Contents

Introducción (PDF) **5**

1. Caracterización y perspectiva de agente 5

1.1. Caracterización 5

1.2 ¿Qué es un sistema inteligente? 5

1.3 Concepto de agente 5

1.4 Comportamiento de un agente 6

1.5 Mundo del aspirador 6

1.6 ¿Cómo determinar la calidad de un agente? 6

1.7 Agente racional 7

1.8 *Racionalidad* no es perfección y requiere aprendizaje 7

1.9 Ejemplos de agentes y descripción REAS (Medida de Rendimiento, Entorno, Actuadores y Sensores) 8

1.10 Estructura de agente 8

2. Paradigmas principales 10

2.1 Sistemas Inteligentes basados en conocimiento (Sistema experto) 10

2.2 Razonamiento basado en casos 11

2.3 Aprendizaje y minería de datos 11

2.4 Razonamiento basado en modelos 12

Lógica proposicional **13**

Introducción 13

Lenguaje 13

Sintaxis 13

Semántica 14

Modelo, consistencia y validez 14

Satisfactibilidad 15

Equivalencia 15

Consecuencia lógica 16

Teorema de refutación 16

Lógica de predicados **16**

Introducción 16

Lenguaje de la lógica de primer orden 17

Sintaxis 17

Semántica 18

Modelo, consistencia y validez 18

Satisfactibilidad 18

Equivalencia 19

Consecuencia lógica 19

Teorema de refutación 20

Reglas de inferencia 20

Teoría 20

Unificación 21

Sustituciones 21

Unificador más general 22

Algoritmo de unificación 22

Estrategias de resolución **22**

Introducción 22

Refutación por resolución 23

Estrategias de resolución 24

Estrategias de dirección: saturación por niveles 24

Estrategias de simplificación 24

Estrategias de restricción 25

Procedimiento de extracción de respuesta 26

Demostradores de teoremas 27

Prolog Technology Theorem Prover (PTTP) 27

Organized Techniques for Theorem-proving and Effective Research (OTTER) 27

Programación lógica y prolog **27**

Introducción a la programación lógica 27

Caracterización de programas lógicos 28

Programas definidos: sintaxis de Edimburgo 28

Resolución SLD 28

Derivación SLD (Cómputo de G por P) 29

Intérprete abstrácto de un programa lógico 29

Concepto de respuesta 29

Teorema de solidez de la resolución SLD 29

Teorema de complitud de la resolución SLD 29

Diferencias entre la respuesta correcta y la computada 30

Teorema de la independencia de la regla de cómputo 30

Intérprete abstracto de un programa lógico 30

Concepto de respuesta 30

Programación lógica y negación 30

Una implementación práctica: prolog estándar 30

Anexo: árbol SLD 31

Representación del conocimiento: Introducción (PDF) **31**

Sistemas basados en conocimiento 31

Tipos de conocimiento 31

Biológico 32

Límites de aplicación 32

Niveles 32

Contenido (Clancey, 85) 33

Tipos de representación 33

Lenguajes de representación de conocimiento 33

Representación del conocimiento: Lógica y representación del conocimiento (PDF) 34

Papel de la lógica en la representación del conocimiento 34

Problema de la cualificación 34

Debate histórico 34

Tendencia actual 34

2. Principios de ingeniería de conocimiento en lógica de primer orden . 35

Ejemplo de desarrollo de una ontología específica y base de conocimiento en el dominio de los circuitos digitales 36

Representación de conocimiento: Sistemas Basados en Reglas (SBR) (PDF) 36

1. Introducción 36

2. Componentes de un sistema de producción 37

3. Lenguajes 38

Lenguaje objeto-atributo-valor 38

Declaración de dominio 38

Hechos en objeto-atributo-valor 39

Memoria de trabajo 39

Reglas en objeto-atributo-valor 39

Condiciones 40

Semántica de los predicados 40

Conclusiones 40

Acción básica: añadir 40

Acción eliminar 41

4. Inferencia en un sistema de producción 41

Espacio de búsqueda: hipergrafos o grafos y/o 41

Métodos básicos de inferencia 41

Estrategias de resolución de conflictos 42

5. Encadenamiento hacia adelante 43

Encadenamiento hacia atrás 43

Lenguajes con variables 43

Ligadura de variables 43
Equiparación de patrones 44
Semántica 44
Particularización de regla 44
Algoritmo de Rete 45

Representación del conocimiento: métodos estructurados 45

Introducción (PDF) 45
Evolución 45
Redes semánticas 46
Sintaxis 46
Inferencia en redes semánticas 47
Marcos (PDF) 48
Introducción 48
Elementos de un sistema de marcos 49

Introducción (PDF)

1. Caracterización y perspectiva de agente

1.1. Caracterización

La IA se refiere al diseño y análisis de agentes autónomos. Son sistemas software o máquinas físicas con sensores y actuadores. Un **sistema inteligente** tiene que percibir su entorno, actuar racionalmente hacia sus tareas preasignadas, interaccionar con otros agentes y con seres humanos.

1.2 ¿Qué es un sistema inteligente?

Un sistema inteligente es un agente inteligente o, mejor dicho, racional. Se verá más adelante.

1.3 Concepto de agente

Entidad que percibe su entorno a través de sensores y modifica el entorno mediante actuadores. El típico ejemplo del termostato.

1.4 Comportamiento de un agente

Teóricamente la **función de agente**, que consiste en, dada una secuencia de percepciones, ejecutar una serie de acciones.

1.5 Mundo del aspirador

Fila de dos casillas. El agente detecta su situación y si la casilla en la que está está sucia. Sus posibles acciones son derecha, izquierda, asiparar y noop. Su función de agente es: si la casilla actual está sucia, aspirar. Si no, ir a la otra casilla.

Tenemos pues secuencias de percepciones, acciones y una función de agente que las relaciona, pero también tenemos un entorno. El entorno puede ser:

- Discreto o continuo.
- Estático (No cambia mientras el agente delibera) o dinámico.
- Determinista (Las acciones siempre funcionan como se espera) o estocástico.
- Uniagente o multiagente.
- Episódico (La calidad de las acciones no depende de las decisiones previas) o secuencial.
- Parcial o totalmente observable.

El mundo del aspirador es de cada una de estas opciones la más sencilla excepto en el último caso, puesto que el mundo es parcialmente observable. El agente aspirador solo conoce el estado de la casilla en la que se encuentra.

1.6 ¿Cómo determinar la calidad de un agente?

Hay que definir una medida del rendimiento que cuantifique el grado de éxito, que sea objetiva y externa, es decir, desde el punto de vista del entorno.

Para el mundo del aspirador, se podrían definir las siguientes:

- Suciedad aspirada en un turno de 8 horas.
- Tiempo total en que el suelo está limpio.
- Suelo limpio sin demasiado consumo eléctrico.

Cada una tiene sus pegas, hay muchas variables que intervienen y que queremos optimizar.

1.7 Agente racional

Según Rusell y Norvig:

> Aquél que para cada posible secuencia de percepciones, realiza la acción que se espera que maximice su medida de rendimiento, basándose en la evidencia proporcionada por su secuencia de percepción y el conocimiento que el agente mantiene almacenado.

¿El agente aspirador es racional? Depende. Supone que:

- el entorno tiene una geometría conocida y una distribución inicial de suciedad desconocida,
- aspirar limpia una casilla y ésta permanece limpia,
- derecha e izquierda mueven al agente a la derecha y a la izquierda sin errores, salvo si le sacan del entorno, que permanece en su casilla,
- la percepción de situación y suciedad siempre es correcta,
- tiene cuatro acciones perfectamente definidas (aspirar, dcha, izda, noop),
- función de agente descrita anteriormente,
- rendimiento medido como 10 puntos por casilla limpia por instante de tiempo.

En ese caso **sí es racional** porque **ningún otro agente obtendrá un mejor rendimiento esperado**. Si el comportamiento parece defectuoso se puede definir otra medida de rendimiento, por ejemplo, suponer que no queremos al agente moviédose en un entorno limpio. Añadimos a la medida de rendimiento una penalización de 1 punto por cada movimiento.

Para maximizar el rendimiento, añadimos a la función de agente que si la secuencia de percepciones incluye ambas casillas limpias, la acción será `noop`.

1.8 *Racionalidad* no es perfección y requiere aprendizaje

Un comportamiento perfecto requiere saber por adelantado el resultado de las acciones del agente: solo se pide maximizar el rendimiento esperado.

Excepto para entornos sencillos e invariables, el conocimiento inicial del agente no puede garantizar maximizar el rendimiento a largo plazo.

La racionalidad normalmente requiere aprendizaje para proporcionar autonomía al agente.

Tipo de Agente	Medida de Rendimiento	Entorno	Actuadores	Sensores
Robot para la selección de componentes	Porcentaje de componentes clasificados en los cubos correctos	Cinta transportadora con componentes, cubos	Brazo y mano articulados	Cámara, sensor angular
Controlador de una refinería	Maximizar la pureza, producción y seguridad	Refinería, operadores	Válvulas, bombas, calentadores, monitores	Temperatura, presión, sensores químicos
Tutor de Ingles interactivo	Maximizar la puntuación de los estudiantes en los exámenes	Conjunto de estudiantes, agencia examinadora	Visualizar los ejercicios, sugerencias, correcciones	Teclado de entrada

Figure 1: Ejemplos de agentes

1.9 Ejemplos de agentes y descripción REAS (Medida de Rendimiento, Entorno, Actuadores y Sensores)

1.10 Estructura de agente

La función de agente no se implementa, pues no se suele conocer. Se **implementa el programa de agente**, que usa solo la percepción actual del entorno y el conocimiento y memoria que pueda tener para seleccionar la acción actual.

Un agente es una **entidad que percibe su entorno a través de sensores y lo modifica mediante actuadores**.

La mayor parte de los principios subyacentes a los sistemas inteligentes se pueden describir con 5 tipos básicos de programas de agente:

- Agente reactivo simple
- Agente reactivo basado en modelos
- Agente basado en metas
- Agente basado en utilidad
- Agentes que aprenden

1.10.1 Agente reactivo simple La entrada de los sensores se usa en una descripción del mundo contenida en el agente, que decide la acción a tomar a través de reglas condición/acción.

Un agente de este tipo es suficiente si la acción actual se puede tomar a partir de la percepción actual. Es útil para muchas tareas de clasificación como por ejemplo:

- Riesgo crediticio.
- Etiquetado de imágenes.
- Comportamiento anómalo en aeropuertos.
- Sistemas sencillos de diagnosis de fallos, etc.

Sin embargo, el agente reactivo simple es muy limitado. No es útil en entornos no totalmente observables y secuenciales. Un agente reactivo simple no puede ser racional con la función de rendimiento definida para el mundo del aspirador (10 puntos por casilla limpia por instante de tiempo, -1 punto por cada movimiento).

1.10.2 Agente reactivo basado en modelos Añade estado, evolución del mundo y efecto de las acciones a la descripción del mundo del modelo de agente reactivo simple.

Puede generar comportamiento racional en numerosas tareas analíticas.

Sistemas basados en conocimiento para tareas analíticas:

- Clasificación.
- Evaluación (Assessment).
- Monitorización.
- Diagnosis.

También tiene limitaciones. A veces, el agente necesita considerar su meta actual para seleccionar la mejor acción. Por ejemplo, el aspirador en un edificio docente tendría dos metas: limpieza y silencio. Habría que fomentar el silencio en horas de docencia y la limpieza fuera de dicho horario.

