la trayectoria en que nos deje el giro esté más libre y por lo tanto más lejos tenga los obstáculos más próximos.

Con estos operadores y restricciones, podemos resolver el problema utilizando los pasos de resolución de la estrategia proponer y aplicar para encontrar la ruta.

10. Razonamiento aproximado e incertidumbre

10.1 Incertidumbre y falta de información

Por lo general, el conocimiento que se debe manejar dentro de la mayoría de los dominios tratados por los sistemas basados en el conocimiento (SBC) no es de naturaleza exacta. En la práctica nos encontramos con problemas como:

- Representar el conocimiento para cubrir todos los hechos que son relevantes para un problema es difícil
- Existen dominios en los que se desconocen todos los hechos y reglas necesarias para resolver el problema
- Existen problemas en los que aún teniendo las reglas para resolverlos no disponemos de toda la información necesaria para aplicarlas

Esto significa que para poder razonar dentro de estos sistemas tendremos que utilizar herramientas más potentes que las que nos brinda la lógica clásica, que sólo nos permitiría trabajar con conocimiento del que pudiéramos establecer de manera efectiva su veracidad o falsedad.

De hecho, este objetivo no es descabellado ya que podemos observar como toda persona esta acostumbrada a tomar decisiones ante información incompleta o imprecisa (Invertimos en bolsa, diagnosticamos enfermedades, ...) y esa imprecisión o falta de conocimiento no impide la toma de decisiones. Esta claro que si deseamos que los SBC emulen la capacidad de los expertos hemos de dotarlos de mecanismos que sean capaces de abordar este problema.

La imprecisión o la falta de certeza en la información proviene de muchas fuentes, de entre ellas podemos citar:

1. *Incompletitud de los datos* debida a la no disponibilidad de éstos.
2. *Incertidumbre de los datos* debida a las limitaciones de los aparatos de medida, o a apreciaciones subjetivas del observador.
3. *Incertidumbre en las asociaciones* realizadas entre datos y conclusiones.
4. *Imprecisión en el lenguaje de descripción* debida al uso del lenguaje natural, ya que se presta a ambigüedades y malas interpretaciones.

El tratar con este problema ha llevado a desarrollar un conjunto de lógicas y modelos que intentan tratar el problema de la incompletitud e imprecisión del conocimiento desde diferentes perspectivas y modelizar de esta manera los procesos de razonamiento que aplican las personas. Muchas son las propuestas que se han desarrollado a lo largo de la evolución de los SBC, nos centraremos únicamente en dos formalismos que provienen de dos visiones distintas de la incertidumbre:

- Modelo probabilista (Redes Bayesianas)
- Modelo posibilista (Lógica difusa)

11. Modelo Probabilista

11.1 Introducción

Los modelos probabilistas se fundamentan en la teoría de la probabilidad. Las probabilidades se utilizan para modelizar nuestra creencia sobre la veracidad o falsedad de los hechos, de manera que podamos asignar valores de probabilidad a los diferentes hechos con los que tratamos y utilizar esas probabilidades para razonar sobre su certidumbre.

Cada hecho tendrá una probabilidad asociada de por si, o derivada de la probabilidad de aparición de otros hechos. Estas probabilidades serán las que nos permitirán tomar decisiones. Esta toma de decisiones no es estática, la probabilidad de un hecho podrá ser modificada por la observación y la modificación de la creencia en otros hechos que estén relacionados.

Podemos por ejemplo suponer que el hecho de decidir llevar o no un paraguas al salir de casa por la mañana puede verse afectada por multiples circunstancias. En principio tendremos una decisión a priori que dependerá del clima donde nos encontremos. Si estamos en una zona donde no llueve apenas nuesta decisión por omisión será no cogerlo.

Si por ejemplo vemos la predicción del tiempo el dia anterior esta decisión puede variar dependiendo de lo que nos cuenten, si nos dicen que al dia siguiente habrá nubosidad la decisión se inclinará más hacia coger el paraguas. Si además obervamos al dia siguiente que el suelo está mojado esta decisión se reforzará aún mas.

11.2 Teoría de probabilidades

Antes de comenzar a hablar de como modelizar el razonamiento mediante probabilidades, debemos repasar algunos conceptos básicos de la teoría de probabilidades.

El elemento fundamental de teoría de probabilidades es la **variable aleatoria**. Una variable aleatoria tiene un dominio de valores (valores posibles que puede tomar y sobre los que establecemos una distribución de probabilidad), podemos tener variables aleatorias **booleanas**, **discretas** o **continuas**.

Para poder trasladar la teoría de la probabilidad a un sistema basado en el conocimiento, deberemos crear una relación entre la representación del conocimiento que utilizamos y los elementos sobre los que establecemos las distribuciones de probabilidad.

En la práctica, toda representación del conocimiento que utilizamos se fundamenta en la lógica, de manera que la utilizaremos como lenguaje representación y utilizaremos las fórmulas lógicas como elemento básico. De esta forma, definiremos una **proposición lógica** como cualquier fórmula en lógica de enunciados o predicados, siendo éstas elementos primitivos de nuestra representación. Una proposición lógica tendrá asociada una variable aleatoria que indicará nuestro grado de creencia en ella.

Una variable aleatoria tendrá asociada una **distribución de probabilidad**. La forma de expresar esta distribución de probabilidad dependerá del tipo de variable aleatoria (Discretas: Binomial, Multinomial, ...; Continuas: Normal, χ^2, ...). El elegir un tipo de variable aleatoria u otro depende de como creamos que la información correspondiente a la proposición lógica debe modelarse. Para simplificar, sólo trabajaremos con variables aleatorias discretas, de manera que toda proposición lógica

tendrá un conjunto enumerado de posibles respuestas.

En cualquier problema, tendremos distintas proposiciones lógicas que intervendrán en una decisión, por lo tanto, tendremos que describir como son las variables aleatorias que describen estas proposiciones y como se debe calcular su influencia sobre las deducciones que permiten realizar las proposiciones.

La unión de variables aleatorias se puede describir mediante una **distribución de probabilidad conjunta**. Este será el mecanismo que nos va a permitir describir como se puede razonar mediante probabilidades.

Denotaremos como $P(a)$ la probabilidad de que la proposición A tenga el valor a. Por ejemplo, la proposición $Fumar$ puede tener los valores $\{fumar, \neg fumar\}$, $P(\neg fumar)$ es la probabilidad de la proposición $Fumar = \neg fumar$. Denotaremos como $P(A)$ al **vector de probabilidades** de todos los posibles valores de la proposición A

Definiremos como **probabilidad a priori** ($P(a)$) asociada a una proposición como el grado de creencia en ella a falta de otra información. Del conjunto de proposiciones que tengamos, algunas no tienen por que estar influidas por otras, de estas dispondremos de una distribución de probabilidad a priori que representará la probabilidad de que tomen cualquiera de sus valores.

Definiremos como **probabilidad a posteriori** o **condicional** ($P(a|b)$) como el grado de creencia en una proposición tras la observación de proposiciones asociadas a ella. Esta probabilidad estará asociada a las proposiciones que se ven influidas por la observación de otras proposiciones, por lo que nuestra creencia en ellas variará según la observación de éstas.

La probabilidad a posteriori se puede definir a partir de probabilidades a priori como:

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

Esta fórmula se puede transformar en lo que denominaremos la **regla del producto**:

$$P(a \wedge b) = P(a|b)P(b) = P(b|a)P(a)$$

Podemos observar por esta regla que la teoría de la probabilidad no asigna a priori una dirección a la causalidad y que se puede calcular la influencia de una proposición en otra y viceversa.

11.3 Inferencia probabilística

El usar como base de un mecanismo de inferencia la teoría de la probabilidad, restringe las cosas que podemos creer y deducir al marco de los axiomas en los que se fundamenta la probabilidad. Estos axiomas son:

- Toda probabilidad está en el intervalo $[0, 1]$

$$0 \leq P(a) \leq 1$$

- La proposición *cierto* tiene probabilidad 1 y la proposición $falso$ tiene probabilidad 0

$$P(cierto) = 1 \quad P(falso) = 0$$

- La probabilidad de la disyunción se obtiene mediante la fórmula

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

Dadas estas reglas básicas, podemos establecer una serie de mecanismos de inferencia, como por ejemplo:

- **Marginalización:** Probabilidad de una proposición atómica con independencia de los valores del resto de proposiciones
$$P(Y) = \sum_z P(Y, z)$$
- **Probabilidades condicionadas:** Probabilidad de una proposición dados unos valores para algunas proposiciones e independiente del resto de proposiciones (a partir de la regla del producto)
$$P(X|e) = \alpha \sum_y P(X, e, y)$$
El valor α es un factor de normalización que corresponde a factores comunes que hacen que las probabilidades sumen 1.

Ejemplo 11.1 *Consideremos un problema en el que intervengan las proposiciones* $Fumador = \{fumador,$
$\neg fumador\}$, $Sexo = \{varon, mujer\}$, $Enfisema = \{enfisema, \neg enfisema\}$

La siguiente tabla nos describe las distribuciones de probabilidad conjunta de estas proposiciones

	$enfisema$		$\neg enfisema$	
	$varon$	$mujer$	$varon$	$mujer$
$fumador$	*0.2*	*0.1*	*0.05*	*0.05*
$\neg fumador$	*0.02*	*0.02*	*0.23*	*0.33*

A partir de ella podemos hacer ciertas inferencias probabilísticas respecto a la combinación de las diferentes proposiciones y su influencia entre ellas

$$\begin{aligned}
P(enfisema \wedge varon) &= 0.2 + 0.02 \\
P(fumador \vee mujer) &= 0.2 + 0.1 + 0.05 + 0.05 + 0.02 + 0.33 \\
P(Fumador|enfisema) &= \langle P(fumador, enfisema, varon) \\
&\quad + P(fumador, enfisema, mujer), \\
&\quad P(\neg fumador, enfisema, varon) \\
&\quad + P(\neg fumador, enfisema, mujer)\rangle \\
&= \alpha\langle 0.3, 0.04\rangle \\
&= \langle 0.88, 0.12\rangle
\end{aligned}$$

Para poder realizar todos estos procesos de inferencia se requiere almacenar y recorrer la distribución de probabilidad conjunta de todas las proposiciones. Esto supone un gasto en tiempo y espacio impracticable. Suponiendo proposiciones binarias el coste en espacio y tiempo es $O(2^n)$ siendo n el número de proposiciones.

