Representación avanzada de conocimiento y razonamiento

Departamento de Ingeniería del Software
e Inteligencia Artificial

Índice

- Introducción
- Sistemas basados en lógica
  - Lógica de predicados
  - Prolog
- Representación estructurada del conocimiento
  - Redes semánticas
  - Web Semántica
  - Sistemas de marcos
  - Ontologías

Introducción

Representación de conocimiento

- La **representación de conocimiento** es el área de la inteligencia artificial que estudia como representar el mundo en un ordenador de manera que pueda ser usado para resolver problemas complejos
  - Sistemas de diagnóstico médico
  - Sistemas de diálogo en lenguaje natural
  - Sistemas de enseñanza con tutores virtuales

- Son **formalismos de representación** que facilitan el diseño, construcción y mantenimiento de **bases de conocimiento**.
  - Incorpora ideas de la **psicología** sobre como las personas conceptualizan la información y la usan para resolver problemas
  - Incorpora ideas de la **lógica** matemática para definir mecanismos de razonamiento correctos y completos
  - Incorpora ideas de la **ingeniería** del software para construir grandes bases de conocimiento de forma colaborativa y modular

Representación de conocimiento

- La representación de conocimiento va unida a la capacidad de **razonamiento automático** sobre dicho conocimiento
  - Realizar inferencias, comprobar consistencia, responder preguntas…
  - Todos los lenguajes de representación de conocimiento van unidos a mecanismos de inferencia

- Muchos aspectos a considerar
  - Ingeniero del conocimiento vs experto del dominio
  - Conocimiento factual vs procedimental vs meta-conocimiento
  - Representaciones declarativas vs procedimentales
  - Conocimiento estático vs dinámico
  - Expresividad vs garantías computacionales
  - Mundo abierto vs cerrado
  - …

¿Necesitamos formalismos para representar conocimiento?

- Podemos representan información usando lenguajes de programación estándar pero…
  - La programación procedimental es buena para representar conocimiento **procedural** (cómo hacer las cosas)
  - No es buena para representar conocimiento **factual** (conceptos abstractos y sus relaciones, hechos)
  - Los expertos de cada dominio ya tienen formas de representar conocimiento (reglas de negocio, taxonomías, etc.) pero no entienden de código
    - **Los expertos deberían poder crear y mantener las bases de conocimiento**
  - En general es una buena idea separar el conocimiento de un dominio de los mecanismos que lo usan para resolver problemas

¿Necesitamos formalismos para representar conocimiento?

- Podemos utilizar técnicas de aprendizaje automático para aprender conceptos abstractos a partir de ejemplos pero…
  - Las mayoría de las representaciones que obtenemos no son fácilmente interpretables (redes neuronales, svm, …)
  - Cuanto más abstracto es un concepto, más complejo es aprenderlo a partir de ejemplos concretos
    - Metaconocimiento, introspección, sensaciones y sentimientos, lenguaje…
  - Los seres humanos llevamos miles de años acumulando y transmitiendo conocimiento, no tiene sentido aprenderlo todo desde cero otra vez
    - Los ejemplos son sólo un recurso docente pero también enseñamos con reglas generales y conceptos abstractos
  - Si queremos sistemas que “piensen” como nosotros necesitamos que “razonen” como lo hacemos nosotros

Conocimiento declarativo vs procedimental

- **Representación declarativa**
  - Representamos el conocimiento de forma independiente a cómo va a ser usado
  - Especialmente interesante para representar conocimiento factual (hechos)
  - Se acompaña de un programa que especifica qué hacer con el conocimiento y cómo hacerlo
- **Representación procedimental**
  - La representación del conocimiento se hace pensando en cómo se va a utilizar
  - Especialmente interesante para representar conocimiento heurístico (estrategias)
    - Optimización mediante información de control
  - El conocimiento no es directamente utilizable en otro sistema diferente
- Siempre ha habido mucha controversia en IA sobre qué tipo de representación es mejor en la práctica
  - La representación declarativa es más pura pero dificulta codificar procesos de razonamiento óptimos en situaciones concretas

Conocimiento declarativo vs procedimental

- Ejemplo: tipo de un triángulo según las longitudes de sus lados

- **Representación declarativa** (independiente de cómo se usa)

  Equilátero ⇔ a == b **and** b == c

  Isósceles ⇔ (a == b **and** b != c) **or** (a == c **and** c != b) **or** (b == c **and** c != a)

  Escaleno ⇔ a != b **and** b != c **and** a != c

- **Representación procedimental** (sabemos cómo se va a usar)

  a == b **and** b == c ⟹ Equilátero
  a == b **or** b == c **or** a == c ⟹ Isósceles
  Escaleno

Tipos de conocimiento

- Tipos de conocimiento según la taxonomía de Bloom

qué

- **Conocimiento factual** (*)
  - Terminología, detalles específicos
- **Conocimiento conceptual** (*)
  - Categorías, generalizaciones, modelos, teorías, …

cómo

- **Conocimiento procedimental**
  - Habilidades, técnicas, algoritmos y cuando usarlos
- **Metaconocimiento**
  - Estrategias, introspección, conocimiento sobre los propios mecanismos de razonamiento

(*) nota: nosotros usaremos el término factual para referirnos a las dos primeras categorías

A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. Anderson LW, Krathwohl DR, Airasian PW, et al (2001)

Principales tipos de conocimiento factual

hechos

- **conceptos**
  - individuales *(ejemplares, individuos o instancias)*
  - abstractos *(clases o conjuntos)*
    - comprensión
    - extensión
- Conocimiento terminológico o estático *(vocabulario del sistema)*
  - clasificación
    - ejemplar
    - subclase
  - **relaciones entre conceptos**
    - propiedades
      - esenciales
      - estereotípicas
      - individuales
- Conocimiento dinámico o asertivo
  - asertos

Expresividad vs Garantías computacionales

- En todo los formalismos de representación hay que buscar un equilibro entre
  - **Expresividad**: lo que podemos expresar o representar
  - **Computabilidad y eficiencia**: que existan algoritmos eficientes para usar la información representada

- A mayor expresividad menores garantías computacionales de los procesos de inferencia
  - La lógica de primer orden es semidecidible (no existe un algoritmo que determine siempre si algo no es consecuencia lógica de otra cosa).