1.10.3 Agente basado en metas / objetivos Entre la descripción del mundo y la acción, el agente analiza el efecto de realizar una acción teniendo en cuenta la evolución del mundo y el efecto de sus acciones para determinar la acción a tomar. Normalmente el agente considera secuencias de acciones.

Se usa en problemas de búsqueda (rutas, juegos, etc) o en problemas de planificación (horarios de clase, asignación de tiempos a tareas o scheduling, etc).

Sus limitaciones radican en que hay algunas metas que se contradicen y el agente tendría que razonar sobre ellas (como en el ejemplo del aspirador en el centro educativo). También está limitado cuando hay incertidumbre sobre la obtención de las metas, como por ejemplo en un juego de póker.

1.10.4 Agente basado en utilidad

1.10.5 Agentes que aprenden

2. Paradigmas principales

2.1 Sistemas Inteligentes basados en conocimiento (Sistema experto)

No son solo técnicas de búsqueda, sino que incluyen conocimiento, implícito o explícito, sobre el problema a resolver. Son de interés en numerosos entornos como juegos, optimización de rutas, etc.

Los sistemas expertos identifican y explotan el conocimiento humano para la resolución de problemas. Se basan en unas suposiciones básicas:

- El conocimiento proviene de la experiencia.
- Los expertos codifican el conocimiento mediante asociaciones (heurística).
- El conocimiento se puede extraer del experto y codificar mediante un Lenguaje de Representación de Conocimiento.

El conocimiento se codifica con **reglas de producción** que tienen la estructura `SI antecedente ENTONCES consecuente`. Un ejemplo sería:

```
SI las diferencias de temperatura son anormalmente altas Y las presiones se mantienen aproxim
```

Un ejemplo de sistema experto es *MYCIN*, que tomaba decisiones de diagnosis y terapia de enfermedades infecciosas. Su base de conocimiento estaba compuesta de reglas de producción como la siguiente:

```
if  (1) the stain of the organism is gram-negative
    (2) the morphology of the organisms is coccus
    (3) the growth configuration of the organism is chains
then there is a suggestive evidence (0.7)
    that the identity of the organisms is streptococcus
```

Control: encadenamiento hacia atrás, meta-reglas

Razonamiento aproximado con factores de certeza. No daba respuestas absolutas, sino probabilidades.

Los sistemas expertos tienen numerosos dominios de aplicación: medicina, mecánica, electrónica, control de procesos, aeronáutica, etc. Existen numerosas tareas basadas en el conocimiento como la diagnosis, configuración, clasificación y otras.

Existen numerosos sistemas expertos. Los más representativos son *MYCIN*, *INTERNIST* o *R1/XCONF*.

La aproximación de los sistemas expertos está bien establecida a través de una serie de metodologías (ingeniería del conocimiento, ontologías) y sistemas. Esta

aproximación será adecuada si hay suficiente experiencia, si no hay otras fuentes de conocimiento, si hay suficientes observaciones y si el sistema permanece estable.

También existen algunos inconvenientes. Sobre la experiencia, puede que sea difícil extraer el conocimiento, disponer de experiencia o expertos, y además existe dependencia del dispositivo . Sobre el método de solución, pueden darse situaciones nuevas no previstas, puede que se combinen soluciones (por ejemplo, a la hora de encontrar la causa de un fallo múltiple), y el sistema puede ser frágil. También es costosa la obtención del conocimiento, la reutilización del mismo y el mantenimiento de la consistencia de la base de conocimiento; problemas que hay que resolver desde la ingeniería de software.

2.2 Razonamiento basado en casos

Un caso es la descripción de un problema y su solución. Bajo este concepto, para solucionar un problema nuevo hay que buscar los casos con descripciones similares, adaptar las soluciones revisando el resultado, y almacenar el nuevo caso generado si el resultado es interesante.

Este proceso, denominado **ciclo CBR** (case-based reasoning) se formaliza de la siguiente manera: Partimos de un problema, y lo primero es la fase de recuperación (retrieve) de los casos más parecidos. Después, reutilizamos (reuse) las soluciones de los casos recuperados para tratar de resolver el problema. Seguimos revisando (revise) la solución propuesta hasta dar con una correcta, que almacenamos (retain) como un nuevo caso.

Ventajas:

- No requiere representación explícita del conocimiento.
- Tiene capacidad de aprendizaje.
- Es adecuado si no se dispone de un modelo explícito.

Inconvenientes:

- Son necesarios casos previos.
- La etapa de revisión (revise) tiene limitaciones.

2.3 Aprendizaje y minería de datos

Permiten inferir conocimiento nuevo ya sea describiendo conceptos a partir de ejemplos etiquetados (clasificación), descubriendo conceptos a partir de ejemplos no etiquetados (clustering), descubriendo regularidades en los datos o mejorando la eficiencia.

Ventajas:

- Permiten generar automáticamente conocimiento a partir de datos.
- Es adecuado si no dispone de modelo explícito.

Inconvenientes:

- Es necesario disponer de datos suficientes.
- Suele ser de utilidad solo en alguna parte del proceso de solución.

2.4 Razonamiento basado en modelos

Se tiene un modelo que predice el comportamiento del sistema real. Esta predición se compara con el comportamiento observado y deseado y con las diferencias se genera una solución.

El conocimiento se codifica en modelos (estructura y comportamiento) de los componentes del sistema. El razonamiento se traduce en el proceso de manipulación de los modelos hasta obtener la solución del problema.

Ventajas:

- Es independiente de la experiencia, y por tanto aplicable a dispositivos nuevos.
- Independencia del dispositivo (Problema de las variantes).
- Soluciones complejas (Por ejemplo, fallos múltiples).
- Sólido y completo respecto a los modelos.
- Mantenimiento y reutilización del conocimiento a través de bibliotecas de modelos que están disponibles desde el diseño.

Inconvenientes:

- Dificultad de obtención de los modelos. Hay procesos que son poco conocidos, sistemas con demasiados componentes, comportamientos complejos como dinámica, no linealidades, modelos con un rango de validez, etc.
- Existe una mayor carga computacional del proceso de razonamiento, lo que limita esta técnica en aplicaciones en tiempo real.

Áreas de aplicación:

- Planificación y scheduling.
- Configuración y diseño.
- Diagnosis.
- Control.
- Visión.

- Tecnologías del habla.
- Robótica.
- Lenguaje natural.
- Análisis de transacciones.
- Industria informática actual: navegadores, buscadores, gestor e-mail, reconocimiento de matrículas en aparcamientos, compras on-line…

En la industria del automóvil es de gran interés en cuestiones de seguridad, medio ambiente y economía. Hay proyectos de diagnosis a bordo o en el taller, de elaboración de manuales o de diagnosis y mantenimiento preventivo.

En la industria aeroespacial se usa en sistemas de monitorización de las funciones básicas de las naves espaciales (Health Management Systems) como los subsistemas de propulsión, de guiado, de sostenimiento de vida, etc. También para detección, localización y reconfiguración de satélites o lanzaderas por parte de múltiples agencias espaciales y empresas privadas.

Muy útil también en la supervisión de procesos industriales, aportando una atención continuada, seguridad y una calidad homogénea.

Lógica proposicional

Introducción

Es el lenguaje lógico simbólico más sencillo. Representa sentencias simples del lenguaje mediante **fórmulas atómicas** que se van componiendo.

Lenguaje

Sintaxis

Símbolos proposicionales que representan proposiciones. Representados por letras del final del abecedario en minúscula, normalmente empezando por la `p`.

Conectores lógicos:

- `¬`: Negación lógica (no)
- `^`: Conjunción lógica (y)
- `v`: Disyunción lógica (0)
- ` `: Condición lógica (implica)
- `<->`: Bicondición lógica (sí y solo sí)

Símbolos auxiliares, que son los paréntesis y ayudan a expresar precedencia.

Estos tres grupos no pueden compartir símbolos entre sí.

El **alfabeto proposicional** está compuesto por la unión de los símbolos proposicionales, los conectores lógicos y los símbolos auxiliares.

Una **fórmula atómica o átomo** es cualquier símbolo proposicional.

Una **fórmula bien formada** (FBF) es:

- Un átomo.
- La negación de otra FBF.
- La conjunción, disyunción, condición o bicondición de dos FBF.

Se usan convenios de asociatividad y prioridad de los conectores para reducir el uso de paréntesis.

Semántica

Una **interpretación** sobre los símbolos proposicionales es una función que asigna a cada átomo un valor de verdad:

```
I: SP ---> {T, F}
```

Una **función de evaluación** de FBFs, a partir de una interpretación `I`, se define de la siguiente forma:

```
V: L_{AP} ---> {T, F}
```

- Para átomos, el valor de la función es el mismo que el de la interpretación.
- Para átomos negados, la negación del valor interpretado del átomo sin negar.
- Para el resto de FBFs, se siguen las reglas de los conectores lógicos y la interpretación de cada uno de los átomos.

Modelo, consistencia y validez

Una interpretación es **modelo** de una FBF si la evaluación de la FBF devuelve un valor verdadero. Una interpretación es modelo de un conjunto finito de FBFs si la evaluación es verdadera para todas las FBFs. Es decir, el modelo de una FBF es una interpretación que la hace verdadera.

Una FBF es **consistente o satisfactible** si y solo si tiene un modelo. Un conjunto finito de FBFs lo es si tiene modelo.

Una FBF, o un conjunto finito de ellas, es **inconsistente o insatisfactible** si no es consistente.

Una FBF es **válida o tautológica** si y solo si la FBF es cierta bajo todas las interpretaciones de sus símbolos proposicionales.

Satisfactibilidad

El **problema de la satisfactibilidad** es demostrar la consistencia o inconsistencia de un conjunto finito de fórmulas.

Para probar en lógica proposicional…

- la consistencia, hay que obtener una interpretación que valide las fórmulas.
- la inconsistencia y validez, por medio de tablas de verdad o refutación.

Para probar por refutación la inconsistencia de `((p q) (p ¬q))`, suponemos que existe una interpretación tal que `V( ) = T`, lo que implica que `V(p q) = V(p ¬q) = T`, cosa que no puede ser cierta así que queda demostrado.

Para probar con tablas de verdad la inconsistencia de la misma FBF habría que considerar 2^3 interpretaciones y una tabla con 7 entradas, mucho más costoso.

Una lógica es **decidible** si el problema de la satisfactibilidad es computable en esa lógica. Es decir, si se puede dar un procedimiento de cómputo que para cualquier conjunto finito de FBFs como entrada, devuelva si es o no consistente.

La lógica proposicional es decidible porque siempre se puede recurrir a las tablas de verdad. Sin embargo el problema es NP y O(2^n).

Equivalencia

Dos FBFs son **equivalentes** (=) si y solo si ambas tienen los mismos valores de verdad bajo cualquier interpretación. Las leyes de equivalencia son las siguientes:

- Eliminación del bicondicional: conjunción de condicionales.
- Eliminación del condicional: ( ) = (¬)
- Conmutativa de la conjunción y la disyunción.
- Asociativa de la conjunción.
- Distributiva de disyunción respecto de conjunción.
- Absorción: La conjunción de una FBF con otra válida es equivalente a la primera FBF. La disyunción con una inconsistente también.
- Contradicción: La conjunción de una FBF con su opuesto o con una FBF inconsistente es equivalente a una FBF inconsistente.

- Exclusión de los medios: La disyunción de una FBF con su opuesto o con una FBF válida es equivalente a una FBF válida.
- Idempotencia: La conjunción o disyunción de una FBF consigo misma es equivalente a la FBF.
- Doble negación.
- De Morgan: Negación de la conjunción, disyunción de negaciones. Negación de la disyunción, conjunción de negaciones.

