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.

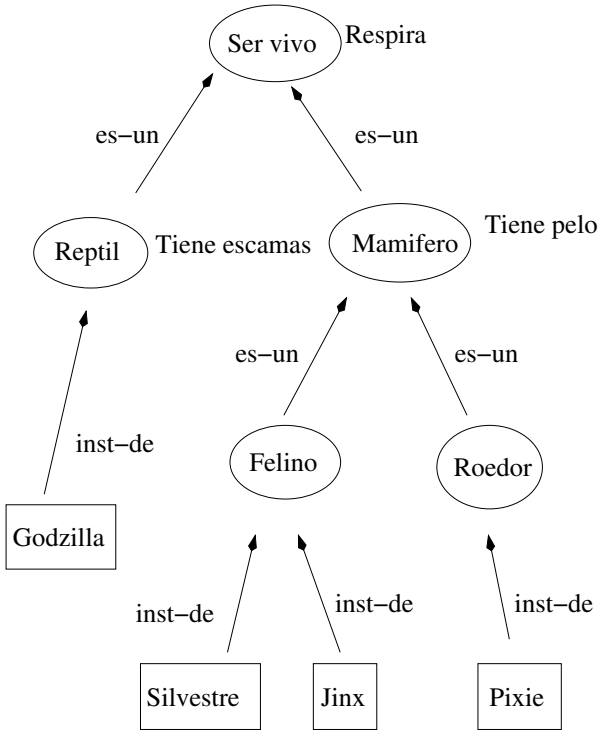


Figura 1.1: Jerarquia de conceptos

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:

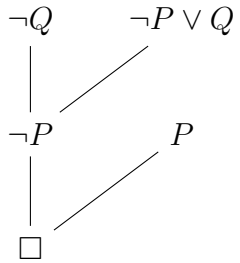
$$A \vee B, \neg A \vee C \vdash B \vee C$$

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.

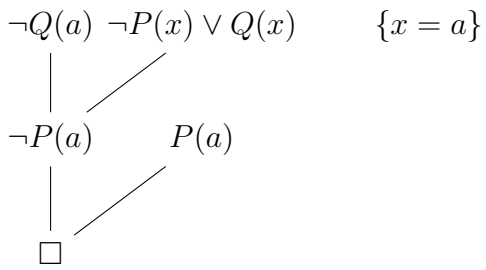
El método de resolución también se puede aplicar como método de validación en lógica de predicados, pero necesita de un conjunto de procedimientos adicionales para tener en cuenta las extensiones que hemos hecho en el lenguaje. En este caso las fórmulas son transformadas a la *forma normal de skolem*, 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**.