- En la práctica tenemos que elegir entre grandes bases de conocimiento con información semántica limitada o bases de conocimiento pequeñas con más información semántica.
  - Bases de datos, redes semánticas, sistemas de producción, demostradores de teoremas utilizan distintos puntos de equilibrio entre expresividad y eficiencia.

Formalismos de representación basados en lógica

- **Basados en lógica**
  - Lógica de primer orden
  - Cláusulas de Horn (Prolog)
  - Lógicas descriptivas
  - Lógica proposicional
  - …

- Como la lógica de primer orden no es decidible se usan subconjuntos con más garantías computacionales
  - Sacrificando expresividad
  - Ganando eficiencia en ciertos procesos de inferencia

- ¿Cuál? El más adecuado en cada caso…
  - ¿Cuánto conocimiento necesitas representar?
  - ¿Qué tipos de razonamiento quieres hacer?
  - ¿Cuánto pueden tardar?

Formalismos de representación basados en lógica

- Ventajas
  - Semántica formal bien definida
  - Mecanismos de inferencia bien definidos y estudiados
    - Contestar preguntas
    - Comprobar la consistencia de la base de conocimiento
    - Resolver problemas complejos
  - Representación declarativa del conocimiento
  - Representación concisa

- Desventajas
  - Requiere conocimientos matemáticos avanzados
    - No son adecuados para expertos de otros dominios
  - Bases de conocimiento sin estructura clara
    - Dificultad para crear y mantener grandes bases de conocimiento
  - Es difícil representar conocimiento procedural y heurístico

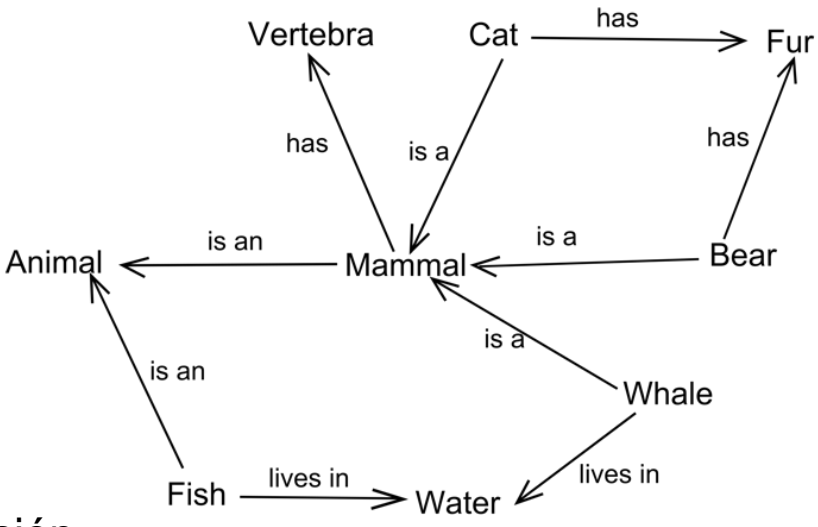
Formalismos basados en representaciones estructuradas

- **Basados en representaciones estructuradas**
  - Redes semánticas
  - Sistemas de marcos (frames)
  - Ontologías

- Orientados a la representación modular de grandes bases de conocimiento
- Más intuitivos para no matemáticos (para los expertos)
- Pueden tener por debajo un motor de inferencia basado en algún tipo de lógica

Redes semánticas

- Conocimiento representado mediante grafos
  - Conceptos como nodos
  - Relaciones binarias como aristas

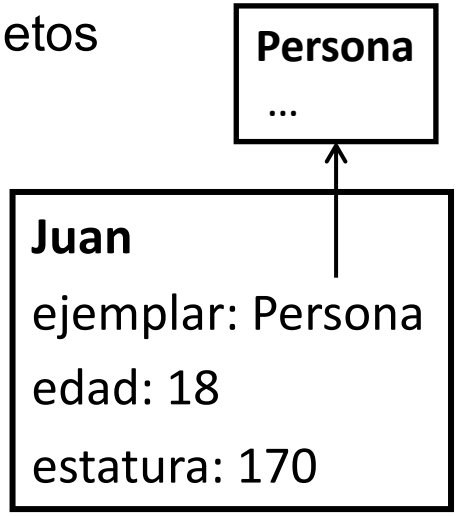


- Ventajas
  - Relaciones estándar (primitivas) bien definidas
    - instancia, subtipo, agregación
  - Representación declarativa del conocimiento
  - Representación gráfica que facilita su comprensión
  - Recuperación de información mediante encaje de grafos
  - Auto-organización de la base de conocimiento en base a relaciones primitivas

- Desventajas
  - No hay una separación explícita de instancias y clases
  - Relaciones no estándar para cada dominio que carecen de una semántica formal bien definida
    - Dificultad para razonar con ellas
  - No hay una representación modular de las entidades
    - Red de relaciones donde es difícil delimitar entidades