Consecuencia lógica

Una FBF es **consecuencia lógica** de una serie de premisas (más FBFs) si y solo si todo modelo de las premisas es modelo de la consecuencia lógica. Se denota por `1, 2, ... , n |=` .

Teorema de refutación

Sean , 1, 2, … , n FBF's. Las siguientes expresiones son equivalentes

1. 1, 2, …, n |=
2. ((1 2 … n)) es una tautología
3. (1 2 … n ¬) es inconsistente

El interés del teorema es que 3. nos proporciona un método para comprar si una fórmula es consecuencia lógica de unas premisas. Si la negación de una FBF produce una FBF inconsistente al ponerlo en conjunción con una conjunción de FBFs, quiere decir que la FBF es una consecuencia lógica de las otras.

Lógica de predicados

Introducción

Las fórmulas atómicas solo representan proposiciones simples, no permite acceder a los elementos de la proposición. La **lógica de primer orden o lógica de predicados** incluye el concepto de **término**, un componente de la fórmula atómica que referencia a los elementos que las forman.

Si se consideran las sentencias "Confucio es un hombre" y "todos los hombres son mortales", en lógica proposicional se representaría como "p ^ q". En lógica de predicados, podría representarse como "H(c) ^ para todo x tal que H(x) -> M(x)", permitiendo deducir que "M(s)" (Sócrates es mortal) si introducimos "H(c)" (Sócrates es un hombre), cosa que no podríamos haber deducido en lógica proposicional.

Lenguaje de la lógica de primer orden

Sintaxis

Las fórmulas del lenguaje se construyen con los siguientes **símbolos**:

- Constantes: Mayúsculas, palabras comenzando por una mayúscula.
- Funciones: Minúsculas con un entero asociado denominado grado o aridad que indica la cantidad de argumentos que tomará el símbolo de función, pero no suele representarse.
- Predicados: Mayúsculas, palabras en mayúsculas. También tienen asociado un grado o aridad.
- Variables: Minúsculas, puede que con subíndice para identificarlas más fácilmente.
- Conectores lógicos: $\neg$, , , ,
- Cuantificadores: Cuantificador universal (para todo) y cuantificador existencial (existe) .
- Símbolos auxiliares: Paréntesis y coma.

Un **vocabulario** es una cuádrupla donde C son los símbolos de constante, F los de función, P los de predicado y d la función que da la aridad de los símbolos de función y predicado. Los elementos de cualquiera de los conjuntos C, F y P no pueden estar en cualquier otro de esos conjuntos. Además, suponemos que no contienen símbolos de variable, conectores lógicos, cuantificadores o símbolos auxiliares.

> Una definición más adecuada elimina el conjunto C y considera los símbolos de constante como símbolos de función de grado 0.

Los **términos** de un vocabulario son sus constantes, variables, y funciones con tantos términos como parámetros como aridad tengan. Los términos denotan objetos. Una constante referencia siempre al mismo elemento, una variable representa a un elemento concreto dependiendo del contexto, y una función referencia a un elemento concreto de forma indirecta.

Las **fórmulas atómicas o átomos** son predicados con sus términos. Expresan relaciones entre los objetos que representan sus términos. Por ejemplo, la fórmula atómica H(s) puede significar que Sócrates es un hombre, y la fórmula PADRE(pepe, ana) puede significar que Pepe es el padre de Ana.

Una **fórmula bien formada** es una fórmula atómica, la negación de una FBF, la relación con conectores lógicos de dos FBFs, o la aplicación de cuantificadores a una FBF. Un conjunto de FBFs forman un lenguaje de primer orden sobre W denominado L^1_w, o L^1 si W es fijo.

El **alcance** de un cuantificador Q sobre una FBF $Qx\alpha$, es α.

Una ocurrencia de una variable está **ligada** si y solo si es la ocurrencia del cuantificador o está en el alcance de un cuantificador sobre esa variable.

Una ocurriencia de una variable es **libre** si no está ligada.

Semántica

Una **interpretación** de un vocabulario es un par formado por el dominio o universo de discurso y una **función de interpretación**, que se define como:

A partir de una interpretación se define de forma única una **función de evaluación de términos y átomos** de la siguiente forma:

Una **interpretación modificada**

Evaluación de términos y fórmulas atómicas modificada

A partir de una función de evaluación de términos y fórmulas atómicas se define de forma únca una **función de evaluación de FBFs** de la siguiente forma:

Se dice que **una FBF es cierta bajo una interpretación** o **una interpretación satisface una FBF** si y solo si la evaluación de la FBF a partir de una interpretación es verdadera. Si no, la FBF es **falsa** bajo esa interpretación.

Una **sentencia o fórmula cerrada** es una FBF en la que no hay ocurrencias de variables libres.

Modelo, consistencia y validez

Una interpretación es un **modelo** de una sentencia si y solo si la evaluación de esa sentencia bajo esa interpretación es verdadera. Es modelo de un conjunto finito de sentencias si y solo si es modelo de cada una de las sentencias.

Una sentencia es **consistente o satisfactible** si y solo si tiene un modelo. Un conjunto finito de sentencias lo es si tiene un modelo.

Una sentencia es **inconsistente o insatisfactible** si y solo si no es consistente. Un conjunto finito de sentencias es inconsistente si y solo si no es consistente.

Una sentencia es **válida o tautológica** si y solo si es cierta bajo todas las interpretaciones del lenguaje.

Satisfactibilidad

Una lógica es **semi-decidible** si el problema de la satisfacibilidad no es computable en dicha lógica, pero se puede dar un procedimiento de cómputo que para cualquier conjunto finito de sentencias inconsistente termine indicando su inconsistencia. Por tanto, no se garantiza que el proceso termine para cualquier conjunto finito de sentencias. La lógica de primer orden es semi-decidible.

En la práctica, un procedimiento para comprobar la inconsistencia de un conjunto finito de sentencias puede terminar normalmente indicando la consistencia o inconsistencia o terminar porque se queda sin recursos. En este último caso, el conjunto de sentencias puede ser inconsistente o no.

Equivalencia

Dos sentencias son equivalentes si tienen los mismos valores de verdad para cualquier interpretación de un vocabulario.

Las leyes de equivalencia son las siguientes:

- Eliminación del bicondicional: conjunción de condicionales.
- Eliminación del condicional: ( ) = (¬)
- Conmutativa de la conjunción y la disyunción.
- Asociativa de la conjunción.
- Distributiva de disyunción respecto de conjunción y viceversa.
- Absorción: La conjunción de una FBF con otra válida es equivalente a la primera FBF. La disyunción con una inconsistente también.
- Contradicción: La conjunción de una FBF con su opuesto o con una FBF inconsistente es equivalente a una FBF inconsistente.
- Exclusión de los medios: La disyunción de una FBF con su opuesto o con una FBF válida es equivalente a una FBF válida.
- Idempotencia: La conjunción o disyunción de una FBF consigo misma es equivalente a la FBF.
- Doble negación.
- De Morgan: Negación de la conjunción, disyunción de negaciones. Negación de la disyunción, conjunción de negaciones.
- La conjunción o disyunción de una FBF sin ocurrencias libres de `x` con una FBF cuantificada es equivalente a la cuantificación de la conjunción o disyunción de la primera FBF con la FBF cuantificada.
- De Morgan con cuantificadores: La negación de un cuantificador es equivalente al cuantificador contrario aplicado a la negación de la FBF cuantificada.
- Distributiva de la conjunción respecto del cuantificador universal y viceversa.

Cambiar FBFs dentro de una FBF por sus equivalentes, o cambiar el nombre de las variables en una FBF genera FBFs equivalentes.

Consecuencia lógica

Una sentencia es **consecuencia lógica** de una serie de premisas si y solo si todo modelo de las premisas es un modelo de la sentencia.

Teorema de refutación

Al igual que con FBFs, pero con sentencias.

Reglas de inferencia

Las **reglas de inferencia** son reglas de manipulación sintáctica que permiten generar nuevas fórmulas a partir de unas fórmulas dadas.

Un **patrón de FBF** es un esquema de FBF que consta de ocurrencias de conectores lógicos, símbolos auxiliares y variables cuyo rango es el conjunto de FBFs. A partir de un esquema se obtienen FBFs remplazando las variables por otras FBFs.

Una **regla de inferencia** es una estructura con la forma `antecedente → consecuente`, donde el **antecedente o premisas** y el **consecuente** son secuencias de patrones de FBF. Una regla de inferencia se puede aplicar si los patrones del antecedente se pueden particularizar a fórmulas de un conjunto finito de FBFs, y si su efecto es obtener las FBFs resultantes de particularizar el consecuente.

Una regla de inferencia es **sólida** si y solo si las fórmulas que permite generar son consecuencia lógica de las fórmulas sobre las que se aplica.

Las reglas de inferencia más comunes son las siguientes:

- Modus Ponens: , →
- Modus Tollens: , ¬ → ¬
- Abducción: , →
- Eliminación and: → ,
- Instanciación universal: x → , donde se obtiene reemplazando las ocurrencias libres de `x` en por un término `t` que sea libre respecto a `x` en , condición que se cumple si y solo si `x` no ocurre, en , en el alcance de un cuantificador de una variable de `t` salvo, quizás, la propia `x`.

Teoría

Dado un conjunto finito de FBFs, un conjunto finito de reglas de inferencia y una secuencia de FBFs, dicha secuencia es una **derivación** de una de las FBFs a partir del conjunto finito de FBFs usando el conjunto de reglas de inferencia si y solo si para todas las FBFs de la secuencia o bien pertenecen al conjunto finito de FBFs, o bien existen fórmulas previas de la secuencia y una regla de inferencia del conjunto que permiten generarla.

La existencia de una **prueba** se denota por Ω |-- R , siendo Ω el conjunto finito de FBFs, `R` el conjunto finito de reglas de inferencia, y la FBF derivada.

Un **axioma propio** es un conjunto finito de FBFs consistente.

Una **teoría de axiomas propios** es el conjunto de axiomas propios de una lógica. La teoría se denomina por $Th(AP)L.$, donde AP son los axiomas propios y L la lógica.

Dada una teoría, una FBF y un conjunto de reglas de inferencia, esa FBF es un **teorema** de la teoría usando las reglas de inferencia si la FBF es una prueba de la teoría.

Una teoría es **sólida** si y solo si todos sus teoremas son consecuencia lógica de sus axiomas propios, y es **completa** si y solo si todas las consecuencias lógicas de sus axiomas propios son teoremas de la teoría.

Unificación

La **unificación** es un procedimiento que permite comprobar si, dadas dos fórmulas cuantificadas universalmente, la regla de inferencia de instanciación universal permite obtener otras dos fórmulas no cuantificadas de modo que éstas sean sintácticamente iguales. Esta comprobación suele ser un paso previo a la aplicación de cualquier regla de inferencia.

Sustituciones

Una **ligadura** es un par ordenado formado por un término y una variable que no ocurra en dicho término. Se representa como `t / x`.

Una **sustitución** es un conjunto finito de ligaduras tales que no hay dos variables (segundo término de la ligadura) iguales, y ninguna variable (de las del segundo término) ocurre en ningún término.

Una **expresión** es un término o una FBF.

Una **particularización por sustitución** de una expresión mediante una sustitución a la expresión obtenida sustituyendo las variables de la expresión ligadas en la sustitución por los términos de sus ligaduras correspondientes. Si la particularización no contiene variables, se denomina **particularización básica**.

Dos expresiones son **variante alfabética** si y solo si se diferencian en el nombre de las variables.