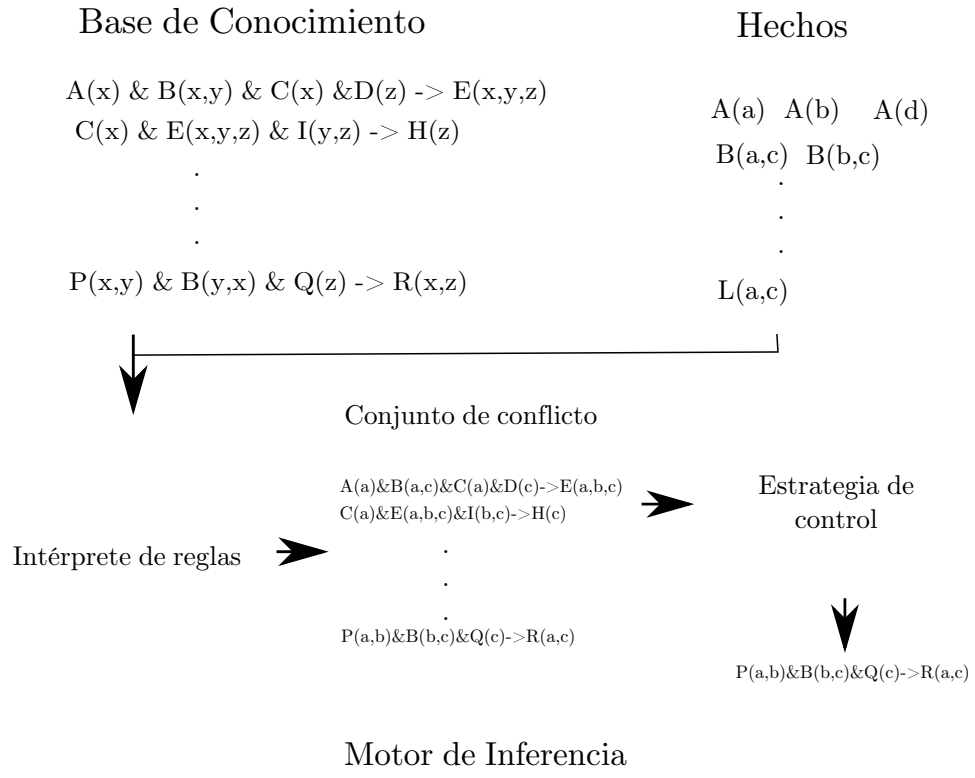


Figura 3.1: Sistema de producción

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.

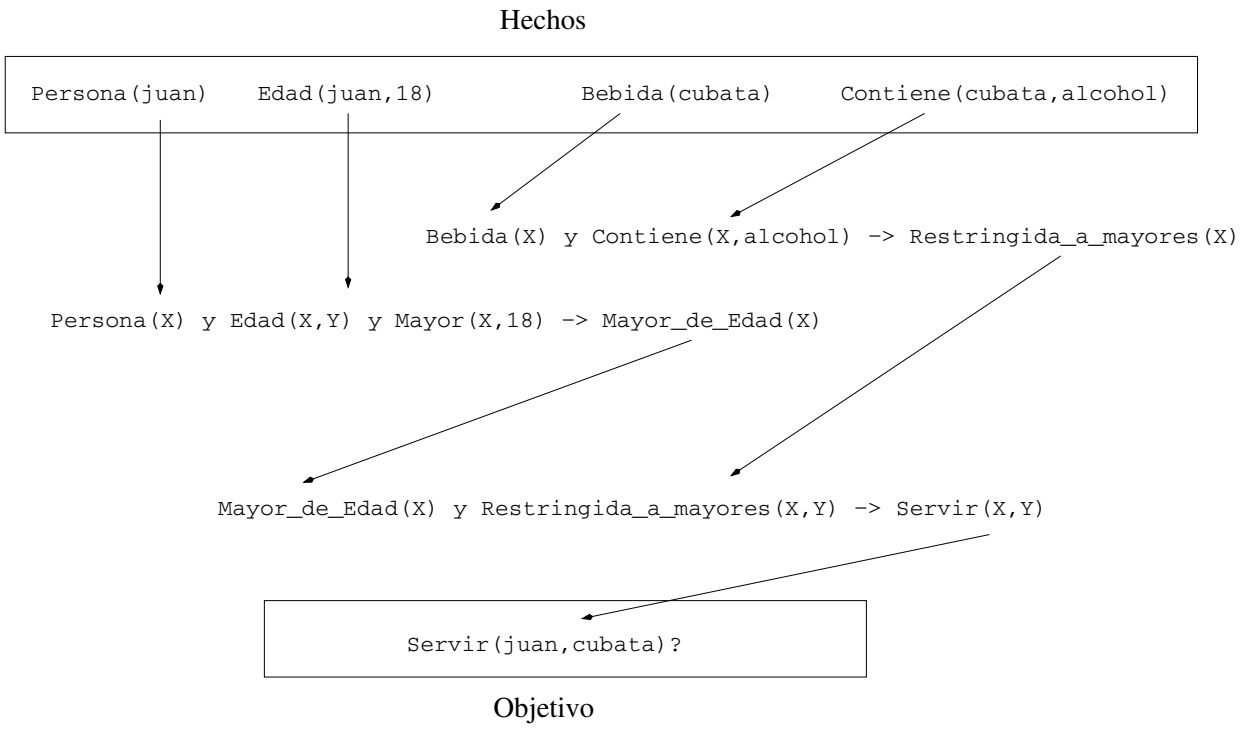


Figura 3.2: ¿Le servimos a Juan un cubata?