Cualquier problema real tiene un número de proposiciones suficiente para hacer que estos mecanismos de inferencia no sean útiles por su coste computacional. Se hace pues necesario crear mecanismos que nos simplifiquen el coste del razonamiento

11.3.1 Probabilidades condicionadas y reglas de producción

Hasta ahora podría parecer que los métodos de inferencia probabilística son mecanismos alejados de los sistemas de producción, ya que no parece que existan los conceptos de premisas y conclusiones, ni de encadenamiento.

En el mundo probabilístico la realidad está compuesta por un conjunto de eventos (variables aleatorias) que están ligados a través de las distribuciones de probabilidad conjunta. Un suceso es la observación de una combinación de valores de esas variables.

Al utilizar la formula de probabilidades condicionadas estamos imponiendo una relación de causalidad entre las variables que componen esos sucesos, cuando escribimos:

$$P(X|e) = \alpha \sum_{y} P(X, e, y)$$

estamos preguntando como influyen el conjunto de variables conocidas e sobre el valor de verdad de la variable en la que estamos interesados X (¿es mas probable x o $\neg x$?). Asumimos que X es consecuencia de e.

La ventaja de la formalización probabilística es que esta fórmula nos permite evaluar todas las reglas posibles que podamos crear como combinación de todas las variables de nuestro problema.

Por ejemplo, supongamos que en un problema intervienen un conjunto de variables $\mathcal{V} = \{A, B, C, D, E, F\}$, todas las variables están ligadas mediante una distribución de probabilidad conjunta $P(A, B, C, D, E, F)$.

En el formalismo de reglas de producción si consideramos que F se ve influida por los valores del resto de variables escribiríamos reglas como por ejemplo:

$$A \wedge \neg B \wedge C \wedge \neg D \wedge E \rightarrow F$$
$$\neg A \wedge B \wedge \neg C \rightarrow \neg F$$

Pero si nos fijamos en el significado de la distribución de probabilidad conjunta, ésta precisamente nos representa todas las posibles reglas que podemos construir con esas variables. Esta distribución nos permite calcular la influencia que tiene cualquier subconjunto de variables respecto al valor de verdad de cualquier otro subconjunto. La forma de calcularla es aplicar fórmula de probabilidades condicionadas.

Evidentemente, supone una ventaja poder codificar todas estas reglas como una distribución de probabilidad y poseer un mecanismo de razonamiento simple.

Al ser este planteamiento exhaustivo (representamos todas las reglas posibles) el formalismo probabilístico también nos permite realizar inferencia ante la falta de premisas (en nuestro caso serán variables ocultas). La distribución de probabilidad conjunta y la formula de probabilidades condicionadas nos permiten incluir la influencia de estas variables a pesar de que no conozcamos su valor.

Los problemas que tendremos serán obtener esa distribución de probabilidad conjunta y que el mecanismo de inferencia es computacionalmente prohibitivo tal como está planteado hasta ahora.

11.4 Independencia probabilística y la regla de Bayes

Por lo general, no todas las proposiciones que aparecen en un problema están relacionadas entre si. De hecho para cada proposición dependiente podemos identificar sólo un subconjunto de proposiciones que las influyen, siendo el resto irrelevantes para la inferencia de sus probabilidades. Llamaremos a esta propiedad **independencia probabilística**

Suponiendo que dos proposiciones X e Y no se influyen entre si, podemos reescribir sus probabilidades como:

$$P(X|Y) = P(X); \quad P(Y|X) = P(Y); \quad P(X, Y) = P(X)P(Y)$$

Dadas estas propiedades podremos reescribir las probabilidades conjuntas de manera más compacta reduciendo la complejidad

Anteriormente hemos enunciado la regla del producto como:

$$P(X,Y) = P(X|Y)P(Y) = P(Y|X)P(X)$$

Esta regla nos lleva a lo que denominaremos la **regla de Bayes**

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Esta regla y la propiedad de independencia serán el fundamento del razonamiento probabilístico y nos permitirá relacionar las probabilidades de unas evidencias con otras.

Suponiendo que podemos estimar exhaustivamente todas las probabilidades que involucran la variable Y podemos prescindir de las probabilidades a priori de la variable X y reescribir la formula de Bayes como:

$$P(Y|X) = \alpha P(X|Y)P(Y)$$

Esto es así porque las probabilidades $P(Y = y_1|X) \dots P(Y = y_n|X)$ han de sumar uno, α será un factor de normalización.

Suponiendo independencia condicional entre dos variables X e Y podremos escribir la probabilidad condicional de otra variable Z respecto a estas como:

$$P(X,Y|Z) = P(X|Z)P(Y|Z)$$

De manera que si substituimos en la regla de Bayes:

$$P(Z|X,Y) = \alpha P(X|Z)P(Y|Z)P(Z)$$

11.5 Redes Bayesianas

Si determinamos la independencia entre variables podemos simplificar el cálculo de la combinación de sus probabilidades y su representación, de manera que podremos razonar sobre la influencia de las probabilidades de unas proposiciones lógicas sobre otras de una manera más eficiente

Las **redes bayesianas** son un formalismo que permite la representación de las relaciones de independencia entre un conjunto de variables aleatorias. Una red bayesiana es un **grafo dirigido acíclico** que contiene información probabilística en sus nodos, indicando cual es la influencia que tienen sobre un nodo X_i sus padres en el grafo ($P(X_i|padres(X_i))$). El significado intuitivo de un enlace entre dos nodos X e Y es que la variable X tiene influencia directa sobre Y.

En cada uno de los nodos de la red aparece la distribución de probabilidad del nodo respecto a sus padres, es decir, como estos influyen la probabilidad del hijo. Esta forma de representar las influencias entre variables permite factorizar la distribución de probabilidad conjunta, convirtiéndose en el producto de probabilidades condicionales independientes:

$$P(x_1, x_2, \dots, x_n) = \prod_{i=1}^{n} P(x_i|padres(x_i))$$

El conjunto de probabilidades representadas en la red bayesiana describe la distribución de probabilidad conjunta de todas las variables, esto hace que no sea necesaria una tabla completa que describa la influencia entre todas ellas.

Intuitivamente podríamos decir que con la red bayesiana estamos añadiendo suposiciones adicionales al modelo probabilístico puro (todo esta relacionado con todo) estableciendo que solamente algunas influencias son posibles y que existe una dirección específica para la causalidad, que es la que indican los arcos la red.

Las propiedades que tienen las redes bayesianas nos dan ciertas ideas sobre como debemos construirlas a partir de un conjunto de proposiciones. Si consideramos que (regla del producto):

$$P(x_1, x_2, \ldots, x_n) = P(x_n|x_{n-1}, \ldots, x_1)P(x_{n-1}, \ldots, x_1)$$

Iterando el proceso tenemos que:

$$\begin{aligned} P(x_1, \ldots, x_n) &= P(x_n|x_{n-1}, \ldots, x_1)P(x_{n-1}|x_{n-2}, \ldots, x_1) \\ &\quad \cdots P(x_2|x_1)P(x_1) \\ &= \prod_{i=1}^{n} P(x_i|x_{i-1}, \ldots, x_1) \end{aligned}$$

Esta es la llamada **regla de la cadena**

Dadas estas propiedades, podemos afirmar que si $padres(X_i) \subseteq \{X_{i-1}, \ldots, X_1\}$, entonces:

$$P(X_i|X_{i-1}, \ldots, X_1) = P(X_i|padres(X_i))$$

Esto quiere decir que una red bayesiana es una representación correcta de un dominio sólo si cada nodo es condicionalmente independiente de sus predecesores en orden, dados sus padres.

Para lograr esto, se han de escoger como padres de una variable X_i aquellas de entre las variables $X_1, \ldots X_{i-1}$ que influyan directamente en X_i. Es decir, para describir la influencia que recibe una proposición del resto de proposiciones de las que disponemos, sólo es necesario utilizar las que influyen más directamente. La influencia del resto de proposiciones (si es que existe) estará descrita por las relaciones que puedan tener estas con los padres inmediatos de la proposición.