La sustitución $s2$ sería **distinta** de la sustitución $s1$ si y solo si ninguna variable ligada de $s1$ ocurre en $s2$.

Si $s2$ es distinta a $s1$, se denomina **composición** de $s2$ con $s1$ y se denota $s1s2$ a la sustitución obtenida aplicando $s2$ a los términos de $s1$ y añadiendo al conjunto resultante las ligaduras de $s2$

Unificador más general

La **particularización del conjunto finito de expresiones** `C` por la sutitución `s` se denomina `Cs` y se define como la aplicación de `s` a cada expresión de `C` eliminando las particularizaciones repetidas.

`s` es un **unificador** de `C` si y solo si `Cs` tiene un único elemento. En ese caso, se dice que `C` es **unificable**.

Siendo `g` un unificador de `C`, `g` es el **unificador más general** si y solo si para cualquier otro unificador de `C` existe una sustitución que al ser compuesta con `g` es equivalente a ese unificador.

Teorema de unificación: si un conjunto finito de expresiones es unificable, su unificador más general existe y es único salvo variantes alfabéticas.

Algoritmo de unificación

El **conjunto de desacuerdos** de un conjunto finito de expresiones se obtiene localizando el primer símbolo por la izquierda en el que no todas las expresiones del conjunto tienen el mismo símbolo, y extrayendo de cada expresión la subexpresión que comienza por el símbolo que ocupa dicha posición.

El **algoritmo de unificación** tiene como entrada un conjunto finito de expresiones `Ω`, y como salida, el unificador más general de dicho conjunto o fallo si el conjunto no es unificable. El algoritmo es:

1. Hacer `k = 0, Ω_k = Ω, s_k = 0`.
2. Si `Ω_k` tiene más de un elemento, crear su conjunto de desacuerdos. En caso contrario, devolver `s_k`, que es el unificador más general del conjunto de entrada.
3. Si no existen `v_k, t_k` en el conjunto de desacuerdos tales que `v_k` no ocurra en `t_k`, devolver fallo pues `Ω` no es unificable.
4. Elegir `v_k, t_k` tales que `v_k` no ocurra en `t_k` y hacer `s_{k+1} = s_k·{t_k / v_k}` y `Ω_{k+1} = Ω_k · {t_k / v_k}`
5. Hacer `k = k+1` e ir a 2.

Estrategias de resolución

Introducción

La utilidad de la lógica simbólica radica en la representación de problemas mediante fórmulas bien formadas (a través de teorías, conjuntos de FBFs consistentes) y en solucionar problemas de forma deductiva usando reglas de inferencia de la lógica y el proceso de búsqueda en el espacio de las FBFs.

La dificultad es que la aplicación no controlada de reglas de inferencia da lugar a un problema de explosión combinatoria. El número de FBFs generadas crece exponencialmente.

Refutación por resolución

Los **métodos uniformes de demostración** utilizan una única regla de inferencia, siendo una forma de reducir la complejidad de la búsqueda, pero requieren transformar las fórmulas a un formato estándar.

Existen dos aproximaciones básicas:

- **Sistemas de resolución**: Hay que transformar las FBF a forma de cláusula y aplicar la resolución hasta generar la cláusula vacía. Se reduce la complejidad sin limitar la capacidad de representación seleccionando las estrategias adecuadas.
- **Programación lógica**: Solo admite cláusula de Horn. Se realiza a través de la resolución SLD. Reduce la complejidad limitando la capacidad de representación a las cláusulas de Horn.

La **refutación por resolución** dice que para probar que una sentencia es consecuencia lógica de una serie de premisas (`Ω |=  `), hay que probar que la unión de las premisas con la negación de la sentencia (`Ω  ¬`) es inconsistente. Usa por tanto una única regla de inferencia: la resolución.

El procedimiento para la refutación por resolución es el siguiente:

Sea `T` una teoría sólida y completa (axiomas), y `t` una FBF (teorema). Para probar que existe una teoría `T` que prueba `t` (`T  t`) mediante refutación por resolución:

1. Convertir los axiomas de `T` a forma normal conjuntiva (FNC) y crear el conjunto `S_0` como la conjunción de todas las cláusulas obtenidas.
2. Negar `t` y convertir a FNC. Añadir las cláusulas obtenidas a `S_0`, obteniendo `S`.
3. Repetir hasta obtener cláusula vacía o no se generen nuevas cláusulas; seleccionar dos cláusulas que se puedan resolver (buscar su unificador más general y obtener la resolvente). Si el resolvente no es cláusula vacía, añadir a `S`.

La lógica proposicional es decidible así que el algoritmo siempre termina, ya sea generando cláusula vacía y por tanto `S` es inconsistente y `t` es consecuencia lógica de `S_0` y de la teoría `T`, o bien no se generan neuvas resolventes en cuyo caso `S` es consistente y `t` no es consecuencia lógica de `S_0` ni `T`.

La lógica de primer orden es semi-decidible, puede que el cómputo de nuevas resolventes no termine y el proceso finalice por consumo de recursos. Por tanto, si el cómputo termina las conclusiones son las mismas que para la lógica proposicional. Pero si termina por consumo de recursos, no sabemos nada y ambas opciones serían posibles.

Estrategias de resolución

La generación incontrolada de cláusulas hace que estas crezcan de forma exponencial. Por tanto se hace necesario planear estrategias que eviten esta explosión combinatoria.

Se puede simplificar (eliminando o reemplazando), dirigir (eligiendo la siguiente cláusula a considerar) o restringir (evitando la generación de resolventes).

Una estrategia de resolución es **completa** para la refutación si y solo si, usada con una regla de inferencia completa, garantiza que encuentra una derivación de la cláusula vacía a partir de una forma clausulada inconsistente. La regla de resolución es completa para la refutación.

Estrategias de dirección: saturación por niveles

Siendo `S` el conjunto de todas las cláusulas de partida, `S =S`. Si `k` pertenece a `S`, `k` es una cláusula de nivel 0. Los sucesivos conjuntos `S^i` (niveles) se forman con la resultante entre una cláusula de los niveles previos y otra del nivel inmediatamente anterior. La estrategia es obtener primero todas las resolventes de un nivel antes de obtener resolventes del siguiente. Es completa pero ineficiente.

Estrategias de simplificación

Eliminación de literales puros Un literal es un **literal puro** si y solo si no existe ningún otro literal negado que unifica con él. Es decir, `L` es un literal puro si y solo si no existe ningún `¬L'` tal que `L` y `L'` unifiquen.

La **estrategia de eliminación de literales puros** consiste en eliminar todas las cláusulas que contengan literales puros, solo al conjunto inicial. Es una estrategia completa.

Eliminación de tautologías Las cláusulas tautológicas no afectan a la satisfactibilidad, por lo que la **estrategia de eliminación de tautologías** consiste en eliminar dichas cláusulas, tanto iniciales como las que se vayan generando. Es una estrategia completa.

Eliminación de cláusulas subsumidas Una cláusula **subsume** a otra si y solo si todos los literales que aparecen en ella aparecen también en la otra, bien sin sustituciones o si existe una sustitución que haga cumplir la condición.

La **estrategia de eliminación de cláusulas subsumidas** puede ser hacia adelante si la resolvente puede ser subsumida o hacia atrás si la resolvente puede subsumir.

Es completa cuando se usa con saturación por niveles, pero puede que no lo sea con alguna otra estrategia de restricción. Es muy eficiente, su aplicación suele ser imprescindible.

Asociación de procedimientos Consiste en evaluar funciones o literales básicos sobre un dominio. Afecta a la satisfactibilidad pues estamos fijando una interpretación parcial. No es completa.

Evaluar una función consiste en asociar un símbolo de función con un procedimiento cuya evaluación devuelva un elemento del dominio, reemplazando el término funcional por el elemento del dominio.

Evaluar un literal consiste en asociar un símbolo de predicado con un procedimiento cuya evaluación devuelva un valor de verdad. Si el literal evalúa a un valor verdadero, eliminar la cláusula. Si evalúa a un valor falso, eliminar el literal de la cláusula.

Estrategias de restricción

Estrategia del conjunto soporte Un **conjunto soporte** es un subconjunto de una forma clausulada distinto de ésta y distinto del conjunto vacío.

Un **conjunto soporte de nivel i-ésimo** es el conjunto soporte de las resolventes...

Una cláusula tiene **T-soporte** si dicha cláusula pertenece a un conjunto soporte de algún nivel.

La **estrategia del conjunto soporte**...

Teorema del conjunto soporte Si tenemos una fórmula clausulada inconsistente y un conjunto soporte de ella tal que si quitamos el conjunto soporte de la fórmula clausuladoa obtenemos una fórmula clausulada consistente, se puede llegar a la cláusula vacía usando la estrategia del conjunto soporte con ese conjunto soporte.

Resolución lineal Se elige una cláusula central de partida y solo permite obtener como resolventes otras cláusulas centrales que sean resolvente de la cláusula central anterior y otra distinta.

Teorema de la complitud de la resolución lineal Existe una prueba de que una fórmula clausulada inconsistente puede derivar a cláusula vacía si se usa la estrategia de resolución lineal con la cláusula que hace inconsistente a la fórmula como cláusula central de partida.

Resolución por entradas Las **cláusulas de entrada** son las del conjunto base. Las **resolventes de entrada** son aquellas con al menos un padre cláusula de entrada.

La **estrategia de resolución por entradas** solo permite obtener resolventes de entrada. No es completa.

Resolución unitaria Un **resolvente unitario** es aquel en el que al menos una de las cláusulas padre es unitaria.

La **estrategia de resolución unitaria** solo permite obtener resolventes unitarios. No es completa.

Teorema de equivalencia de la resolución unitaria y por entradas Se puede llegar a la cláusula vacía a través de la resolución unitaria si y solo si se puede llegar con la resolución por entradas.

Cláusulas de Horn Una **cláusula de Horn** es aquella que tiene, a lo sumo, un literal positivo. Por tanto, se pueden interpretar como implicaciones con el literal positivo a la derecha del símbolo de implicación.

Teorema complitud resolución por entradas (unitaria)

Procedimiento de extracción de respuesta

Consiste en extraer la respuesta mediante la refutación por resolución. Es el proceso de encontrar los elementos del dominio que hacen cierto el teorema a demostrar.

Las **preguntas** son, en general, cualquier FBF; pero por motivos prácticos se usan sentencias en forma normal prenexa (FNP) con una matriz en forma de conjunción de literales y un prefijo que contiene solo cuantificadores existenciales (`x  y(P(x)  Q(x, y))`). De esta forma, la negación de estos teoremas da lugar a una única cláusula.

Un **literal respuesta** es la respuesta para todas las variables que ocurren en una pregunta. Para obtener la respuesta hay que negar la pregunta para transformarlo en cláusulas, formar la disyunción de las cláusulas obtenidas con el literal respuesta y buscar derivaciones de cláusulas que solo contengan literales respuestas.

La respuesta puede contener más de un literal y no es única, sino que depende de la derivación que la produce.

Demostradores de teoremas

Son programas que dada una teoría y una FBF intentan comprobar si existe un conjunto de axiomas propios que demuestran dicha FBF. Generalmente aceptan FBFs de la lógica de primer orden como entrada. Suelen estar basados en refutación por resolución. Aplican estrategias de control para limitar la búsqueda.

Prolog Technology Theorem Prover (PTTP)

Búsqueda con descenso iterativo. Completa. La inferencia es completa con la regla de resolución lineal y por entradas. Se implementa una rutina para probar `P` y otra para probar `¬P` (negación lógica). Reintroduce chequeo de ocurrencias en la unificación.