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.

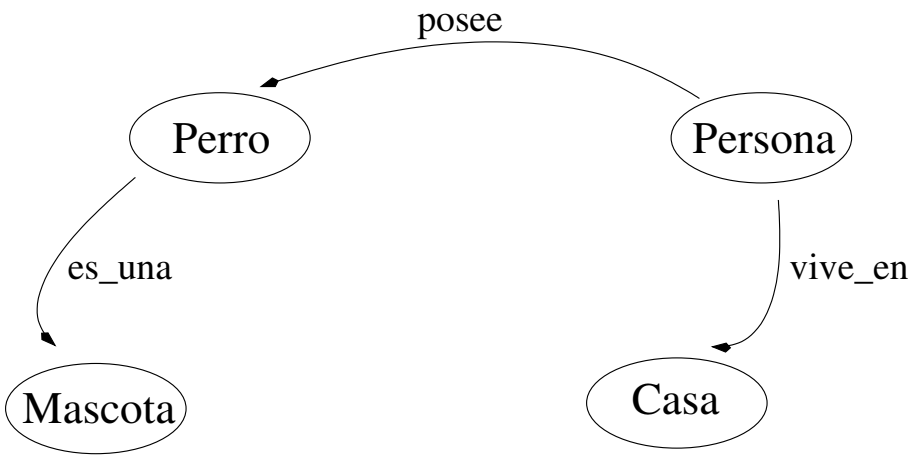


Figura 4.1: Ejemplo de red semántica

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.

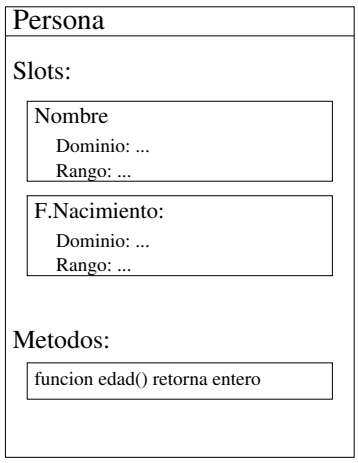


Figura 4.2: Ejemplo de frame

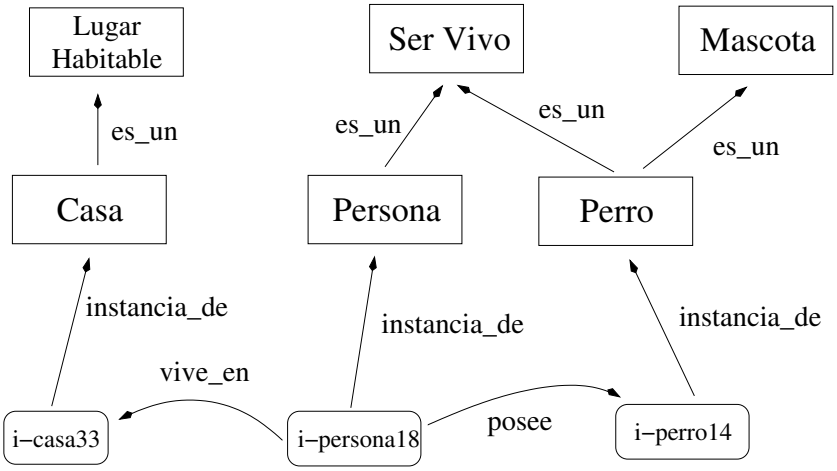


Figura 4.3: Ejemplo de red de frames

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.