El utilizar una red bayesiana como representación de la distribución de probabilidad conjunta de un grupo de proposiciones supone una gran reducción en coste espacial. Como comentamos, el coste de representar la distribución de probabilidad conjunta de n variables binarias es $O(2^n)$. La representación de redes bayesianas nos permite una representación mas compacta gracias a la factorización de la distribución conjunta. Suponiendo que cada nodo de la red tenga como máximo k padres ($k \ll n$), un nodo necesitará 2^k para representar la influencia de sus padres, por lo tanto el espacio necesario será $O(n2^k)$. Por ejemplo, con 10 variables y suponiendo 3 padres como máximo tenemos 80 frente a 1024, con 100 variables y suponiendo 5 padres tenemos 3200 frente a aproximadamente 10^{30}

Ejemplo 11.2 *La red bayesiana de la figura 11.1 muestra las relaciones de dependencia entre un conjunto de proposiciones lógicas y la distribución de probabilidad que sigue cada una de esas influencias.*

A partir de la red podemos calcular la probabilidad de una proposición lógica utilizando las relaciones de dependencia entre las variables

$$\begin{aligned} &P(Infarto = si \wedge Presion = alta \wedge Fumador = si \\ &\wedge Deporte = si \wedge Alimentacion = equil) \\ &= \\ &P(Infarto = si|Presion = alta, Fumador = si) \\ &P(Presion = alta|Deporte = si, Alimentacion = equil) \\ &P(Fumador = si)P(Deporte = si)P(Alimentacion = equil) \\ &= 0.8 \times 0.01 \times 0.4 \times 0.1 \times 0.4 \\ &= 0.000128 \end{aligned}$$

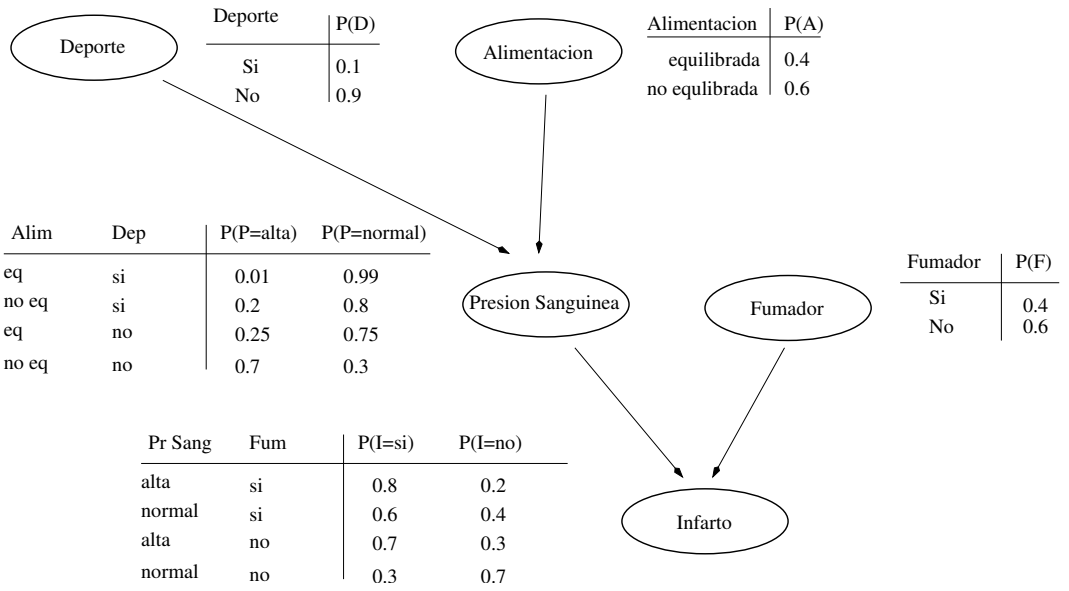


Figura 11.1: Red bayesiana que describe la probabilidad de sufrir un infarto

11.6 Inferencia probabilística mediante redes bayesianas

El objetivo de la inferencia probabilística es calcular la distribución de probabilidad a posteriori de un conjunto de variables dada la observación de un evento (valores observados para un subconjunto de variables).

Denotaremos como X la variable sobre la que queremos conocer la distribución, $\mathbf{E}$ será el conjunto de variables de las que conocemos su valor $\{E_1, \ldots, E_n\}$, e $\mathbf{Y}$ será el conjunto de variables que no hemos observado $\{Y_1, \ldots, Y_n\}$ (variables ocultas). De esta manera $\mathbf{X} = \{X\} \cup \mathbf{E} \cup \mathbf{Y}$ será el conjunto completo de variables. Nos plantearemos el cálculo de $P(X|\mathbf{e})$, es decir la distribución de probabilidad de los valores de X a partir de la influencia de los valores observados de las variables de $\mathbf{E}$.

Nosotros nos plantearemos lo que denominaremos la inferencia exacta, que es la que se realiza utilizando directamente la distribución de probabilidad que describe la red bayesiana. Como veremos mas adelante ésta sólo es tratable computacionalmente si la topología de la red tiene ciertas propiedades.

11.6.1 Inferencia por enumeración

El primer algoritmo de inferencia exacta que veremos es el denominado de **Inferencia por enumeración**. Éste se basa en que cualquier probabilidad condicionada se puede calcular como la suma de todos los posibles casos a partir de la distribución de probabilidad conjunta.

$$P(X|\mathbf{e}) = \alpha P(X, \mathbf{e}) = \alpha \sum_{y} P(X, \mathbf{e}, \mathbf{y})$$

La red bayesiana nos permite factorizar la distribución de probabilidad conjunta y obtener una expresión mas fácil de evaluar.

Eso no quita que el número de operaciones crezca de manera exponencial con el número de variables que tiene la expresión, además de realizar parte de las operaciones múltiples veces dependiendo de la estructura de la red. Por esta causa este método no es utilizado en la práctica.

Ejemplo 11.3 *Usando la red bayesiana ejemplo podemos calcular la probabilidad de ser*

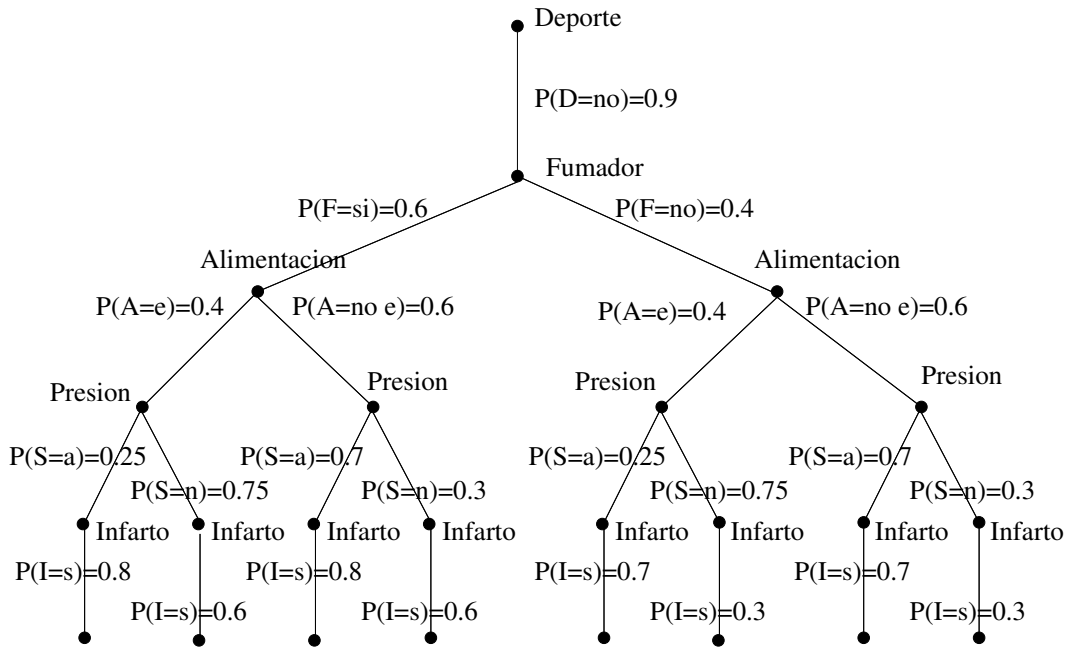


Figura 11.2: Árbol de enumeración de probabilidades

fumador si se ha tenido un infarto y no se hace deporte

$$P(Fumador|Infarto = si, Deporte = no)$$

La distribución de probabilidad conjunta de la red sería:

$$P(D, A, S, F, I) = P(I|S, F)P(F)P(S|D, A)P(D)P(A)$$

Debemos calcular $P(F|I = si, D = no)$*, por lo tanto tenemos*

$$\begin{aligned}
&P(F|I = s, D = n) = \alpha P(F, I = s, D = n) \\
&= \alpha \sum_{A\in\{e,\neg e\}} \sum_{S\in\{a,n\}} P(D = n, A, S, F, I = s) \\
&= \alpha P(D = n)P(F) \sum_{A\in\{e,\neg e\}} P(A) \sum_{S\in\{a,n\}} P(S|D = n, A)P(I = s|S, F)
\end{aligned}$$

Si enumeramos todas las posibilidades y las sumamos de acuerdo con la distribución de probabilidad conjunta tenemos que:

$$\begin{aligned}
&P(Fumador|Infarto = si, Deporte = no) \\
&= \alpha\langle 0.9 \cdot 0.4 \cdot (0.4 \cdot (0.25 \cdot 0.8 + 0.75 \cdot 0,6) + 0.6 \cdot (0.7 \cdot 0.8 + 0.3 \cdot 0.6)), \\
&\qquad 0.9 \cdot 0.6 \cdot (0.4 \cdot (0.25 \cdot 0.7 + 0,75 \cdot 0.3) + 0.6 \cdot (0.7 \cdot 0.7 + 0.3 \cdot 0.3))\rangle \\
&= \alpha\langle 0.253, 0, 274\rangle \\
&= \langle 0, 48, 0.52\rangle
\end{aligned}$$

Podemos ver las operaciones que se realizan en el árbol de probabilidades que se calcula (figura 11.2). Cada una de las ramas del árbol corresponde a cada uno de los eventos posibles.