Sistemas de marcos

- Representación estructurada de instancias y clases usando pares atributo-valor
  - Conceptos similares a los de programación orientada a objetos



- Ventajas
  - Separación explícita entre instancias y clases
  - Posibilidad de meta-razonamiento (metaclases)
  - Semántica formal de algunas relaciones estándar
  - Combina conocimiento procedimental y declarativo
  - Proporcionan mecanismos para trabajar con excepciones

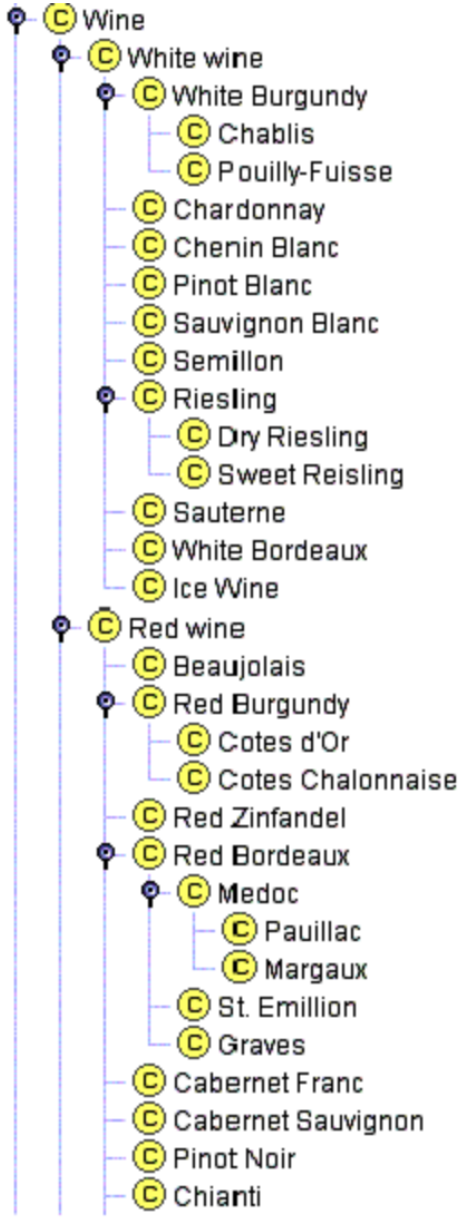
- Desventajas
  - Las relaciones de cada dominio particular siguen sin estar bien definidas
  - Capacidad de razonamiento limitado
  - Problemas para representar ciertos tipos de conocimiento

Ontologías

- Definición formal del vocabulario de un dominio
  - Clases, relaciones, propiedades, individuos
  - Definición de taxonomías en base a la relación de subtipo

- Ventajas
  - Semántica formal bien definida
  - Mecanismos de inferencia bien definidos y estudiados
  - Herramientas visuales para crear y gestionar bases de conocimiento
  - Representación declarativa del conocimiento
  - Separación explícita entre instancias y clases
  - Potencia la reutilización de las bases de conocimiento

- Desventajas
  - Sigue siendo difícil crear grandes bases de conocimiento
  - Difícil equilibrio entre expresividad y eficiencia
  - Dificultad para expresar ciertos tipos de conocimiento



Mundo abierto vs cerrado

- Las bases de conocimiento nunca contienen “todo” el conocimiento. ¿Qué ocurre con el conocimiento no representado?

- **Mundo cerrado**
  - El conocimiento que no está se asume que es falso
  - Facilita la creación de las bases de conocimiento, es más intuitivo
  - Puede llegar nueva información que invalide inferencias hechas (!!)

- **Mundo abierto**
  - No se asume nada del conocimiento que no está
  - Es menos intuitivo y nos obliga a añadir mucho conocimiento “de sentido común”
  - Menos inferencias posibles pero ninguna se invalida si llega nuevo conocimiento consistente con lo que ya sabíamos

Mundo abierto vs cerrado

- Base de conocimiento
  - Juan tiene alergia a las nueces y a las almendras
  - Juan y Sara son hermanos
  - Las personas sin alergias comen de todo

Pregunta	Mundo cerrado	Mundo abierto
¿Sara tiene alergia a las nueces?	No	?
¿Cuántos hermanos tiene Sara?	1	>=1
¿Sara come de todo?	Sí	?
¿Juan come de todo?	No (no se dice que coma de todo en la BC)	? (no sabemos si las personas con alergias comen de todo)