Organized Techniques for Theorem-proving and Effective Research (OTTER)

Es uno de los primeros y más populares demostradores de teoremas y aplica refutación por resolución. Se usa en investigación matemática y verificación de hardware.

Como entrada recibe hechos importantes sobre el dominio (conjunto de soporte, etc), conocimiento sobre el problema (axiomas de utilidad), demoduladores (reglas de reescritura) y parámetros y cláusulas que definen la estrategia de control.

Programación lógica y prolog

Introducción a la programación lógica

La **programación lógica** es la parte de la informática que se ocupa de la lógica como lenguaje de programación.

En este paradigma, un **programa** es un conjunto finito de FBFs, y la **computación** es la obtención de pruebas formales.

Una **cláusula definida** es aquella que tiene, a lo sumo, un literal positivo. En programación lógica se representa como `a <-- b1, b2, ..., bn` siendo `a` el único literal positivo de haberlo. Los `b` representan los literales que eran negativos en la fórmula clausal, pero se representan sin la negación en programación lógica. Además, todas las variables se consideran cuantificadas universalmente.

La parte izquierda de la flecha se denomina **cabeza** de la cláusula, y la parte derecha es el **cuerpo** de la cláusula.

Caracterización de programas lógicos

- Programas definidos: Cláusulas de Horn o definidas.
- Programas normales: Cláusulas normales como extensión de la cláusula de Horn admitiendo literales negativos en el cuerpo de las cláusulas.
- Programas: Cualquier FBF.

Programas definidos: sintaxis de Edimburgo

- Términos: constantes numéricas o atómicas, variables y funciones.
- Átomos: funciones con sus parámetros.
- Cláusulas: Construidas con átomos, se componen de una cabeza y un cuerpo separados con una flecha.

Si una cláusula solo tiene cabeza, se denomina **hecho** (cláusula unitaria de programa). Si tiene cabeza y cuerpo, es una **regla** (cláusula de programa). Si no tiene cabeza, se trata de una **pregunta o meta**.

Un **programa** es un conjunto finito de cláusulas de programa.

Resolución SLD

SLD: Resolución lineal con función de selección para cláusulas definidas.

Si tenemos una cláusula de programa con el literal (el único positivo) `a` a la izquierda de la flecha, y una pregunta con uno o más `a_n` en el cuerpo y sin variables en común con la cláusula de programa, seleccionamos un literal de la pregunta a través de una función de selección. Si el literal seleccionado y `a` unifican con un umg, el **resolvente SLD** del programa y la pregunta es la meta en cuyo cuerpo están todos los literales de la pregunta hasta el seleccionado, el cuerpo del programa y el resto de los literales de la pregunta (sin el seleccionado) compuestos con el umg.

O, dicho de otra forma: se escoge con una función de selección (regla de cómputo) un átomo de la pregunta. Se unifica con alguna cabeza de las cláusulas de programa. La resolvente genera una nueva pregunta consistente en la pregunta anterior sin el átomo unificado más el cuerpo de la cláusula de programa con la que se unificó, aplicando el umg que se usó para unificar los dos átomos.

Derivación SLD (Cómputo de G por P)

Sean `P` un programa y `G` una meta. una **derivación SLD de `P U {G}`** consiste en tres secuencias, posiblemente infinitas, de metas, cláusulas de P renombradas y umg's de una cláusula renombrada con la meta de la cláusula previa; tal que la meta de la cláusula siguiente es el resolvente SLD de la meta actual y la cláusula de P siguiente, usando el siguiente umg.

La **refutación SLD** es la derivación SLD de la cláusula vacía.

Intérprete abstrácto de un programa lógico

Trata de unificar el átomo elegido de la meta con la cabeza de cláusulas renombradas del programa. Si alcanza la cláusula vacía, devuelve la meta con los unificadores aplicados. Si no puede unificar con ninguna cabeza, devuelve fallo.

Este tipo de intérpretes tienen que elegir la regla de cómputo (el literal sobre el que se resuelve, dado por la función de selección; por ejemplo, primer literal a la izquierda) y la regla de búsqueda (criterio de selección de la cláusula que resuelve o reduce la meta; por ejemplo, primera cláusula no utilizada o búsqueda primero en profundidad). La regla de cómputo es arbitraria, no afecta a la terminación, como mucho al orden de las respuestas. La regla de búsqueda no es determinista y sí afecta a la terminación.

Concepto de respuesta

Una **respuesta para `P U {G}`** es una sustitución para las variables de `G` o "No".

Una respuesta es **correcta** si y solo si es consecuencia lógica del programa. "No" también es respuesta correcta si y solo si `P U {G}` es satisfactible.

Una respuesta es **computada** si y solo si es la sustitución obtenida seleccionando de todas las sustituciones intermedias las ligaduras de las variables que ocurren en `G`.

Teorema de solidez de la resolución SLD

Toda respuesta computada de la unión de un programa con una meta definida también es correcta.

Teorema de complitud de la resolución SLD

La respuesta computada puede ser más general que la correcta.

Diferencias entre la respuesta correcta y la computada

Puede que no se computen respuetas pero sí haya respuestas correctas.

Teorema de la independencia de la regla de cómputo

Si se usan reglas de cómputo distintas, las respuestas obtenidas son variantes alfabéticas.

Intérprete abstracto de un programa lógico

Concepto de respuesta

Programación lógica y negación

Un **programa definido** es un conjunto de hechos y reglas que describen explícitamente qué es cierto, sin información explícita sobre qué es falso.

Una implementación práctica: prolog estándar

Prolog implement a un modelo de programación lógica de forma secuencial. Sirve para programas normales. Usa como regla de cómputo el primer literal a la izquierda, y como regla de búsqueda el primero en profundidad (con backtracking).

Prolog no es completo por la regla de búsqueda que usa. Tampoco es sólido porque SLDNF (al usar programas normales y no programas definidos) no lo es. Por eso es bueno evitar variables libres en literales negativos seleccionados no incluyéndolos o intentando forzar la ligadura operacionalmente.

Prolog se desvía del modelo lógico porque no comprueba ocurrencias, da respuestas computadas no correctas y puede producir bucles infinitos. Introduce el operador de corte, que impide explorar un subárbol del árbol SLD pero afecta a la complitud de programas definidos y normales y a la solidez de los normales.

Anexo: árbol SLD

Representación del conocimiento: Introducción (PDF)

Sistemas basados en conocimiento

Son sistemas con una representación explícita y simbólica del conocimiento para resolver un problema. Además, el comportamiento del sistema debe ser consecuencia del contenido de la base del conocimiento. Si usan conocimiento experto, se denominan sistemas expertos.

Un sistema basado en conocimiento es un sistema informático con, al menos, dos componentes estructurales:

- Base de conocimiento: representación explícita y natural del conocimiento del sistema.
- Motor de inferencias: componente de control responsable de utilizar el conocimiento para la solución del problema.

Otros componentes habituales son un editor de conocimiento o un módulo de explicación.

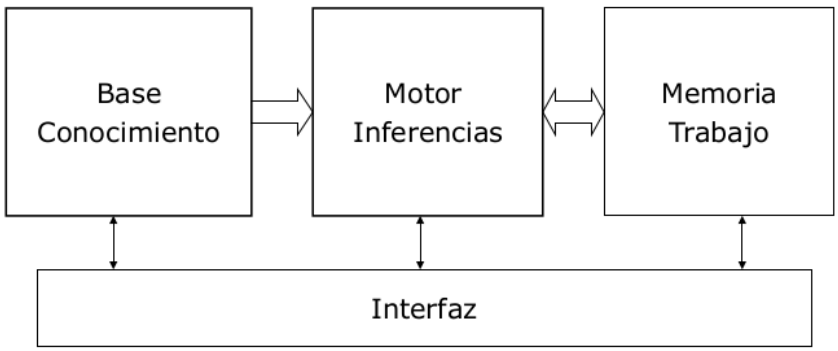


Figure 2: Arquitectura de un sistema basado en conocimiento

Tipos de conocimiento

Se pueden analizar según varios puntos de vista: biológico, límites de aplicación, niveles o contenidos.

Biológico

- Fisiológico
  - Necesario para sobrevivir en el medio.
  - Somatizado, sistema nervioso.
  - Reaccionar a estímulos, constantes vitales…
- Automatismos
  - Consecuencia de la interacción con el medio.
  - Requiere aprendizaje.
  - Comunicación, equilibrio, reflejos…
- Cortical
  - Necesidad de comprender y explicar, curiosidad.
  - Gran cantidad de aprendizaje y asimilación de experiencia pasada.
  - Elemento característico: lenguaja hablado y escrito.

Límites de aplicación

Dependiente del dominio Es el conocimiento necesario para resolver un problema concreto y no es extrapolable a otro tipo de problemas.

Incluye datos, hechos, relaciones, criterios de decisión…

Un ejemplo es la diagnosis de enfermedades pulmonares: - Datos: capacidad pulmonar total, capacidad vital forzada, volumen residual… - Síntomas: obstrucción moderada de las vías respiratorias. - Diagnóstico: asma. - Establece relaciones entre ellos.

Independiente del dominio Componentes del conocimiento que tienen que ver con la representación, uso y modificación del conocimiento con independencia del contenido del mismo. Es aplicable a clases de problemas.

Un ejemplo sería que si no hay suficientes datos para establecer un diagnóstico, sigue acumulando datos para llegar a una solución.

Niveles

Reune conocimiento del dominio como objetos y relaciones de un dominio particular, junto con metaconocimiento (conocimiento sobre el conocimiento) que permite aplicar de modo efectivo el conocimiento a la resolución de problemas, reconocer las limitaciones del conocimiento o explicar a un observador externo el proceso de solución de un problema.

Nivel-1: Estratégico

- Si con los síntomas disponibles son posibles varios diagnósticos, examinar primero los menos costosos de comprobar.
- Si el paciente es inmunodepresivo, investigar primero infecciones de lugares no estériles.

Contenido (Clancey, 85)

- Heurístico: meras asociaciones entre hechos y conclusiones.
- Estratégico: conocimiento sobre los pasos a seguir para solucionar un problema.
- Estructural: información sobre la estructura del dominio, como jerarquías (taxonomías, parte-de) o componentes e interconexiones.
- Soporte: información relevante para la comprensión del conocimiento de un sistema, como teoría subyaciente, modelos, referencias bibliográficas, justificaciones, autor, fecha, etc.

Tipos de representación

Históricamente encontramos representaciones declarativas ("sabe qué") y representaciones operacionales ("sabe cómo").

- Declarativa: el conocimiento está compuesto por un conjunto de secuencias más procedimientos que las manipulan.
- Operacional: conjunto de procedimientos que permiten resolver el problema.

En la prática: espectro continuo.

Lenguajes de representación de conocimiento

Todo lenguaje de representación del conocimiento (LRC) debe proporcionar:

- Lenguaje de representación con sintaxis y semántica precisa.
- Capacidad de inferencia.

Las principales aproximaciones son:

- Lógica simbólica.
- Sistemas de producción.

- Métodos estructurados como redes semánticas y marcos.
- Actualmente la tendencia son las ontologías.

Es deseable que el lenguaje proporcione representación y gestión de la incertidumbre.

Representación del conocimiento: Lógica y representación del conocimiento (PDF)

Papel de la lógica en la representación del conocimiento

Intuitivamente, la lógica es atractiva como LRC por tener una sintaxis bien definida, una semántica precisa y mecanismos formales de deducción. Sin embargo, presenta dificultades como el problema de la cualificación, la dinámica, la incertidumbre y las creencias.