Algoritmo 11.1 Algoritmo de eliminación de variables

```
Function: Elimination de Variables (X, e, rb)
factores ← []
vars ← REVERSE(VARS(rb))
foreach var ∈ vars do
    factores ← concatena(factores,CALCULA-FACTOR(var,e))
    if var es variable oculta then
        factores ← PRODUCTO-Y-SUMA(var,factores)
    end
end
return NORMALIZA(PRODUCTO(factores))
```

11.6.2 Algoritmo de eliminación de variables

La inferencia por enumeración puede ser bastante ineficiente dependiendo de la estructura de la red y dar lugar a muchos cálculos repetidos, por lo que se han intentado hacer algoritmos más eficientes. El **algoritmo de eliminación de variables** intenta evitar esta repetición de cálculos. El algoritmo utiliza técnicas de programación dinámica (memorización) de manera que se guardan cálculos intermedios para cada variable para poder reutilizarlos. A estos cálculos intermedios los denominaremos **factores**

El cálculo de la probabilidad se realiza evaluando la expresión de la distribución de probabilidad conjunta de izquierda a derecha, aprovechando ese orden para obtener los factores. Esta estrategia hace que los cálculos que impliquen una variable se realicen una sola vez. Los *factores* correspondientes a cada variable se van acumulando y utilizándose según se necesitan.

Una ventaja adicional de este algoritmo es que las variables no relevantes desaparecen al ser factores constantes en las operaciones y por lo tanto permite eliminarlas del cálculo (de ahí el nombre de algoritmo de eliminación de variables).

Su implementación se puede ver en el algoritmo 11.1. CALCULA_FACTOR genera el factor correspondiente a la variable en la función de distribución de probabilidad conjunta, PRODUCTO_Y_SUMA multiplica los factores y suma respecto a la variable oculta, PRODUCTO multiplica un conjunto de factores.

Un factor corresponde a la probabilidad de un conjunto de variables dadas las variables ocultas. Se representa por una tabla que para cada combinación de variables ocultas da la probabilidad de las variables del factor, por ejemplo:

$f_X(Y,Z)=$

Y	Z	
C	C	0.2
C	F	0.4
F	C	0.8
F	F	0.6

Los factores tienen dos operaciones, la suma y producto de factores.

La suma se aplica a un factor y sobre una variable oculta del factor. Como resultado obtenemos una matriz reducida en la que las filas del mismo valor se han acumulado, por ejemplo

$f_{X\overline{Z}}(Y) = \sum_Z f_X(Y,Z)=$

Y	
C	0.6
F	1.4

Es igual que una operación de agregación sobre una columna en bases de datos

El producto de factores permite juntar varios factores entre ellos utilizando las variables ocultas comunes, por ejemplo:

$f_{X_1X_2}(Y,W,Z) = f_{X_1}(Y,Z) \times f_{X_2}(Z,W)=$

Y	Z		Z	W		Y	Z	W	
C	C	0.2	C	C	0.3	C	C	C	0.2×0.3
C	F	0.8	C	F	0.7	C	C	F	0.2×0.7
F	C	0.4	F	C	0.1	C	F	C	0.8×0.1
F	F	0.6	F	F	0.9	C	F	F	0.8×0.9
						F	C	C	0.4×0.3
						F	C	F	0.4×0.7
						F	F	C	0.6×0.1
						F	F	F	0.6×0.9

Es igual que una operación de join en una base de datos multiplicando los valores de las columnas de datos.

Ejemplo 11.4 *Volveremos a calcular $P(Fumador|Infarto = si, Deporte = no)$ a partir de la distribución de probabilidad conjunta:*

$$P(D,A,S,F,I) = P(I|S,F)P(F)P(S|D,A)P(D)P(A)$$

Debemos calcular $P(F|I = si, D = no)$, por lo tanto tenemos

$$\begin{aligned} &P(F|I = s, D = n) = \alpha P(I = s, F, D = n) \\ &= \alpha \sum_{A\in\{e,\neg e\}} \sum_{S\in\{a,n\}} P(D = n, A, S, F, I = s) \end{aligned}$$

En esta ocasión no sacamos factores comunes para seguir el algoritmo

$$\alpha P(D = n) \sum_{A\in\{e,\neg e\}} P(A) \sum_{S\in\{a,n\}} P(S|D = n, A)P(F)P(I = s|S,F)$$

El algoritmo empieza calculando el factor para la variable Infarto ($P(I = s|S,F)$), esta tiene fijo su valor a `si`*, depende de las variables Presión Sanguinea y Fumador*

$f_I(S,F)=$

S	F	
a	s	*0.8*
a	n	*0.7*
n	s	*0.6*
n	n	*0.3*

La variable fumador ($P(F)$) no depende de ninguna otra variable, al ser la variable que preguntamos el factor incluye todos los valores

$f_F(F)=$

F	
s	*0.4*
n	*0.6*

La variable Presión Sanguinea ($P(S|D = n, A)$), depende de las variable Deporte que tiene fijo su valor a `no` *y Alimentación. Esta es una variable oculta, por lo que se debe calcular para todos sus valores*

$f_S(S,A)=$

S	A	
a	e	0.25
a	$\neg e$	0.7
n	e	0.75
n	$\neg\ e$	0.3

Al ser la variable Presión Sanguinea una variable oculta debemos acumular todos los factores que hemos calculado

$$f_S(S,A) \times f_F(F) \times f_I(S,F)$$

$f_{FI}(S,F) = f_F(F) \times f_I(S,F)=$

S	F	
a	s	0.8×0.4
a	n	0.7×0.6
n	s	0.6×0.4
n	n	0.3×0.6

$f_{FIS}(S,F,A) = f_{FI}(S,F) \times f_S(S,A)=$

S	F	A	
a	s	e	$0.8\times 0.4\times 0.25$
a	s	$\neg e$	$0.8\times 0.4\times 0.7$
a	n	e	$0.7\times 0.6\times 0.25$
a	n	$\neg e$	$0.7\times 0.6\times 0.7$
n	s	e	$0.6\times 0.4\times 0.75$
n	s	$\neg e$	$0.6\times 0.4\times 0.3$
n	n	e	$0.3\times 0.6\times 0.75$
n	n	$\neg e$	$0.3\times 0.6\times 0.3$

Y ahora sumamos sobre todos los valores de la variable S para obtener el factor correspondiente a la variable Presión Sanguinea

$f_{FI\overline{S}}(F,A) = \sum_{S\in\{a,n\}} f_{FIS}(S,F,A) =$

F	A	
s	e	$0.8\times 0.4\times 0.25 + 0.6\times 0.4\times 0.75 = 0.26$
s	$\neg e$	$0.8\times 0.4\times 0.7 + 0.6\times 0.4\times 0.3 = 0.296$
n	e	$0.7\times 0.6\times 0.25 + 0.3\times 0.6\times 0.75 = 0.24$
n	$\neg e$	$0.7\times 0.6\times 0.7 + 0.3\times 0.6\times 0.3 = 0.348$

El factor de la variable Alimentación ($P(A)$) no depende de ninguna variable, al ser una variable oculta generamos todas las posibilidades

$f_A(A)=$

A	
e	0.4
$\neg e$	0.6

Ahora debemos acumular todos los factores calculados

$f_{AFI\overline{S}}(F,A) = f_A(A) \times f_{FI\overline{S}}(F,A)=$

F	A	
s	e	$0.26\times 0.4 = 0.104$
s	$\neg e$	$0.296\times 0.6 = 0.177$
n	e	$0.24\times 0.4 = 0.096$
n	$\neg e$	$0.348\times 0.6 = 0.208$

Y ahora sumamos sobre todos los valores de la variable A para obtener el factor correspondiente a la variable Alimentación

$$f_{\overline{A}FI\overline{S}}(F) = \sum_{A\in\{e,\neg e\}} f_{AFI\overline{S}}(F,A) = \begin{array}{c|l} F & \\ \hline S & 0.104 + 0.177 = 0.281 \\ n & 0.096 + 0.208 = 0.304 \end{array}$$

Y por último la variable Deporte ($P(D = n)$) tiene el valor fijado a `no` *y dado que no depende de la variable fumador se puede obviar, ya que es un factor constante.*

Ahora, si normalizamos a 1

$$P(F|I = s, D = n) = \begin{array}{c|l} F & \\ \hline S & 0.48 \\ n & 0.52 \end{array}$$

La complejidad del algoritmo de eliminación de variables depende del tamaño del mayor factor, que depende del orden en el que se evalúan las variables y la topología de la red. El orden de evaluación que escogeremos será el topológico según el grafo, pero podríamos utilizar cualquier orden. De hecho se podría escoger el orden que más variables eliminara para hacer que el cálculo sea más eficiente, el problema es que encontrar el orden óptimo es NP.

La complejidad de la inferencia exacta es NP-hard en el caso general. En el caso particular en que la red bayesiana cumple que para cada par de nodos hay un único camino no dirigido (**poliárbol**), entonces se puede calcular en tiempo lineal. Por eso es interesante que cuando se construya una red bayesiana para un problema se construyan poliárboles. Si podemos construir una red que cumpla esta propiedad podemos utilizar este algoritmo sin ningún problema.

Para obtener resultados en el caso general se recurre a algoritmos aproximados basados en técnicas de muestreo. Evidentemente el valor obtenido con estos algoritmos es una aproximación del valor real, pero el coste temporal es razonable.

12. Modelo Posibilista

12.1 Introducción

Un método alternativo para representar la imprecisión es el que presenta el modelo posibilista. Este modelo surge de la llamada lógica posibilista, esta es una lógica no clásica especialmente diseñada para el razonamiento con evidencias incompletas y conocimiento parcialmente inconsistente.

Muchas veces los expertos utilizan términos para hablar de los valores de las variables involucradas en su conocimiento en lugar de valores específicos. Estos términos no representan un valor concreto, sino un conjunto de valores que no tiene por que coincidir con la idea clásica de conjunto.

Habitualmente un conjunto tiene una definición suficiente y necesaria que permite establecer claramente la pertenencia de los diferentes elementos al conjunto. Pero podemos observar que el tipo de información que expresan los términos de los expertos permite que ciertos valores puedan pertenecer a diferentes conjuntos con diferente grado.