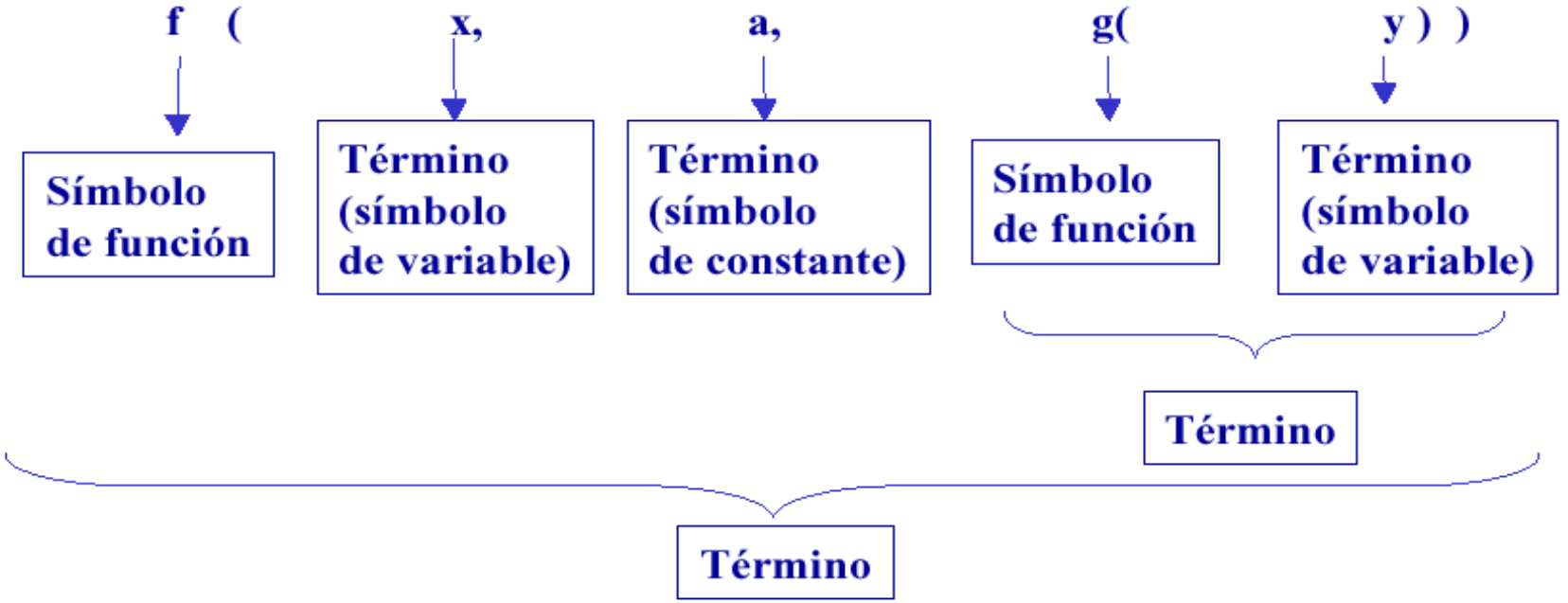
- En sistemas de mundo abierto a veces ni siquiera se asume un nombre único para las entidades. Por ejemplo, si añado a la base de conocimiento:
  - Juan sólo tiene alergia a una cosa

Mundo cerrado	Mundo abierto
Base de conocimiento inconsistente	Nueces y almendras son lo mismo

Lógica de predicados

Sintaxis

- Términos
  - Un símbolo de constante es un término *(a, b, c...)*
  - Un símbolo de variable es un término *(x, y, z...)*
  - Si *f* es un símbolo de función (o functor) de aridad *n*, y *t1, t2, ..., tn* son términos, entonces *f(t1, t2, ..., tn)* es un término compuesto
- Ejemplo: *f (x, a, g(y))*



Sintaxis

- Fórmulas

ATÓMICAS

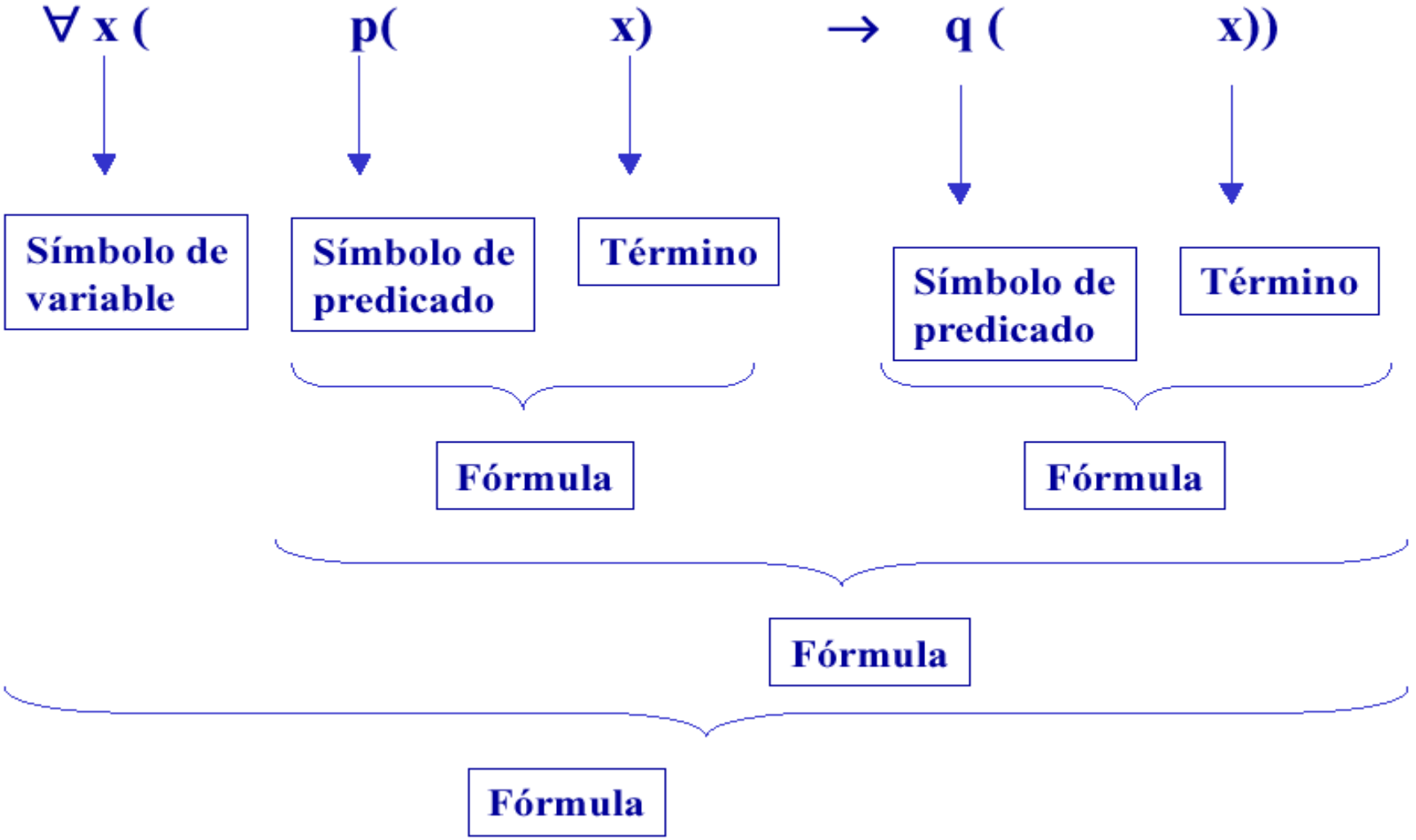
  - Los símbolos de verdad T y el de falsedad o contradicción ⊥ son fórmulas
  - Si *p* es un símbolo de predicado de aridad *n*, y *t1, t2, ..., tn* son términos, entonces *p(t1, t2, ..., tn)* es una fórmula