Problema de la cualificación

El problema de la cualificación se refiere a la limitación de las sentencias cuantificadas universalmente. La sentencia `x (Ave(x) Vuela(x))` es cierta en algunos dominios pero presenta numerosas excepciones. Añadir estas excepciones (`x (Ave(x) ¬Pingüino(x) Vuela(x))`) es una solución ad hoc que no es generalizable. Este problema no tiene solución en lógica clásica como LPO.

Debate histórico

Históricamente, hay una escuela "logicista" que defiende la inteligencia artificial como la aplicación de la lógica (Nilson, McCarthy) enfrentada a los que defienden la lógica como elemento de análisis pero no de resolución de problemas, al menos los complejos (Misky, Newell).

Cuena [99] defiende que la lógica permite formalizar y mecanizar el razonamiento deductivo, y que permite modelar el conocimiento de un agente con independencia de su implementación.

Tendencia actual

Se usa la lógica como lenguaje de modelado del conocimiento. Se ha desarrollado el concepto de **ontología** en el ámbito de la representación del conocimiento. Algunas definiciones de ontología son:

- Sistema particular de categorías sistematizando cierta visión del mundo. [Guarino 98]
- Teoría particular de la naturaleza del ser o de la existencia. [Rusell, Norvig 2010]
- Una ontología es una especificación formal de una conceptualización compartida. [Struder 98]

Se usa la lógica para crer ontologías.

2. Principios de ingeniería de conocimiento en lógica de primer orden

La ingeniería del conocimiento es el proceso de construcción de una base de conocimiento. Requiere adquirir el conocimiento, formalizarlo, representarlo e implementarlo.

En un LPO, la formalización proporciona la representación y la implementación.

Un ingeniero del conocimiento es un profesional que desarrolla la base de conocimiento. Requiere conocer suficientemente el dominio, conocer los lenguajes de representación, conocer los mecanismos de inferencia y conocer los aspectos del diseño y la implementación.

El requisito de conocer suficientemente el dominio es posible en dominios sencillos, pero generalmente requiere la colaboración de expertos del dominio y establecer un proceso de adquisición.

Proceso de ingeniería del conocimiento en LPO Está orientado al dominio y a la tarea (ontología específica del dominio y de la tarea).

Es un proceso iterativo:

1. Identificar la tarea. Hay que identificar el rango de cuestiones que la base de conocimiento debe soportar. Por ejemplo, un asistente al diagnóstico podría responder preguntas como "¿qué bombillas lucen?" o "¿qué interruptor está estropeado?". Hay que identificar también los tipos de hechos disponibles para describir instancias específicas del problema. En el asistente al diagnóstico tenemos componentes, conexiones, tensión en los cables…
2. Reunir el conocimiento relevante. Puede que el ingeniero del conocimiento sea experto del dominio, pero por lo general hay que trabajar con un experto en un proceso de adquisición del conocimiento. Hay que obtener una descripción informal que permita entender cómo funciona el dominio y apreciar el alcance de la base de conocimiento para hacerse una idea de qué elementos hay que representar. La dificultad de este paso aumenta con la complejidad del dominio de aplicación.

3. Elaborar un vocabulario de nombres de predicados, funciones y constantes (a veces erróneamente denominado ontología). Este vocabulario especifica qué individuos y relaciones existen, pero no sus propiedades e interrelaciones, aunque es fundamental para describirlas posteriormente. En el ejemplo, las bombillas se representan con constantes `L1` y `L2` y las conexiones con el predicado `Connected/2`.
4. Codificar el conocimiento general del dominio. Hay que elaborar los axiomas del dominio, que proporcionan las propiedades de los elementos de la ontología (en otras palabras, proporciona el significado de los símbolos de la ontología). En el ejemplo del asistente de diagnóstico, se especifica el funcionamiento de las bombillas con el axioma `x [(Light(x) OK(x)) (Live(x) Lit(x))]` y el funcionamiento de las conexiones con `x y [(Connected(x,y) Live(y)) Live(x)]`.
5. Codificar una descripción de una instancia específica del problema. Si la ontología es adecuada, se limita a proporcionar un conjunto de sentencias atómicas sobre instancias de conceptos. Juegan un papel similar a los datos de entrada de un programa tradicional pero en este caso, por ejemplo, los proporcionan los sensores de un agente. En el ejemplo del asistente al diagnóstico sería `up(s3). live(outside). ok(l1).`.
6. Plantear preguntas al procedimiento de inferencia y obtener respuestas. El procedimiento de inferencia usa los axiomas y hechos específicos del problema para derivar hechos que nos interesan. Siguiendo el ejemplo, algunas preguntas pueden ser `x Lit(x)`, `x Live(x)` o `x y Connected(x,y)`.
7. Depurar la base de conocimiento.

Ejemplo de desarrollo de una ontología específica y base de conocimiento en el dominio de los circuitos digitales

Representación de conocimiento: Sistemas Basados en Reglas (SBR) (PDF)

1. Introducción

Los **sistemas basados en reglas** o **sistemas de producción** se caracterizan por utilizar una única estructura para representar el conocimiento: la **regla**.

El esquema de la regla está compuesto por dos partes que se pueden denominar de distinta manera:

- condición -> acción
- antecedente -> consecuente
- lado izquierdo -> lado derecho (LHS -> RHS)
- si … entonces …

El interés por este tipo de sistemas radica en que es un formalismo adecuado para representar conocimiento heurístico a través de asociaciones entre los elementos del antecedente y del consecuente. Existen numerosos dominios (parcialmente) gobernados por reglas deterministas.

Es un acercamiento popular por su sencillez, ya que usa una única estructura, y por su eficiencia, pues sus mecanismos deductivos son computacionalmente más eficientes que los de la lógica.

2. Componentes de un sistema de producción

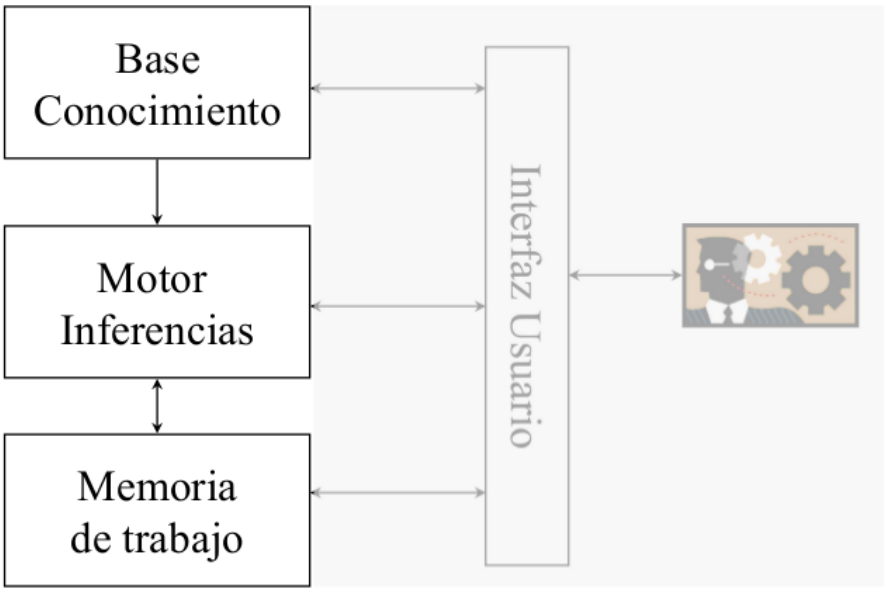


Figure 3: Diagrama de los componentes de un sistema de producción

- Base de conocimiento (BC o KB).
  - Base de reglas (BR): conjunto de reglas de producción.
  - Declaración de dominio: declaración de los elementos básicos que se referencian en hechos y reglas.
- Memoria de trabajo (MT o WM). Conjunto de hechos. Información que se considera cierta.
- Motor de inferencias (MI o IE). Genera nuevos hechos a partir de la memoria de trabajo usando la base de reglas.

3. Lenguajes

Los elementos básicos de la representación son los **hechos**, que representan información que se posee relevante a una instancia o problema concreto, y las **reglas**, que es el conocimiento sobre el dominio del problema que permite derivar hechos adicionales.

Existen variantes de los lenguajes según cómo se referencien los elementos del dominio (tripletes objeto-atributo-valor, patrones simbólicos, etc), según el funcionamiento del motor de inferencias (encadenamiento hacia adelante o hacia atrás) o según otras variables como la incertidumbre, el metaconocimiento, etc.

Lenguaje objeto-atributo-valor

El universo se conceptualiza en tripletes objeto, atributo y valor. El objeto representa aquello de lo que queremos hablar, el atributo una propiedad del objeto, y el valor es el valor de dicha propiedad. En un hecho, se establece el valor del atributo de un objeto. En una regla, se usan para derivar nuevo conocimiento.

Declaración de dominio

Se puede dar un conunto definido de objetos (**declaración de objetos**) y una **declaración de atributos** (DA) que puede ser univaluada o multivaluada según si ese atributo puede tener más de un valor a la vez, y tipada o no tipada según si los valores que puede tomar son los que forman parte de un conjunto definido.

```
| Univaluada          | Multivaluada
```

Tipada	$O_i.a_j^s : \tau$	$O_i.a_j^m : 2^\tau$
No tipada	$O_i.a_j^s$	$O_i.a_j^m$

A partir de la declaración de objetos (O) y de la declaración de atributos (DA), la **declaración de dominio** es $DD = O \cup DA$

Ejemplo de declaración de dominio:

$O = \{paciente - 1\}$

$DA = \{paciente-1.sexo^s : \{varn, hembra\}, paciente-1.edad^s : int, paciente-1.sntoma^m, ...\}$

$DD = O \cup DA$

Hechos en objeto-atributo-valor

Univaluado: $O_i.a = c, six^s : \tau, c\epsilon\tau$

Multivaluado: $O_i.a = C, six^m : 2^\tau, C\epsilon\tau$

Ejemplos:

$paciente - 1.sexo = varn$

$paciente - 1.edad = 25$

$paciente - 1.sntoma = dolorPecho, calambresPiernas$

Memoria de trabajo

La **memoria de trabajo** se considera formalmente como un conjunto (sin elementos repetidos, orden irrelevante) que contiene los hechos.

Ejemplo: $H = \{paciente - 1.sexo = varn, paciente - 1.edad = 25, paciente - 1.sintoma = \{dolorPecho, calambresPiernas\}, ...\}$

Reglas en objeto-atributo-valor

```
<reglaProducción>    ::= if <antecedente> then <consecuente> fi
<antecedente>        ::= <disyunción> {and <disyuncion>}*
<disyunción>         ::=<condición> {or <condición> }*
<consecuente>        ::= <conclusión> {also <conclusión>}*
<condición>          ::= <predicado> ( <objeto> , <atributo>, <constante> )
<conclusión>         ::= <acción> (<objeto> , <atributo>, <constante> )
<predicados>         ::= iguales | noiguales | mayorque | ...
<acción>             ::= añadir | eliminar | ...
```

Figure 4: Sintaxis BNF

Una regla en lenguaje natural podría ser:

```
Si el paciente tiene un dolor abdominal y
    en la auscultación se percibe un rumor abdominal y
    se siente una masa pulsante en el abdomen
entonces el paciente tiene un aneurisma aórtico.
```

Una regla de producción equivalente sería:

```
if iguales(paciente-1, síntoma, dolorAbdominal) and
    iguales(paciente-1, evidencia, rumorAbdominal) and
```

```
    iguales(paciente-1, evidencia, masaPulsante)
then añadir (paciente-1, enfermedad, aneurismaAortico)
fi
```

Condiciones

Las **condiciones** se construyen con un predicado y sus argumentos: objeto, atributo y constante. Por ejemplo, `iguales(paciente, edad, 60)`.