Por ejemplo, un experto puede referirse a la **temperatura** diciendo que es *alta*. La variable *temperatura* estará definida sobre un rango (por lo general continuo) de valores y el término *alta* no tiene por que tener unas fronteras definidas, ya que en un dominio real sería difícil pensar que a partir de un valor específico la temperatura se considera alta, pero que el valor inmediatamente anterior no lo es.

Podríamos decir que la expresión *la temperatura es alta* tiene diferentes grados de verdad dependiendo del valor exacto sobre el que la evaluáramos. Desde el punto de vista del experto esta interpretación es más intuitiva que la interpretación clásica y es la que usa habitualmente para describir ciertos dominios. Este tipo de imprecisión en el lenguaje es la que queremos modelar mediante la lógica posibilista.

De hecho este modelo intenta acercarse más a la forma que tiene el experto de modelar y expresar conocimiento sobre su dominio cuando la información de la que dispone no tiene por que evaluarse directamente a un valor concreto, por lo tanto tenemos no solo incertidumbre sobre la asociacion entre hechos y conclusiones, sino también sobre los propios hechos. Esto difiere a lo que hemos visto en el modelo probabilístico en el que para las evidencias que observamos siempre lo hacemos con total seguridad.

En este tipo de representación trabajaremos con variables que están definidas en un *universo del discurso* (conjunto de valores posibles) y que se definirán a partir de *etiquetas lingüísticas* (términos que podemos usar para referirnos a las variables).

12.2 Conjuntos difusos/Lógica difusa

La lógica posibilista se basa en la teoría de los conjuntos difusos y, por extensión, en la lógica difusa. Los conjuntos difusos se han tomado como punto de partida para la representación de la vaguedad del lenguaje, de esta manera, proposiciones como por ejemplo "*la temperatura de hoy es agradable*" supondría que existe un conjunto de temperaturas asignadas al término lingüístico *agradable* que correspondería a un conjunto difuso, y el razonamiento se haría basándose en éste. Esto permitiría utilizar estos conjuntos para representar los razonamientos cualitativos que suelen utilizar los expertos.

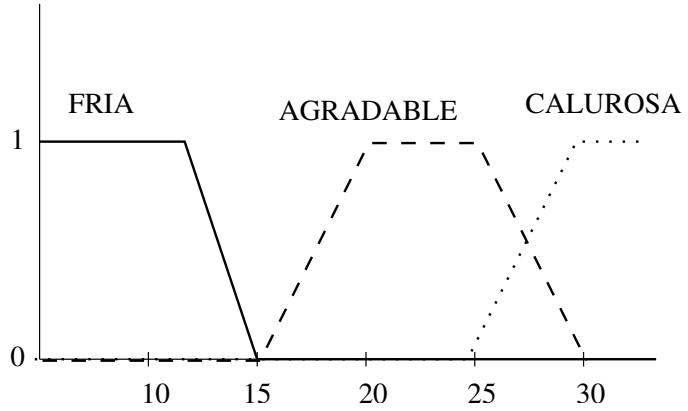


Figura 12.1: Etiquetas lingüísticas para la variable temperatura

Los conjuntos difusos son una generalización de los conjuntos clásicos en los que la pertenencia no está restringida a sí o no, sino que puede haber una gradación de pertenencia en el intervalo $[0, 1]$, determinada por una función de distribución asignada al conjunto.

Los hechos representables mediante conjuntos difusos siguen el esquema $[X$ es $A]$ donde X es una variable definida sobre un *universo* U de valores posibles y A es un *término lingüístico* aplicable a X, que restringe los valores que puede tomar. Estos valores corresponderán al conjunto difuso representado por A sobre el universo U. En el ejemplo anterior, X sería la *temperatura*, el universo U serían todos los posibles valores que puede tomar la temperatura y A sería *agradable*, este término indicaría los valores posibles para X. En la figura 12.1 se pueden ver diferentes etiquetas aplicables al concepto temperatura y sus conjuntos difusos correspondientes.

Todo conjunto difuso está representado por una función característica μ_A que define la pertenencia de un elemento al conjunto. A través de ésta se puede definir la función de posibilidad de los valores de X al conjunto A. De este modo, la posibilidad de que X tenga un valor $u \in U$ sabiendo que $[X\ es\ A]$ viene definida por la función:

$$\pi_A : U \to [0, 1]\ tal\ que\ \pi_A(u) = \mu_A(u)$$

Es decir, la posibilidad para un valor se corresponde con su grado de pertenencia al conjunto difuso representado por la etiqueta lingüística. Un valor de posibilidad se puede interpretar como el grado con el que una proposición es compatible con el conocimiento que se describe con la proposición $[X$ es $A]$.

12.3 Conectivas lógicas en lógica difusa

Al igual que se puede interpretar la representación de las proposiciones con una visión conjuntista, ésta se puede también transformar en una visión lógica[1], que es en definitiva lo que se va a utilizar en la deducción con las reglas de los expertos.

Se puede ver la pertenencia total o la no pertenencia de un valor a un conjunto difuso como los valores de verdad *cierto* y *falso* para una proposición (esta sería la noción clásica de valores de verdad), a ésto le añadimos el poder tener una gradación de valor de verdad entre estos valores extremos, determinada por la función de posibilidad del conjunto difuso.Con esto tenemos una lógica que permite cualquier valor de verdad entre cierto y falso.

[1]Los conjuntos difusos y la lógica difusa se pueden ver como una extensión continua de la teoría de conjuntos y la lógica clásica. Por lo tanto si la teoría de conjuntos se puede ver como una variante notacional de la lógica de predicados, su extensión continua también.

Siguiendo con esta visión de la lógica, podemos establecer las combinaciones que permiten las conectivas de la lógica clásica sobre sus dos valores de verdad se extiendan sobre la gradación de valores de verdad que permite la función de posibilidad de un conjunto difuso.

En lógica de proposiciones disponemos de tres conectivas para la combinación de enunciados: $\wedge$, $\vee$ y $\neg$. Sobre estas, la teoría de modelos establece tres tablas de verdad que indican como se combinan los únicos dos valores de verdad permitidos. La lógica difusa establece tres tipos de funciones equivalentes a estas conectivas que dan la extensión continua de estas combinaciones.

Conjunción difusa

La función que permite combinar en conjunción dos funciones de posibilidad es denominada T-norma y se define como $T(x, y) : [0, 1] \times [0, 1] \to [0, 1]$ y ha de satisfacer las propiedades:

1. Conmutativa, T(a,b)=T(b,a)
2. Asociativa, T(a,T(b,c))=T(T(a,b),c)
3. T(0,a)=0
4. T(1,a)=a
5. Es una función creciente, T(a,b)$\leq$T(c,d) si a$\leq$c y b$\leq$d

Se puede observar que son las mismas propiedades que cumple la conectiva $\wedge$ en lógica de proposiciones, lo mismo pasará con el resto de conectivas. Además, estas propiedades imponen que el comportamiento de la función en sus extremos sea el mismo que los valores de la tabla de verdad de la conectiva $\wedge$.

Ejemplos de funciones que cumplen estas propiedades son:

- $T(x, y) = \text{mín}(x, y)$
- $T(x, y) = x \cdot y$
- $T(x, y) = \text{máx}(0, x + y - 1)$

Se puede ver la forma de estas funciones en la figura 12.2.

Disyunción difusa

La función que permite combinar en disyunción dos funciones de posibilidad es denominada T-conorma y se define como $S(x, y) : [0, 1] \times [0, 1] \to [0, 1]$ y ha de satisfacer las propiedades:

1. Conmutativa, S(a,b)=S(b,a)
2. Asociativa, S(a,S(b,c))=S(S(a,b),c)
3. S(0,a)=a
4. S(1,a)=1
5. Es una función creciente, S(a,b)$\leq$S(c,d) si a$\leq$c y b$\leq$d

Ejemplos de funciones que cumplen estas propiedades son:

- $S(x, y) = \text{máx}(x, y)$

- $S(x, y) = x + y - x \cdot y$
- $S(x, y) = \text{mín}(x + y, 1)$

Se puede ver la forma de estas funciones en la figura 12.2.

Estas tres funciones de T-conorma, junto a las tres anteriores T-normas, corresponden a tres pares de funciones duales respecto a las leyes de DeMorgan y son las más utilizadas. El ser dual respecto a las leyes de DeMorgan es una característica importante ya que permite preservar propiedades de la lógica de proposiciones en el comportamiento de las conectivas difusas.

Negación difusa

La función que permite negar una función de posibilidad es denominada *negación fuerte* y se define como $N(x) : [0, 1] \to [0, 1]$ y ha de satisfacer las propiedades:

1. $N((N(a)) = a$
2. Es una función decreciente, $N(a) \geq N(b)$ si $a \leq b$

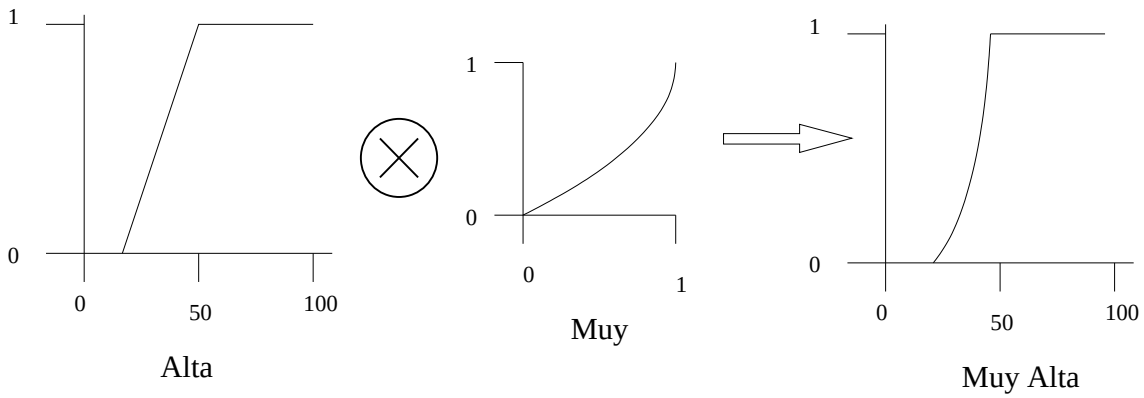
Ejemplos de funciones que cumplen estas propiedades son:

- $N(x) = 1 - x$
- $N(x) = \sqrt{1 - x^2}$
- $N_t(x) = \frac{1-x}{1+t \cdot x}$ $t > 1$

Se puede ver la forma de estas funciones en la figura 12.2.