COMPUESTAS

  - Si *F* es una fórmula, entonces $\neg F$ es una fórmula
  - Si *F* y *G* son fórmulas, entonces:
    - $(F \wedge G)$ es una fórmula
    - $(F \vee G)$ es una fórmula
    - $(F \rightarrow G)$ es una fórmula
    - $(F \leftrightarrow G)$ es una fórmula
  - Si *x* es un símbolo de variable, y *F* es una fórmula, entonces:
    - $\forall x\ F$ es una fórmula
    - $\exists x\ F$ es una fórmula

Sintaxis

- Ejemplo de fórmula: $\forall x\ (p(x) \rightarrow q(x))$



Semántica

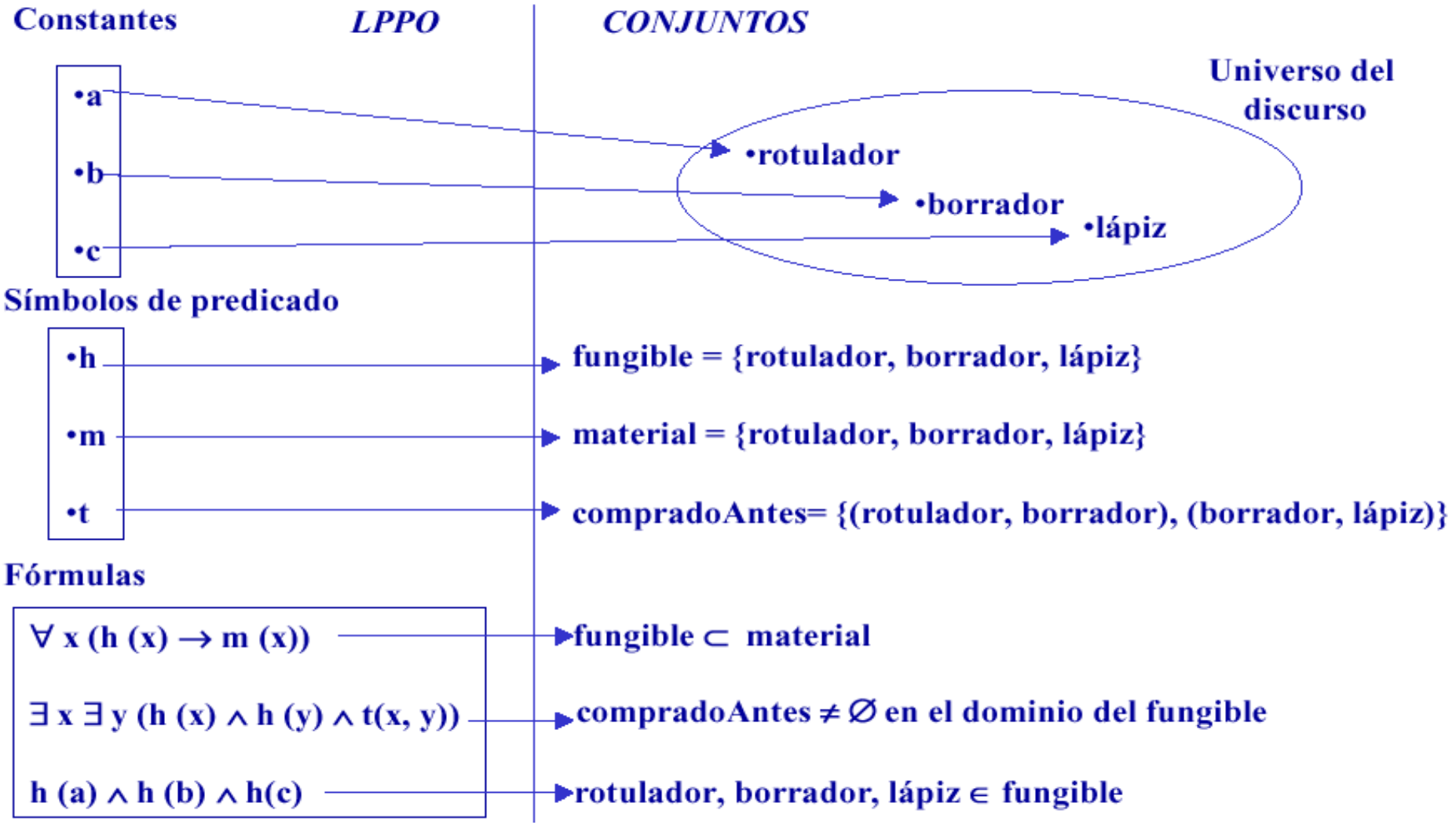
- Los significados de términos y fórmulas se obtienen fijando una interpretación (U, I) que consta de
  - Un conjunto no vacío llamado *U* universo (o dominio) de discurso
  - Una función de interpretación I que asocia:
    - Elementos de *U* a los símbolos de constante
    - Funciones en U a los símbolos de función
    - Relaciones (tuplas de elementos) en U a los símbolos de predicado