Expresa una comparación (según el predicado) entre la constante especificada y el valor asociado al atributo del objeto en la memoria de trabajo.

Un predicado es una función booleana. Si tenemos el hecho `paciente.edad = 50`, el predicado `iguales(paciente, edad, 60)` evaluará a falso.

El antecedente de una regla (`if ...`) se satisface (`then ...`) si dicho antecedente evalúa a cierto.

Semántica de los predicados

La interpretación de los predicados en un sistema de producción es **operacional**.

`iguales(o, a, c)` es cierto si existe algún hecho `o.a = c` en la memoria de trabajo. El comportamiento depende del motor de inferencias: si es hacia adelante, solo se consulta el contenido actual de memoria de trabajo; si es hacia atrás, puede forzar la búsqueda del hecho.

A diferencia con la lógica, el predicado solo es cierto si se puede derivar el hecho necesario.

`noiguales(o, a, c)`. Si es una negación hacia adelante, suele ser una variante limitada de la suposición de mundo cerrado. Es decir, es cierto si no existe `o.a = c` en la memoria de trabajo cuando se evalúa el predicado. Si es negación hacia atrás, suele ser negación por fallo. Es decir, si se ha buscado y ha fracasado, éxito; y si no se ha buscado, fuerza la búsqueda y si ésta fracasa, éxito.

Conclusiones

Tenemos una secuencia de acciones que pueden modificar la memoria de trabajo. Solo se ejecutan si el antecedente de la regla se satisface. Disparar una regla es realizar las acciones de la conclusión.

Acción básica: añadir

Se representa como `añadir(o, a, c)`. Si los atributos son univaluados, tras realizar la acción, la memoria de trabajo contiene `o.a=c` y ningún otro hecho

para `o.a`. Si los atributos son multivaluados, se crea el hecho `o.a={c}` si no existía para `o.a`, y si ya existía, `c` se añade como uno de los valores de `o.a`.

Acción eliminar

Se representa como `eliminar(o, a, c)`. Esta acción no está disponible en un lenguaje lógico. Elimina el hecho de la memoria de trabajo, o el valor si es un atributo multivaluado y hay más de un valor.

4. Inferencia en un sistema de producción

La inferencia en un sistema de producción es, esencialmente, seleccionar reglas cuyo antecedente satisface y realizar su acción. Usan la diferencia entre reglas y hechos: los hechos se gestionan globalmente en la memoria de trabajo (son hechos iniciales que representan conocimiento primitivo), y las reglas se utilizan para derivar nuevos hechos (conocimiento derivado). La inferencia es un proceso de búsqueda que primero examina los hechos y después las reglas.

El concepto de pregunta se sustituye por el de **meta**, es decir, se pregunta por el valor del atributo de un objeto. La declaración de metas es igual que la de atributos pero se añade un subíndice `g`. No es imprescindible declarar la meta si el sistema dispara todas las reglas posibles.

Espacio de búsqueda: hipergrafos o grafos y/o

Un hipergrafo es la generalización del concepto de grafo usando hiperarcos o k-conectores. Nos interesan los hipergrafos dirigidos. Un 1-conector sería un arco de un grafo dirigido convencional, mientras que un k-conector con $k > 1$ generaliza el concepto de arco definiéndose como un arco (n_0, n_1, n_2, … n_{k-1}) donde todos los nodos excepto el primero apuntan al primero.

Los hipergrafos se usan para representar el espacio de búsqueda. Al representar una regla, la disyunción se representa con hiperarcos distintos, mientras que la conjunción se representa con un hiperarco que apunta a la misma meta.

Métodos básicos de inferencia

Control de la búsqueda La búsqueda se controla según la dirección de búsqueda (encadenamiento hacia atrás o hacia adelante), el régimen (tentativo o irrevocable, normalmente adelante es irrevocable y hacia atrás es tentativo), si la búsqueda es primero en anchura o en profundidad, las estrategias de resolución de conflictos, o el orden de evaluación de las premisas.

Encadenamiento hacia adelante Las reglas se interpretan de forma directa. Es decir, si se cumple el antecedente, se añade el consecuente a la memoria de trabajo.

Se parte del conjunto de hechos iniciales, se generan nuevos hechos disparando reglas y se para cuando se alcanza la meta o no hay reglas activadas.

Es útil si se dispone de suficientes datos inicialmente, no hay una meta clara, o se adapta bien a la naturaleza de la tarea.

El régimen de control es irrevocable, los algoritmos de búsqueda son simples y no generan explícitamente grafos de búsqueda. El motor de inferencias itera sobre el ciclo básico reconocimiento-acción. Se ejerce un control adicional a través de las estrategias de resolución de conflictos y el orden de las premisas.

El ciclo básico reconocimiento-acción comienza con un filtrado que selecciona todas las reglas activadas, es decir, aquellas cuyo antecedente se satisface según las reglas de la memoria de trabajo. Después se resuelven conflictos en las reglas activadas con alguna estrategia de resolución de conflictos. Por último, se disparan las reglas del conjunto resultante de la resolución de conflictos, realizando su acción.

Encadenamiento hacia atrás Se entiende a la inversa: para obtener la meta, hay que obtener primero el antecedente de la regla cuyo consecuente es la meta.

Se parte del conjunto de metas, se intenta disparar reglas que concluyan en la meta convirtiendo sus antecedentes en nuevas submetas, y se para cuando se alcanza la meta o no hay reglas activadas.

Sirve si se dispone de pocos datos inicialmente, si se puede plantear razonablemente una meta o si la tarea se ajusta por su naturaleza.

Modelo básico encadenamiento hacia atrás

En el caso del encadenamiento hacia atrás, el ciclo básico reconocimiento-acción parte de un filtrado que selecciona las reglas cuyo consecuente permite obtener la meta actual, que no tienen por qué estar activadas. Después, mediante las estrategias de resolución de conflictos decididas, se selecciona un subconjunto de las reglas filtradas. Por último, se desencadenan las reglas reemplazando la meta actual por las submetas obtenidas de las condiciones de las reglas y se disparan las reglas cuyo antecedente se satisface, realizando su acción.

Estrategias de resolución de conflictos

Las estrategias de resolución de conflictos son criterios adicionales para decidir qué regla o reglas de las activadas se dispararán. Es habitual aplicar secuencialmente varios criterios hasta obtener una única regla.

Algunas estrategias de resolución de conflictos son:

- Refracción: Cada regla solo se puede disparar una vez con los mismos elementos de la memoria de trabajo. En el caso de encadenamiento hacia atrás, se desencadenan en lugar de dispararse.
- Recencia: Se selecciona la regla que se satisfaga con los hechos más recientemente añadidos a la memoria de trabajo.
- Especificidad: Se selecciona la regla que contenga más premisas.
- Prioridad: Se selecciona la regla con máxima prioridad, que se fija manualmente.
- Orden: Se selecciona la primera regla según el orden en la base.
- Todas: Se disparan todas las reglas activadas.

La estrategia de refracción es necesaria ara evitar ciclos.

La estrategia de prioridad se suele usar para agrupar reglas por tareas. No es recomendable utilizar demasiados niveles de prioridad.

La estrategia de orden es más bien un criterio para obtener un comportamiento determinista.

El disparo de reglas debe ser oportunista (en función del contenido de la memoria de trabajo) y no estar prefijado por el programador.

5. Encadenamiento hacia adelante

Encadenamiento hacia atrás

Lenguajes con variables

Las variables extienden el poder de representación de los lenguajes de reglas. Las reglas con variables permiten representar conocimiento general. La semántica de las variables es similar a la de la LPO (Lógica de Primer Orden).

Estos lenguajes extienden el formalismo Objeto-Atributo-Valor, y usan patrones simbólicos.

Un patrón simbólico es una secuencia de constantes y/o variables. Los hechos son patrones de constantes, y las reglas admiten patrones con variables. Aunque esto no está limitado a la estructura O-A-V, sigue siendo la conceptualización más recomendable.

Ligadura de variables

Las variables de los patrones se pueden reemplazar por constantes.

Una ligadura es una constante, d, que reemplaza a una variable.

Una variable ligada es una variable para la que existe una ligadura, $?x = d$.

Una sustitución es reemplazar una variable por su ligadura, en su alcance.

El alcance de una ligadura es: si la variable es muda, la ocurrencia de la variable; si no es muda, la regla en la que ocurre el patrón.

Equiparación de patrones

Se denomina equiparación, confrontación o *pattern matching*. Si p es un patrón y h un hecho, se dice que el patrón p y el hecho h se equiparan si y solo si existen ligaduras para las variables que ocurren en p tales que al sustituir las variables por sus ligaduras, p y h son sintácticamente iguales.

En clips, ? es una variable anónima que no se liga.

Los patrones, con o sin variables, pueden formar parte de las reglas de producción.

Semántica

Predicados

- `iguales<patrón>` es cierto si `<patrón>` confronta con algún hecho de la memoria de trabajo.
- `noiguales<patrón>` es cierto si `<patrón>` no confronta con ningún hecho de la memoria de trabajo.
- `mayorque<patrón>`/`menorque<patron>` es cierto si todas las variables están ligadas y al sustituir todas las variables se obtiene una secuencia de números estríctamente decreciente/creciente.

Acciones

- `añadir<patrón>` sustitye todas las variables y añade el hecho resultante si todas las variables están ligadas.
- `eliminar<patrón>` sustituye todas las variables y elimina el hecho resultante de la memoria de trabajo.

Particularización de regla

Una particularización de una regla es un par formado por la regla y el conjunto minimal de hechos de la memoria de trabajo que satisface su antecedente.

La regla particularizada es la que se obtiene a partir de la regla al sustituir las variables por las ligaduras que hacen que los patrones de la regla confronten con los hechos del conjunto minimal.

Generalmente hay múltiples conjuntos que satisfacen el antecedente y por tanto, varias particularizaciones.

Con esta definición, podemos decir que la estrategia de resolución de conflictos "refracción" indica que no se dispare dos veces la misma particularización de una regla.

Algoritmo de Rete

Equiparación de reglas y hechos.

Aproximación naive: en cada ciclo, comparar antecedente de las reglas con hechos en memoria de trabajo. Esta estrategia es ineficiente debido a que el conjunto de reglas es estable, la memoria de trabajo cambia pero normalmente el porcentaje de cambio por ciclo es pequeño. Además, la mayoría de patrones que se satisfacen en un ciclo también lo hacen en el siguiente.

La alternativa es un algoritmo que recuerde activaciones entre ciclos, actualizando patrones que confronten con los hechos que han cambiado.

El algoritmo de Rete representa las reglas como datos en la denominada red de Rete. El compilador crea esta red a partir de las reglas. La red de Rete se puede asimilar a una máquina de estados que consume modificaciones de hechos. La red recuerda estados anteriores.

Representación del conocimiento: métodos estructurados

Introducción (PDF)

Los métodos estructurados son una familia de métodos que utilizan grafos para la representación del conocimiento. Se basan en las relaciones entre los elementos de un dominio, y hacen explícita la estructura del dominio.

Son, por tanto, particularmente adecuados para representar conocimiento estructural.

Principalmente, estos métodos son o bien redes semánticas o marcos.

Evolución

Los sistemas de métodos estructurados iniciales tenían una semántica poco precisa. Se va elaborando la herencia como mecanismo de inferencia, y se van desarrollando sistemas con semántica bien definida.

En la actualidad existen formalismos para representar el conocimiento estructural.