Modificadores difusos

Algo que también nos permite la lógica difusa es añadir modificadores a los términos lingüísticos que utilizamos de la misma manera en que lo haría el experto. Por ejemplo, podríamos decir que *la temperatura es muy alta* y utilizar una función que represente la semántica del modificador *muy* como una función que se combina con la etiqueta, donde el conjunto difuso *muy alta* sería el resultado de aplicar el modificador **muy** al conjunto difuso *alta*. Los modificadores está definidos como funciones $[0, 1] \longrightarrow [0, 1]$ y varían el valor de verdad del conjunto difuso original.



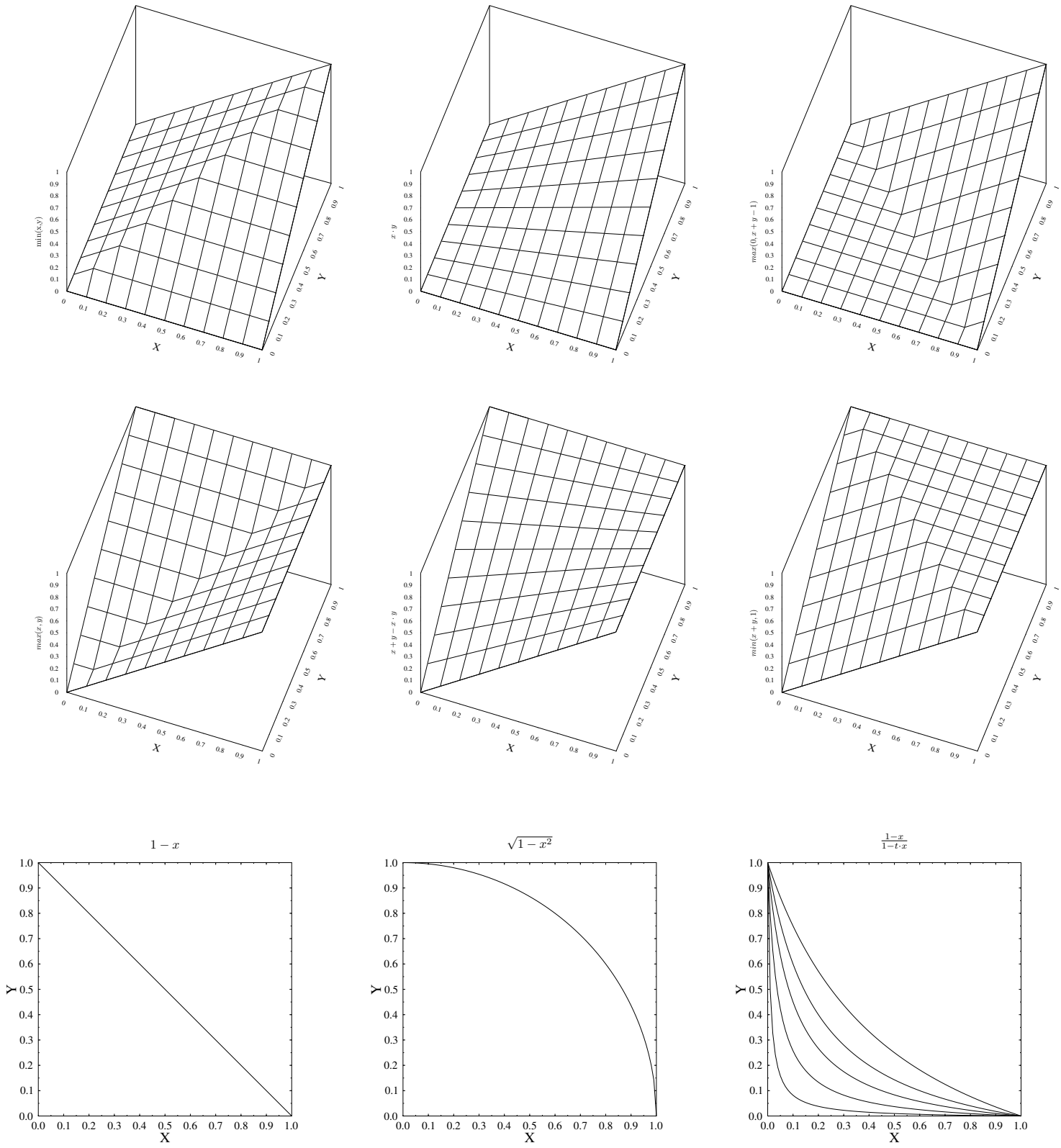


Figura 12.2: Funciones T-norma, T-conorma y negación

12.4 Condiciones difusas

Mediante las funciones de posibilidad podemos construir combinaciones de proposiciones que permiten crear condiciones a partir de proposiciones difusas simples. El conjunto difuso resultante de la condición dependerá de los conjuntos que estemos combinando y su función característica estará definida sobre uno o más universos del discurso.

De esta manera, si tenemos las proposiciones F=[X *es* A] y G=[X *es*B] y π_A y π_B están definidas sobre el mismo universo U, entonces:

$$F \wedge G = [X \text{ es } A \text{ y } B] \;\; con \;\; \pi_{F \wedge G}(u) = T(\pi_A(u), \pi_B(u))$$
$$F \vee G = [X \text{ es } A \text{ o } B] \;\; con \;\; \pi_{F \vee G}(u) = S(\pi_A(u), \pi_B(u))$$

Donde la T-norma y T-conorma se definen sobre una sola dimensión.

En cambio, si tenemos F=[X *es* A] y G=[Y *es* B] con π_A definida sobre U y π_B definida sobre V, con $U \neq V$, tenemos:

$$F \wedge G = [X \text{ es } A] \text{ y } [Y \text{ es } B] \;\; con \;\; \pi_{F \wedge G}(u, v) = T(\pi_A(u), \pi_B(v))$$
$$F \vee G = [X \text{ es } A] \text{ o } [Y \text{ es } B] \;\; con \;\; \pi_{F \vee G}(u, v) = S(\pi_A(u), \pi_B(v))$$

Donde la T-norma y T-conorma se definen sobre dos dimensiones.

Podemos ver que las funciones de combinación de funciones de posibilidad generan una nueva función de posibilidad. En definitiva, estamos definiendo un álgebra sobre funciones en las que las conectivas difusas nos sirven de operaciones de combinación.

Veamos dos ejemplos de combinación de proposiciones en ambos casos.

Ejemplo 12.1 *Supongamos que tenemos las proposiciones [{La temperatura de hoy} es {agradable}] y [{La temperatura de hoy} es {calurosa}], proposiciones definidas sobre el mismo universo U (el de las temperaturas posibles) donde las funciones de posibilidad de las etiquetas* agradable *y* calurosa *se pueden ver en la figura 12.3.*

Podemos utilizar como función de T-norma T(x,y)= $\min(x, y)$ *y de T-conorma S(x,y)=* $\max(x, y)$ *y construir la función de posibilidad que define la conjunción y la disyunción de ambas proposiciones, el resultado se puede ver en la figura 12.4.*

También podemos definir la negación de agradable *mediante la función de negación fuerte N(x)=*$1-x$*, el resultado se puede ver en la figura 12.5.*

Ejemplo 12.2 *Supongamos que tenemos las proposiciones [{Juan} es {alto}] y [{Juan} es {joven}], proposiciones definidas sobre dos universos diferentes, el de las alturas, y el de las edades. Las funciones de posibilidad de las etiquetas lingüísticas están representadas en la figura 12.6*

Podemos utilizar como función de T-norma T(x,y)= $\min(x, y)$ *y de T-conorma S(x,y)=* $\max(x, y)$ *y construir la función de posibilidad que define la conjunción y la disyunción de ambas proposiciones, en este caso, definidas sobre dos dimensiones, ya que las proposiciones corresponden a universos diferentes. La figura 12.7 y 12.8 muestran un mapa de niveles de las etiquetas resultantes y una representación tridimensional.*

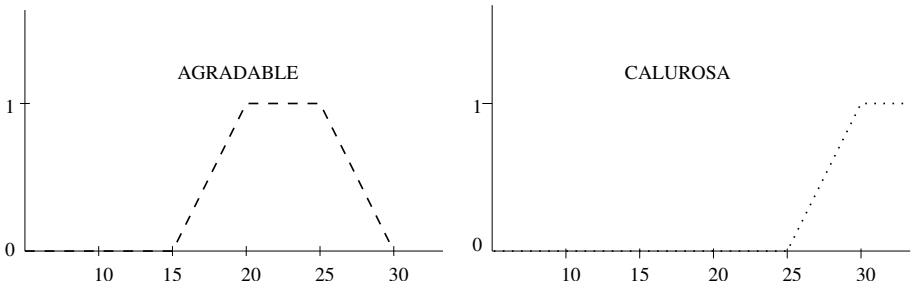


Figura 12.3: Conjuntos difusos agradable y calurosa

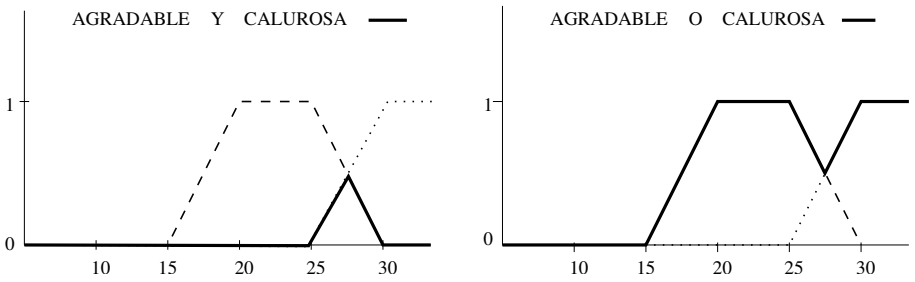


Figura 12.4: Combinación de las etiquetas agradable y caluroso (conjunción y disyunción)