- Las interpretaciones fijan el significado de las fórmulas atómicas
  - *p(t1, t2, ..., tn) es cierto si (t1, t2, ..., tn)* $\in I$

Semántica

- La semántica de las fórmulas compuestas es la habitual
  - $\neg F$ es cierto si F es falso
  - $(F \wedge G)$ es cierto si F y G son ciertos
  - $(F \vee G)$ es cierto si F o G son ciertos
  - $(F \rightarrow G)$ es cierto si F es falso o F y G son ciertos
  - $(F \leftrightarrow G)$ es cierto si F y G son ambos ciertos o ambos falsos
  - $\forall x\ F$ es cierto si F(x) es cierto para todo x
  - $\exists x\ F$ es cierto si F(x) es cierto para algún x

Semántica


Constantes
LPPO
CONJUNTOS
Universo del discurso
•a
•b
•c
•rotulador
•borrador
•lápiz
Símbolos de predicado
•h
•m
•t
fungible = {rotulador, borrador, lápiz}
material = {rotulador, borrador, lápiz}
compradoAntes= {(rotulador, borrador), (borrador, lápiz)}
Fórmulas
∀ x (h (x) → m (x))
∃ x ∃ y (h (x) ∧ h (y) ∧ t(x, y))
h (a) ∧ h (b) ∧ h(c)
fungible ⊂ material
compradoAntes ≠ ∅ en el dominio del fungible
rotulador, borrador, lápiz ∈ fungible

Propiedades

- Representación declarativa
  - No está fijada la forma en la que debe ser usado el conocimiento
    - Sencillez para incorporar nuevo conocimiento, o eliminarlo
    - Sencillez de integración de dos o más BCs
- Corrección y completitud
  - Cuenta con mecanismos deductivos correctos y completos
    - Permiten deducir nuevo conocimiento (conclusiones) a partir del conocimiento de partida (premisas)
    - Pueden usarse para responder a preguntas o resolver problemas
      - Demostración automática de teoremas: una de las áreas iniciales de la IA
- La deducción es semi-decidible en LPO
  - Si lo que pretendemos demostrar no se deduce del contenido de la base de conocimiento, no está garantizado que termine el proceso de demostración
    - No existe un procedimiento de decisión, ni siquiera de coste exponencial

Dificultades en la representación

- Lenguaje natural → lógica de predicados
  - Ambigüedad, sentido común, implicaciones y disyunciones...
- Incompletitud del conocimiento representado
  - Es imposible representar TODO el conocimiento
  - Representamos sólo lo más importante
  - No incluimos conocimiento de “sentido común”
    → habrá razonamientos que no podrán hacerse
- Infinidad de alternativas de representación dentro de la lógica
  - Influyen en el proceso de razonamiento. Algunas representaciones facilitan un tipo de razonamiento y dificultan otro
  - Lo importante es usar una representación consistente dentro de la misma BC
- Dificultades en la representación del conocimiento por defecto
  - Por ejemplo, las excepciones a la herencia

Alternativas de representación

- Ejemplo
  - Representación de conocimiento factual terminológico
    - Relaciones de ejemplar ($\in$) y subclase ($\subseteq$)
    - Propiedades esenciales
- Conocimiento que queremos representar
  - "Flipper es un delfín" *ejemplar de una clase*
  - "Todos los delfines son vertebrados" *relación de subclase*
  - "Los vertebrados tienen esqueleto" *propiedad esencial*
- Criterios para la elegir una alternativa de representación
  - Qué razonamientos se pueden hacer eficientemente
    - Representar lo que tenemos
    - Obtener lo que buscamos (deducciones, inferencias)
  - La representación debe ser consistente en todo el dominio

Alternativas de representación

- Versión 1
  - Representación implícita de ejemplares: el nombre de la clase es un símbolo de predicado unario, el argumento es el ejemplar

    Delfín(flipper)
  - Representación de la subclase *delfín* de la clase *vertebrados*

    ∀x (Delfín(x) → Vertebrado(x))
  - Representación de propiedades esenciales

    ∀x (Vertebrado(x) → Tiene-Esqueleto(x))
  - Ventajas
    - Sencillez de la representación
  - Desventajas
    - Para cualquier clase (Delfín, Elefante...)
      - crear predicados y reglas (subclases) específicas
    - Se complica el razonamiento general (sobre conceptos o clases)
    - El conocimiento implícito se deduce a través de las implicaciones.
      - Es mejor afirmar las cosas explícitamente por medio de hechos: VERSION 2

Alternativas de representación

- Versión 2
  - Representación explícita de la pertenencia de ejemplares a clases: se utiliza un símbolo de predicado binario Es_Un(ejemplar, Clase)

    Es_Un(flipper, Delfín)
  - Representación de la subclase *delfín* de la clase *vertebrados*

    ∀x (Es_Un(x, Delfín) → Es_Un(x, Vertebrado))
  - Representación de propiedades esenciales

    ∀x (Es_Un(x, Vertebrado) → Tiene_Esqueleto(x))