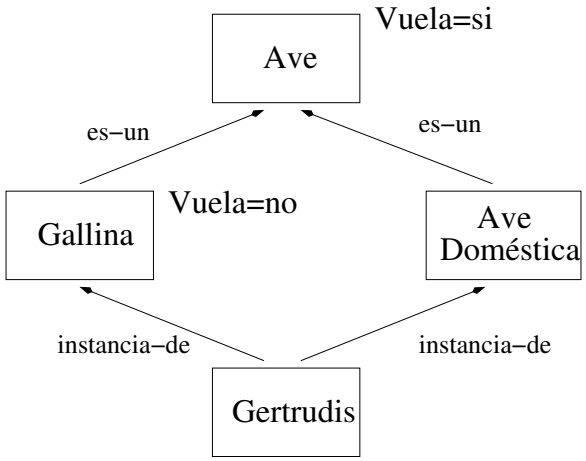


Figura 4.4: Ejemplo de frames

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++.

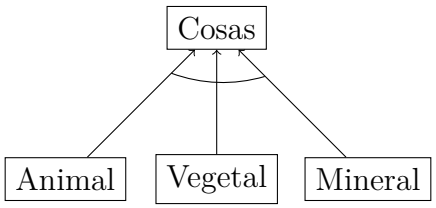


Figura 4.5: Subclasificación completa de un concepto

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*

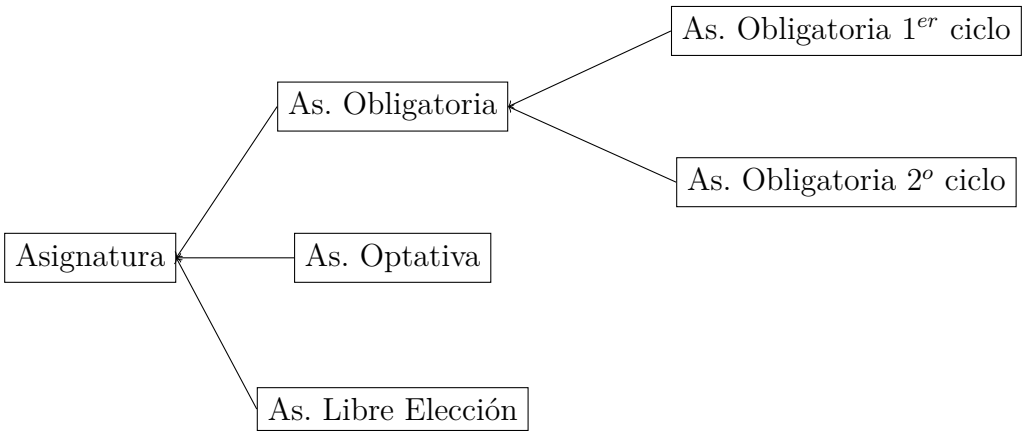


Figura 5.1: Jerarquía del concepto asignatura

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.