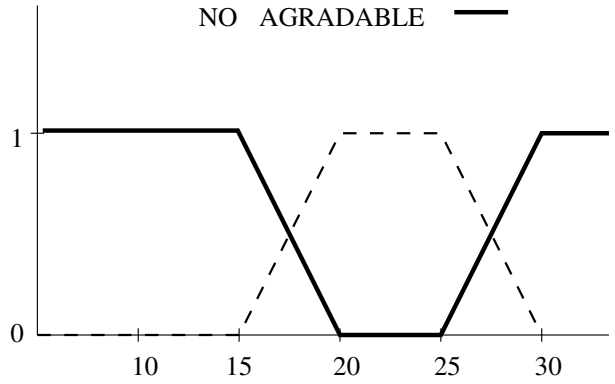


Figura 12.5: Negación de la etiqueta agradable

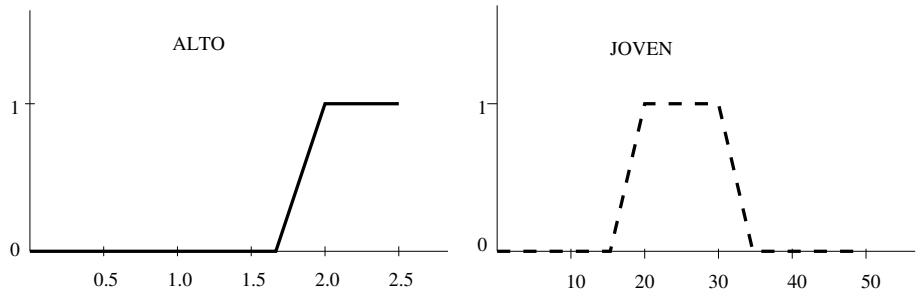


Figura 12.6: Etiquetas difusas alto y joven

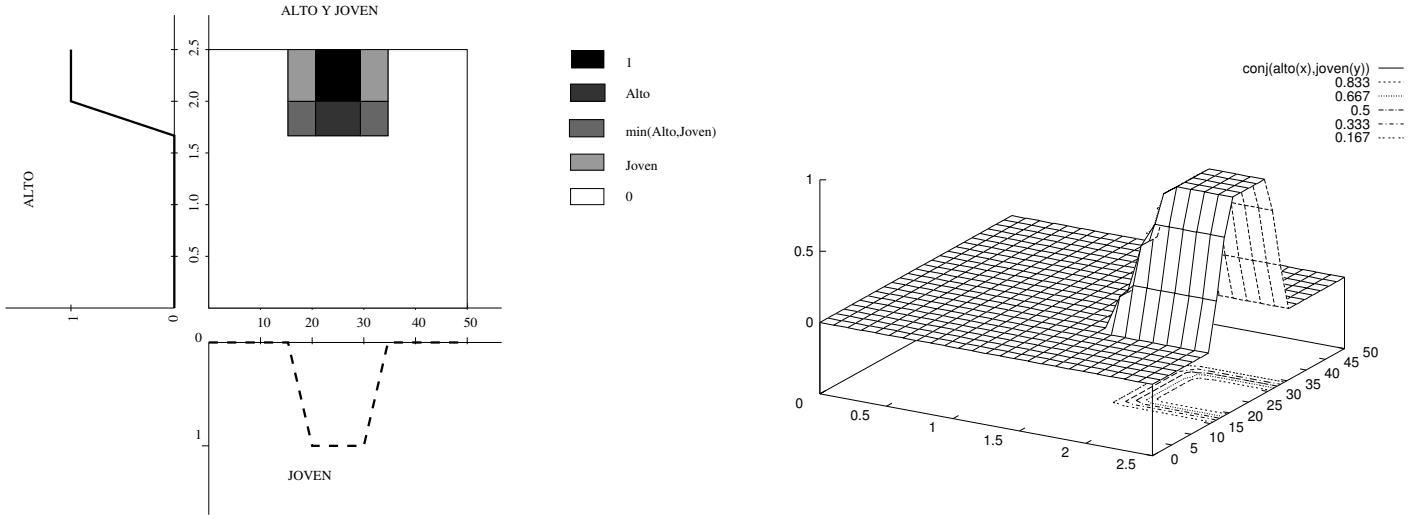


Figura 12.7: Representación de la combinación de las etiquetas alto y joven

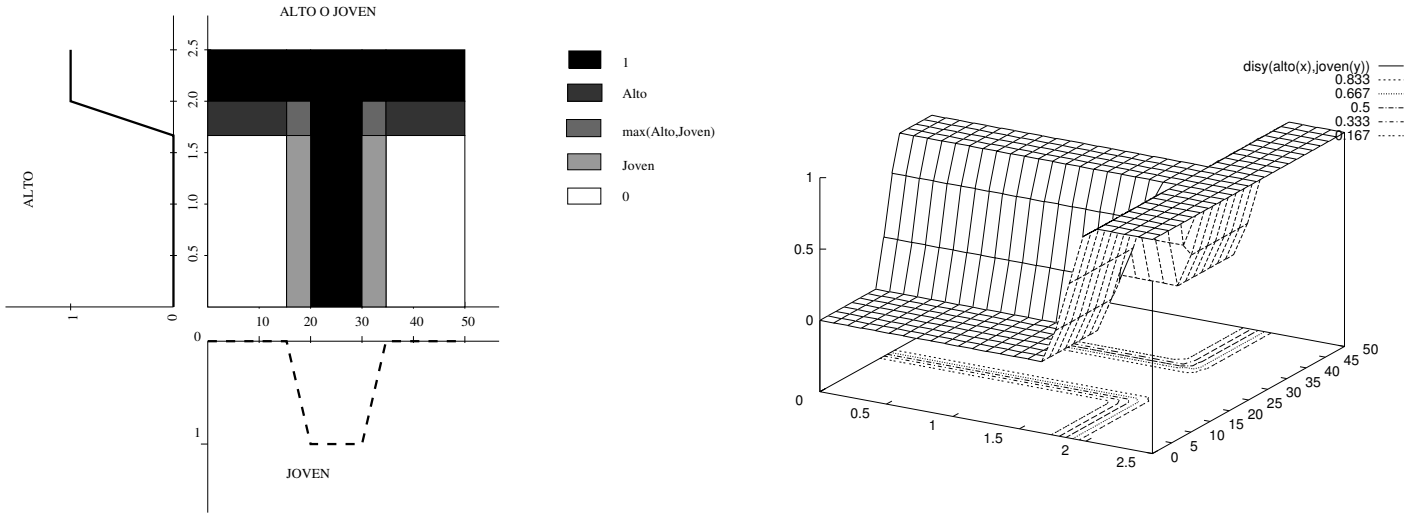


Figura 12.8: Representación de la combinación de las etiquetas alto o joven

12.5 Inferencia difusa con datos precisos

A partir del formalismo de la lógica difusa podemos establecer un modelo de razonamiento incluyendo el resto de operadores de la lógica clásica. El operador que nos permitirá un razonamiento equivalente al que obtenemos con los sistemas de reglas clásicos será el operador de implicación.

Con todos estos operadores podemos escribir reglas utilizando como elementos atómicos expresiones difusas, por ejemplo:

$$[\text{altura(X)=alto}] \wedge [\text{edad(X)=joven}] \rightarrow [\text{peso(X)=medio}]$$

En este escenario nos podemos plantear dos tipos de inferencias. Una primera posibilidad sería que nuestros hechos estén expresados también como hechos difusos, de manera que podríamos utilizar las reglas como el mecanismo de razonamiento habitual. El resultado del razonamiento sería un conjunto difuso que combinara adecuadamente los conjuntos difusos que provienen de los hechos, los antecedentes de las reglas y sus consecuentes. Para ello, se definen las funciones de combinación que corresponden a la implicación y a la regla de deducción del **modus ponens**[2].

Otra posibilidad es que dispongamos de valores precisos para las variables de nuestro dominio, nuestro objetivo será obtener el valor preciso que se obtiene a partir de las relaciones que nos indican las reglas difusas que tenemos.

Este segundo caso es mas sencillo, supone que tenemos una variable respuesta y un conjunto de variables que influyen directamente sobre esta variable respuesta. El conjunto de reglas difusas nos indica cual es la relación entre los valores de unas variables con otras. Esta variable respuesta puede a su vez influir en otras variables.

Si observamos el formalismo que estamos utilizando, éste nos permite simplificar en gran manera como expresamos las relaciones entre evidencias y conclusiones. Dada una regla difusa, esta tiene en cuenta siempre todos los valores posibles de los antecedentes y se puede utilizar el valor de verdad de cada uno de ellos como un factor de influencia sobre la conclusión, lo que nos permite obtener una respuesta gradual a todos los posibles casos que caen bajo el antecedente de la regla.