  - Ventajas
    - Representación explícita de pertenencia de ejemplares a clases, es_un
  - Desventajas
    - La relación de subclase sigue siendo implícita

Alternativas de representación

- Versión 3
  - Representación explícita de ejemplares y subclases utilizando predicados Es_Un

    Es_Un(flipper, Delfín)

    Es_Un(Delfín, Vertebrado)

    ∀x (Es_Un(x, Vertebrado) → Tiene_Esqueleto(x))

  - Ventajas
    - Representación explícita de pertenencia de ejemplares a clases
    - Representación explícita de la relación subclase
  - Desventajas
    - El sistema no puede diferenciar si *flipper* es individuo o clase
    - No puede deducirse que *flipper* tenga esqueleto
    - Falta especificar la transitividad de la relación *Es_Un*. Hay que añadirla

      ∀x∀y∀z (Es_Un(x, y) ∧ Es_Un(y, z) → Es_Un (x, z))

Alternativas de representación

- Versión 4
  - Representación explícita de ejemplares y subclases utilizando dos símbolos de predicado binarios distintos

    Ejemplar(flipper, Delfín)

    Subclase(Delfín, Vertebrado)

    $\forall x$ (Ejemplar(x, Vertebrado) → Tiene_Esqueleto(x))

  - Añadimos la transitividad (entre subclases y ejemplares)

    $\forall x \forall y \forall z$ (Ejemplar(x,y) $\wedge$ Subclase(y,z) → Ejemplar(x,z))

    $\forall x \forall y \forall z$ (Subclase(x,y) $\wedge$ Subclase(y,z) → Subclase(x,z))

Alternativas de representación

- Las cuatro versiones usan como técnica de representación la lógica de predicados (también denominada lógica de primer orden, LPO, o lógica de predicados de primer orden, LPPO)
  - Existen distintas posibilidades de representación dentro de la lógica (en general, esto ocurre con cualquier formalismo)
  - Algunas representaciones facilitan un tipo de razonamiento y dificultan otro

Conocimiento por omisión (by default)

- ¿Cómo añadir las excepciones a la herencia de propiedades?

  $\forall x$ (Gorila(x) → Pelo_Oscuro(x))

  Gorila(Copito)

  ¬ Pelo_Oscuro(Copito)

  – ¡Inconsistencia! *(inaceptable en una BC)*

  – Se necesita incluir las excepciones dentro de la definición general

  $\forall x$ (Gorila(x) $\wedge$ ¬ igual(x, Copito) → Pelo_Oscuro(x))

  Gorila(Copito)

- Excepciones a la herencia son difíciles de representar en LPO

  – En Prolog se simplifica porque no hay que modificar las reglas generales sino que basta con colocar las excepciones delante

Mecanismos de inferencia

- Deducción: obtención de nuevo conocimiento *(implícito)*
- Se trata de saber si una fórmula Q es cierta conociendo:
  - Los axiomas que son lógicamente válidos sea cual sea el significado de los símbolos *(tautologías)*

    $\neg F \vee F$
  - Los axiomas que son válidos bajo ciertas interpretaciones, es decir, sólo suponiendo ciertos significados de los símbolos *(conocimiento explícito)*
  - Las reglas de inferencia
    - Por ejemplo:

modus ponens

$P \rightarrow Q$
P

Q

modus tolens

$P \rightarrow Q$
$\neg Q$

$\neg P$

Ejemplo de deducción formal

- Dado un conjunto de hipótesis o premisas

  perro(milú)

  ∀x (perro(x) → animal(x))

  ∀y (animal(y) → mortal(y))

- Demostrar una conclusión

  mortal(milú)

- Pasos aplicados

  1. Aplicar instanciación universal con *x* = *milú*

     perro(milú) → animal(milú)

  2. Aplicar modus ponens

     animal(milú)

  3. Aplicar instanciación universal con *y* = *milú*

     animal(milú) → mortal(milú)

  4. Aplicar modus ponens

     mortal(milú)

Cláusulas y resolución

- En la práctica, resulta incómodo operar con un sistema deductivo formal en el que las expresiones lógicas utilizadas tienen formas muy variadas y se debe elegir entre muchas reglas de deducción aplicables en cada paso
- Esta situación se puede mejorar notablemente transformando las fórmulas lógicas a una forma normal más sencilla para operar
  - En particular, la forma de cláusula sólo utiliza las conectivas ¬ y ∨ (negación y disyunción), y necesita una sola regla de deducción

Resolución [Robinson, 1965]

$$\begin{array}{rcl} \boldsymbol{P \vee Q} & \sim & \neg P \rightarrow Q \\ \boldsymbol{\neg Q \vee R} & \sim & Q \rightarrow R \\ \hline \boldsymbol{P \vee R} & & \neg P \rightarrow R \end{array}$$

  - Con esta restricción no se pierde generalidad porque existe un algoritmo de conversión a forma normal conjuntiva que permite plantear cualquier problema lógico en forma de cláusulas

Deducción por resolución

Ejemplo: ¿a?