Las tendencias actuales en redes semánticas es la web semántica, y en marcos son las ontologías (con clases, subclases, propiedades y restricciones de las mismas).

Redes semánticas

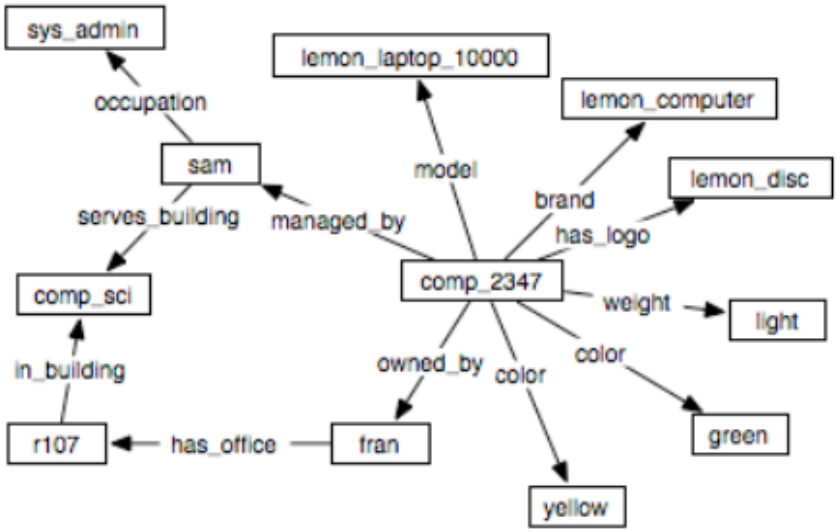
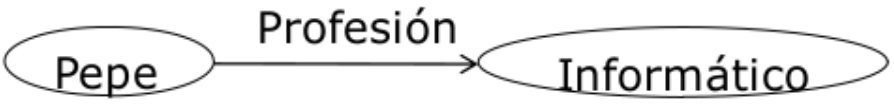


Figure 5: Ejemplo de red semántica

Una red semántica es un formalismo gráfico basado en relaciones binarias. Del lenguaje natural ("Pepe es informático"), pasamos a la lógica de primer orden ("PROFESION(Pepe, Informático)") y por último a la red semántica:



.

La red semántica nos permite una representación gráfica del formalismo Objeto-Atributo-Valor. Los nodos representan los conceptos o entidades (objetos, valores) y los arcos expresan la relación binaria (atributos).

Sintaxis

Una red semántica es un grafo formado por nodos etiquetados que representan entidades, conceptos, valores; y por arcos unidireccionales etiquetados que

representan las relaciones binarias.

Se puede utilizar cualquier etiqueta para un nodo u arco, existe una falta de estandarización.

Hay dos tipos de arcos. Los **descriptivos** proporcionan propiedades de las entidades, y los **estructurales** proporcionan la estructura de la red.

Los estructurales sí tienen cierto grado de estandarización, y su significado es independiente del dominio concreto. Algunos ejemplos son `subclase-de` para generalización, `instancia-de` para instanciación o `parte-de` para agregación.

Para representar predicados no binarios, es necesario **reificar** el predicado creando una instancia del tipo de predicado correspondiente que esté relacionada con cada uno de sus parámetros. La siguiente imagen muestra la reificación del predicado `COMPRAVENTA(Pepe, Luis, Reloj1, 45, Euros)`.

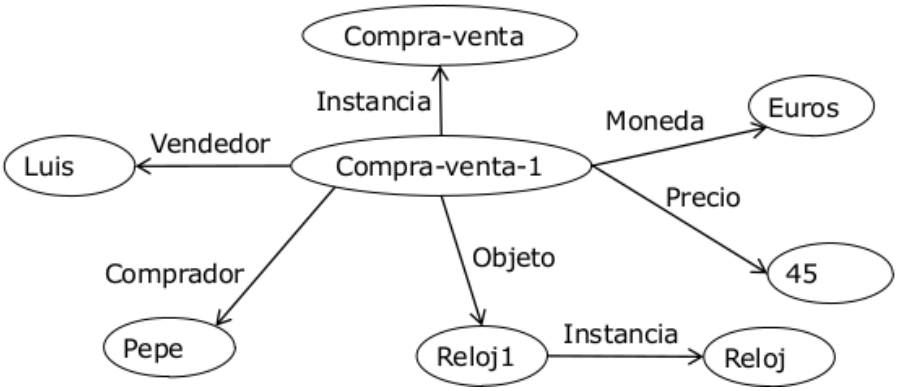


Figure 6: Ejemplo de reificación

Inferencia en redes semánticas

Las redes semánticas proporcionan mecanismos de inferencia asociados a los arcos de la red y a procedimientos que los manipulan. Existen dos tipos de inferencias:

- Equiparación: permite resolver preguntas que se representan como una red semántica. Se crea una subred pregunta, con nodos constantes, nodos variables y arcos etiquetados. Se superpone la subred a la original, y si se consigue una superposición perfecta de los nodos constantes y los arcos, se asigna a los nodos variables los valores encontrados en la red.
- Herencia de propiedades: permite que nodos de la red obtengan las propiedades definidas en otros nodos mediante los arcos `instancia` y `subclase-de`. Para ello se pregunta por un nodo y se busca el arco por

el que se pregunte. Si no se encuentra, el motor de inferencia recorre los arcos `instancia` y `subclase-de`. En cada camino, prevalece el nodo más próximo que tenga la propiedad. La herencia evita repetir propiedades en instancias y subclases, gestiona bien las excepciones herendando el valor de la propiedad más cercano, pero no gestiona las contradicciones que pueden encontrarse por caminos distintos.

Marcos (PDF)

Introducción

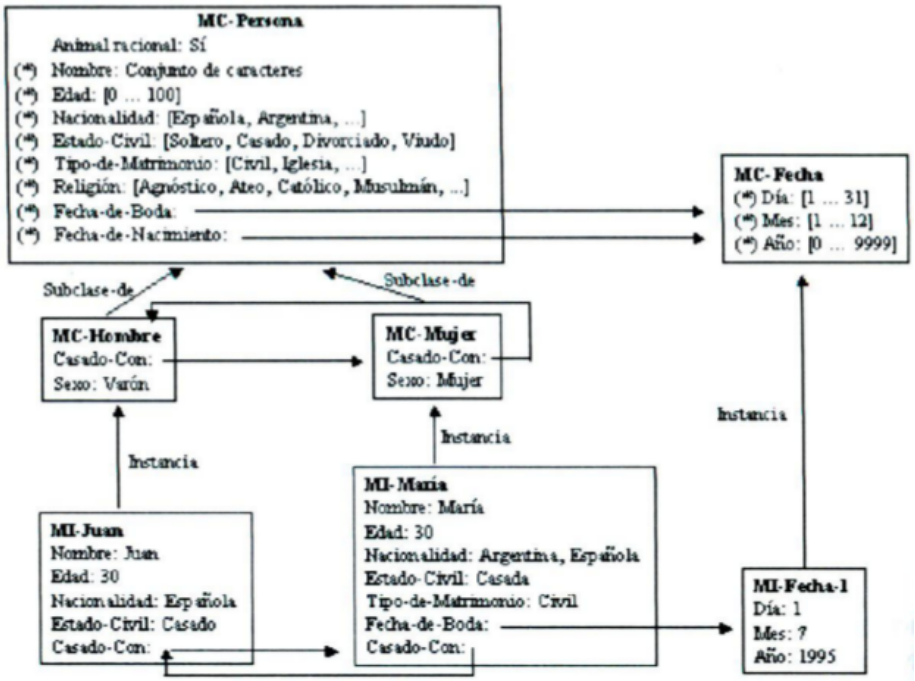


Figure 7: Ejemplo de marcos

Los marcos son estructuras de datos utilizadas para representar elementos bien conocidos (prototípicos). Para adaptarnos a la situación actual, accedemos a la estructura que más se asemeja y modificamos los detalles necesarios.

Como las redes semánticas, hacen explícitas las relaciones de dominio que proporcionan la estructura de la red, pero a diferencia de estas, los nodos tienen estructura y permiten agrupar las propiedades de los elementos del dominio en una unidad denominada **marco** (*frame*).

La representación es principalmente declarativa, especificando las propiedades

de los conceptos o individuos, pero permiten añadir elementos operacionales a propiedades individuales mediante facetas.

El dominio queda estructurado en jeraquías de herencia.

Para la inferencia se usa la equiparación, la herencia y los métodos operacionales.

En la actualidad, los marcos son el principal formalismo de representación del conocimiento cuando éste se organiza en clases o cuando es conocimiento estructural. La organización en jerarquías de clases y la herencia se mantienen en la mayoría de lenguajes de ontologías.

Elementos de un sistema de marcos

- Marcos: representan clases (hombre, puerta lógica, alarma…) e instancias (Juan, AND-1, alarma incendio 327…).
- Relaciones: representan dependencias entre marcos. Las estándar son:
  - `subclase-de`: Relación binaria entre marcos de clase. Es dirigida, de orden parcial (reflexiva, antisimétrica y transitiva). La inversa de la relación es la superclase. Con esta relación se crea una jerarquía de clases desde las más generales a las más específicas.
  - `instancia-de`: Relación dirigida entre marcos de instancia y de clase. La relación inversa es la representación. También establece jerarquías.
  - Otras relaciones no estándar: Pueden existir relaciones binarias y dirigidas dependientes del dominio (`hermano-de`, `casado-con`…). Se han de definir primero entre marcos de clase.

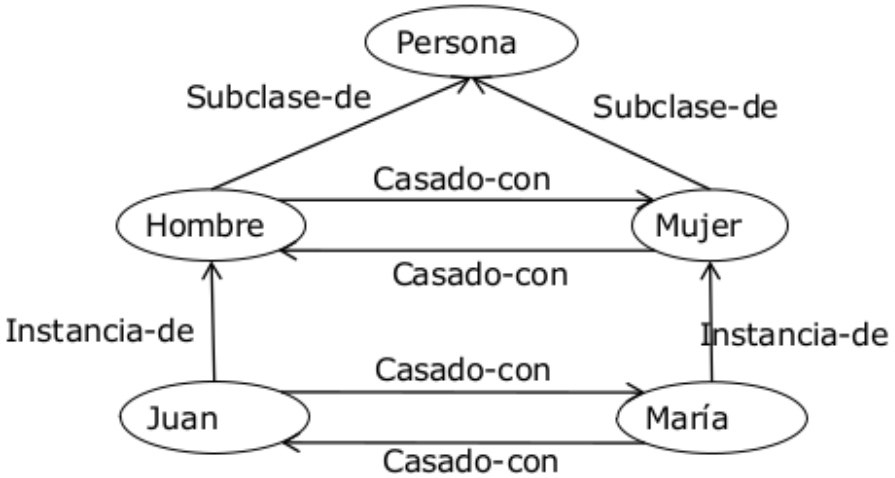


Figure 8: Ejemplo de relaciones no estándar

- Propiedades: describen los marcos de clase o de instancia. Las propiedades de clase reciben el valor en la clase que las define y son comunes a todas las instancias salvo excepciones. Las propiedades de instancia describen las propiedades específicas de una instancia. Se definen en los marcos de clase pero toman valor en los de instancia, y normalmente el valor es distinto en cada instancia.
- Facetas: descripción adicional de los valores que pueden tomar las propiedades. Algunas características adicionales que pueden dar las facetas es información sobre el tipo de una propiedad, cardinalidad de las relaciones, o demonios (procedimientos que se invocan automáticamente al acceder o alterar el valor de una propiedad).