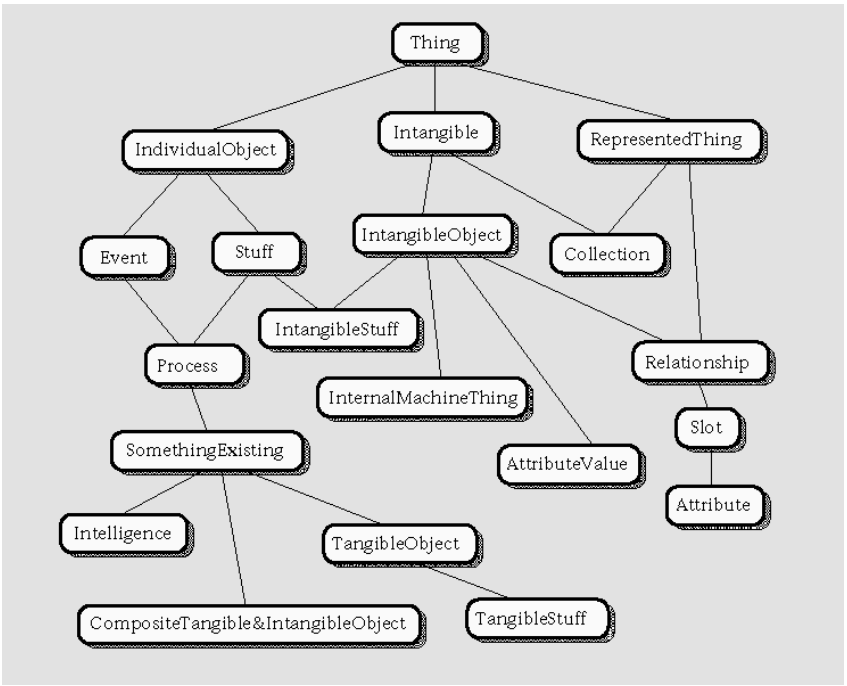


Figura 5.2: Ontología de CYC

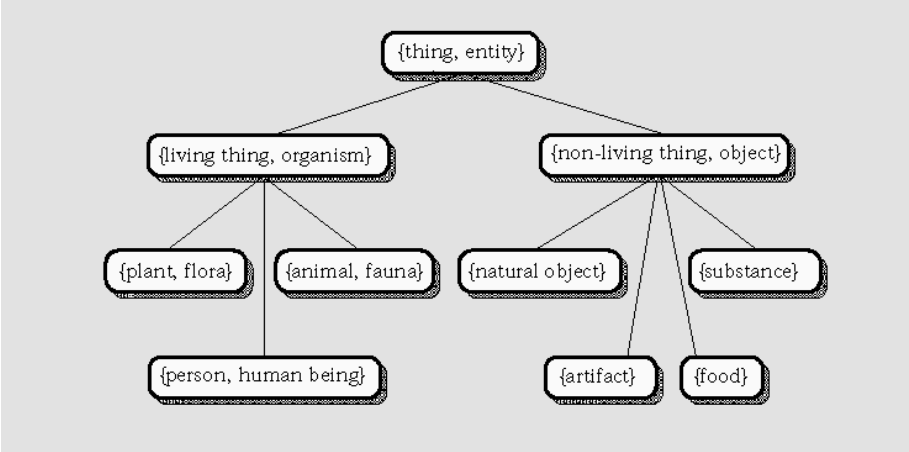


Figura 5.3: Ontología de Wordnet

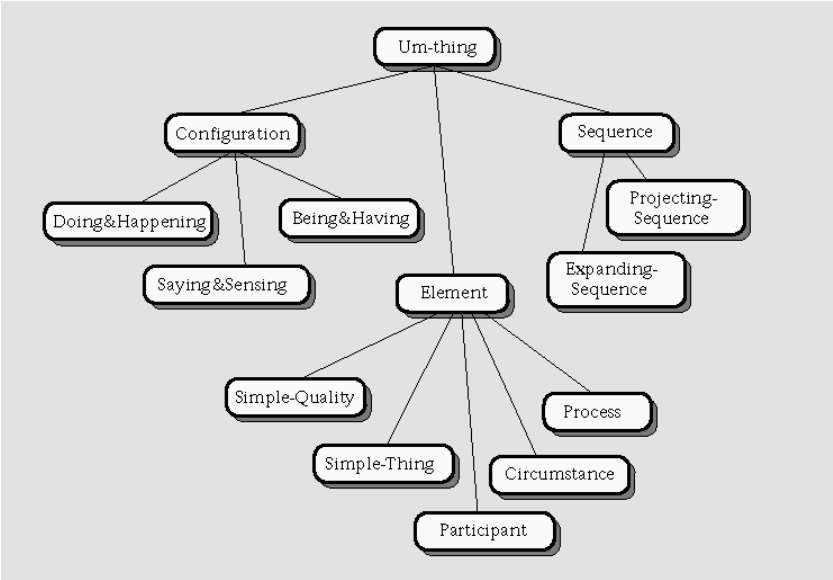


Figura 5.4: Ontología de Generalized Upper Model