Hay dos modelos de inferencia difusa que se utilizan habitualmente, el primero es el modelo de *Mamdani*. Éste asume que la conclusión de la regla es un conjunto difuso y los antecedentes permiten obtener un valor que modifica el conjunto según el valor de verdad del antecedente. El segundo es el modelo de *Sugeno*, éste considera que la conclusión de la regla es una función (por lo general lineal) de los valores del antecedente modificada por el valor de verdad de éste. Nosotros solo nos preocuparemos del primer modelo.

12.5.1 Modelo de inferencia difusa de Mamdani

Dado que los antecedentes de las reglas corresponden a conjuntos difusos, un valor concreto puede disparar varias reglas a la vez, por lo que deberemos evaluarlas todas. Cada una de ellas nos dará un valor de posibilidad que será la fuerza con la que cada regla implica el consecuente.

El proceso de inferencia con datos precisos tendrá los siguientes pasos:

1. **Evaluación de los antecedentes de las reglas:** Esta evaluación pasará por obtener el valor de posibilidad que nos indican los valores precisos para cada elemento del antecedente y su combinación mediante los operadores difusos correspondientes

2. **Evaluación de los consecuentes de las reglas:** Cada uno de los consecuentes se ponderará según el valor de posibilidad indicado por su antecedente. El resultado de esta ponderación será la etiqueta de la conclusión recortada al valor de posibilidad del antecedente.

[2]La conoceréis de la asignatura de lógica como la eliminación de la implicación $A, A \rightarrow B \vdash B$

3. **Combinación de las conclusiones:** Todas las conclusiones de las reglas se combinarán en una etiqueta que representa la conclusión conjunta de las reglas

4. **Obtención del valor preciso (Nitidificación[3]):** Se calcula el valor preciso correspondiente a la etiqueta obtenida. Este cálculo se puede hacer de diferentes maneras, una posibilidad es obtener el centro de gravedad de la etiqueta.

   El centro de gravedad se calcula como una integral definida sobre la función que describe la etiqueta resultante:

$$CDG(f(x)) = \frac{\int_a^b f(x) \cdot x dx}{\int_a^b f(x) dx}$$

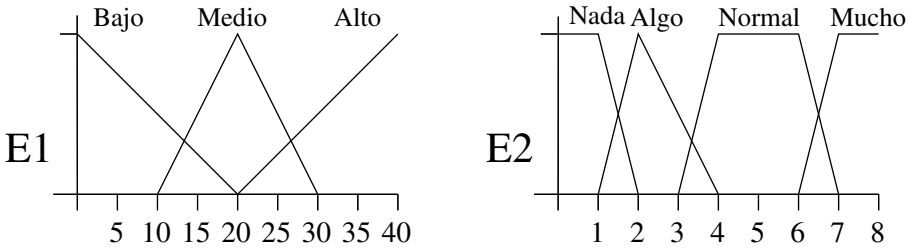
   donde a y b son los extremos de la etiqueta. Ésto no es mas que una versión continua de una media ponderada.

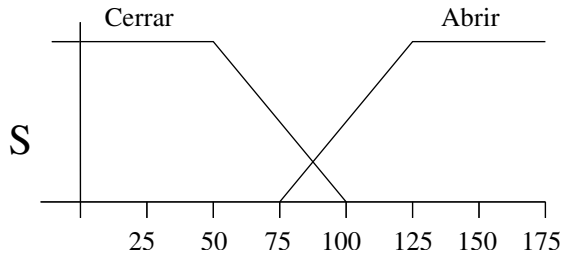
   Existen otras posibilidades, con un coste computacional menor, para obtener un valor preciso de la etiqueta resultante de la composición de reglas, como por ejemplo, escoger la media de los valores donde la etiqueta sea máxima.

Ejemplo 12.3 *Supongamos que tenemos dos variables de entrada E1 y E2 y una variable de salida S que pueden tomar valores en los conjuntos siguientes de etiquetas lingüísticas:*

- *E1={bajo,medio,alto}*
- *E2={nada,algo,normal,mucho}*
- *S={cerrar,abrir}*

Estas etiquetas están descritas mediante los siguientes conjuntos difusos





Supongamos que tenemos el siguiente conjunto de reglas:

R1. si ([E1=medio] o [E1=alto]) y [E2=mucho] entonces [S=cerrar]

[3]Esta es una adaptación del término ingles *defuzzification*

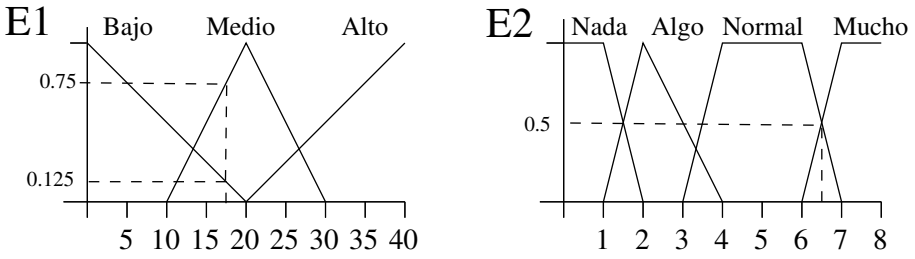
R2. si [E1=alto] y [E2=normal] entonces [S=cerrar]

R3. si [E1=bajo] y no([E2=mucho]) entonces [S=abrir]

R4. si ([E1=bajo] o [E1=medio]) y [E2=algo] entonces [S=abrir]

Las funciones de combinación son el mínimo para la conjunción, máximo para la disyunción y $1-x$ *para la negación y los valores concretos que tenemos para las variables* $E1$ *y* $E2$ *son 17.5 y 6.5 respectivamente.*

Si trasladamos esos valores a los conjuntos difusos de las variables $E1$ *y* $E2$ *obtenemos los siguientes valores de posibilidad*



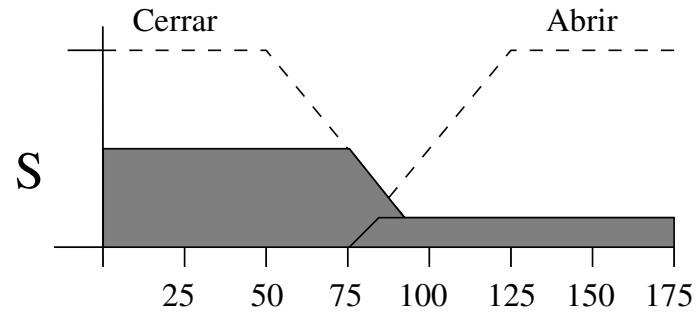
Si evaluamos la regla R1 tendríamos:
[E1=medio] = 0.75, [E1=alto] = 0, [E2=mucho] = 0.5 $\Rightarrow$ *min(max(0.75,0),0.5) = 0.5*
Por lo que tenemos 0.5 $\cdot$ *[S=cerrar]*

Si evaluamos la regla R2 tendríamos:
[E1=alto] = 0, [E2=normal] = 0.5 $\Rightarrow$ *min(0,0.5) = 0*
Por lo que tenemos 0 $\cdot$ *[S=cerrar]*

Si evaluamos la regla R3 tendríamos:
[E1=bajo] = 0.125, [E1=mucho] = 0.5, $\Rightarrow$ *min(0.125,1-0.5) = 0.125*
Por lo que tenemos 0.125 $\cdot$ *[S=abrir]*

Si evaluamos la regla R4 tendríamos:
[E1=bajo] = 0.125, [E1=medio] = 0.75, [E2=algo] = 0 $\Rightarrow$ *min(max(0.125,0.75),0) = 0*
Por lo que tenemos 0 $\cdot$ *[S=abrir]*

La etiqueta resultante de la combinación de las conclusiones sería:



Ahora debemos hallar el centro de gravedad de la etiqueta, ésta se puede describir mediante la función:

$$f(x) = \begin{cases} 0.5 & x \in [0-75] \\ (100-x)/50 & x \in (75-93.75] \\ 0.125 & x \in (93.75-100] \end{cases}$$

Por lo que podemos calcular el CDG como:

$$\frac{\int_0^{75} 0.5 \cdot x \cdot dx + \int_{75}^{93.75} (100 - x)/50 \cdot x \cdot dx + \int_{93.75}^{175} 0.125 \cdot x \cdot dx}{\int_0^{75} 0.5 \cdot dx + \int_{75}^{93.75} (100 - x)/50 \cdot dx + \int_{93.75}^{175} 0.125 \cdot dx} =$$

$$\frac{0.25 \cdot x^2|_0^{75} + (x^2 - x^3/150)|_{75}^{93.75} + 0.0625 \cdot x^2|_{93.75}^{175}}{0.5 \cdot x|_0^{75} + (2 \cdot x - x^2/100)|_{75}^{93.75} + 0.125 \cdot x|_{93.75}^{175}} =$$

$$\frac{(0.25 \cdot 75^2 - 0) + ((93.75^2 - 93.75^3/150) - (75^2 - 75^3/150)) + (0.0625 \cdot 175^2 - 0.0625 \cdot 93.75^2)}{(0.5 \cdot 75 - 0) + ((2 \cdot 93.75 - 93.75^2/100) - (2 \cdot 75 - 75^2/100)) + (0.125 \cdot 175 - 0.125 \cdot 93.75)} =$$

$$\frac{3254.39}{53.53} = 60.79$$

Bibliografía

[1] Joseph Giarratano, Gary Riley *Expert systems: principles and programming*, Thomson Course Technology, 2005.

[2] Avelino J. Gonzlez, Douglas D. Dankel *The Engineering of knowledge-based systems : theory and practice*, Prentice Hall, 1993.

[3] Jackson, P. *Introduction to Expert Systems*, Addison-Wesley, 1990.

[4] Noy & McGuinness *Ontology Development 101: A Guide to Creating Your First Ontology*, (2000)

[5] Stuart J. Russell and Peter Norvig *Artificial intelligence: a modern approach*, Prentice Hall, 2009.

Índice alfabético