	Forma clausal
$b \wedge c \rightarrow a$	$a \vee \neg b \vee \neg c$
$d \rightarrow c$	$c \vee \neg d$
b	b
d	d

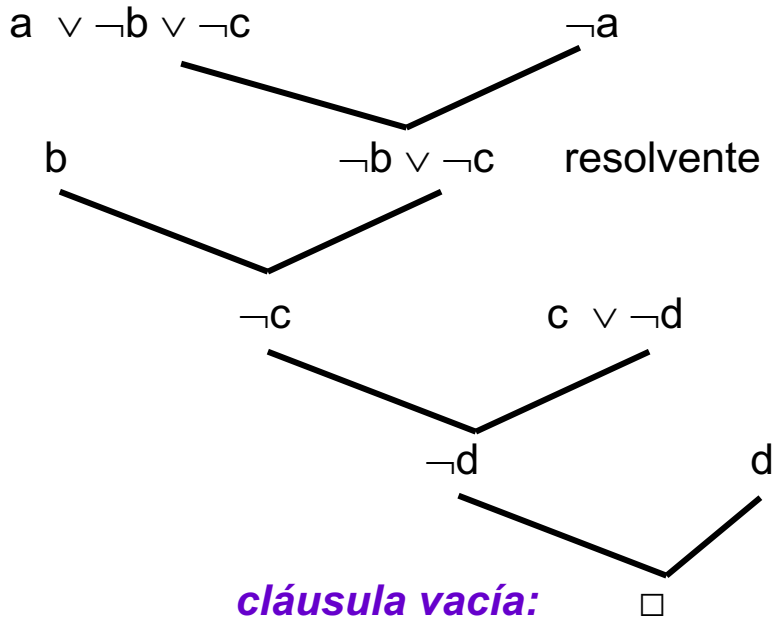
Añadir la negación de lo que queremos demostrar: **¬a**
y probar que llegamos a una contradicción *(conjunto insatisfactible)*

Resolución

$\varphi \vee Q$ ***un literal positivo en una cláusula***

$\neg Q \vee \psi$ ***y el mismo negativo en otra***

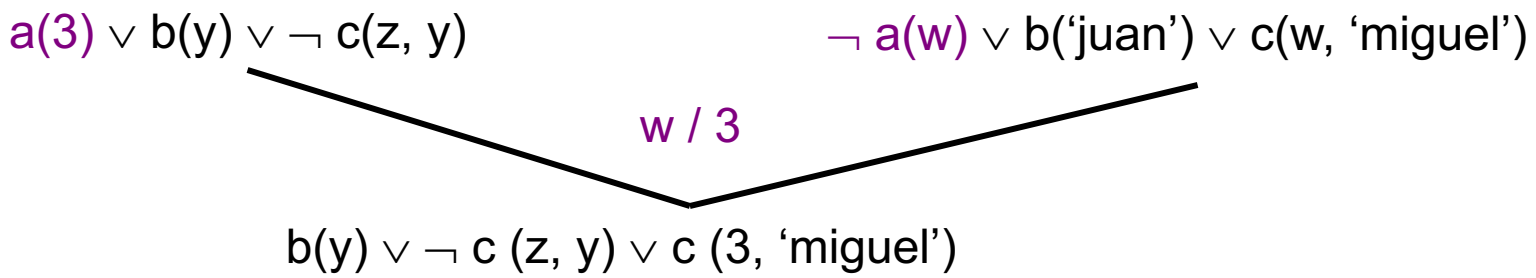
$\varphi \vee \psi$ ***resolvente***



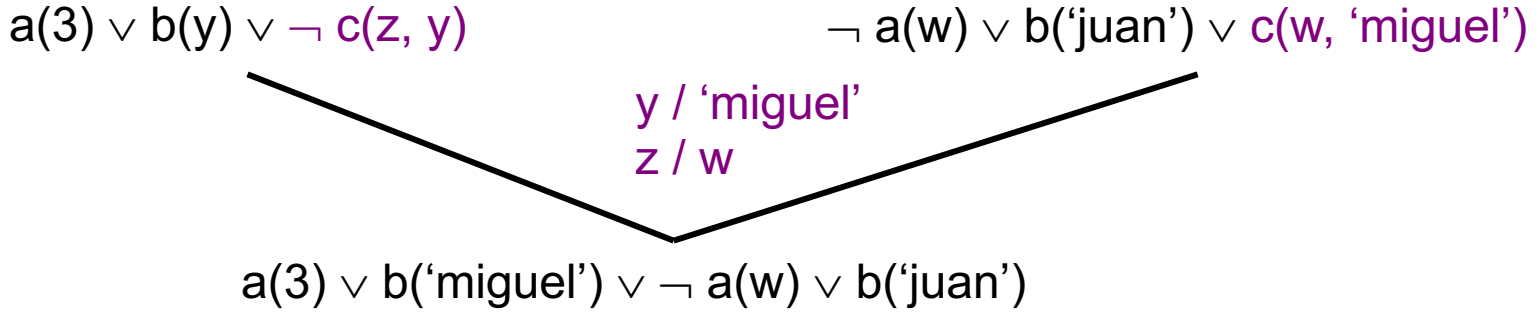
cláusula vacía: contradicción

Proposiciones → Predicados
(Resolución con unificación)

Resolución con unificación



U.m.g.: unificador más general posible de los literales candidatos
Se aplica a las cláusulas padres antes de calcular el resolvente
Deben considerarse variantes de las cláusulas (variables frescas)



Se selecciona un único par de literales complementarios.
Se pueden usar heurísticas para que la búsqueda no sea ciega.

Estrategias de resolución

- No es fácil determinar en cada paso qué cláusulas resolver y con qué literales *(grado de no determinismo muy alto)*
  - Si la elección se realiza de forma sistemática, la resolución llegará a contradicción si el conjunto de cláusulas es insatisfactible
  - Algoritmo completo, pero puede requerir mucho tiempo...
- Se utilizan diversas estrategias para acelerar el proceso
  - Indexar las cláusulas por los predicados que contienen, indicando si están o no negados, para facilitar su localización
  - Eliminar tautologías *($\varphi \vee \neg \varphi$)* y cláusulas que *subsumen* a otras (φ *es subsumida por* $\varphi \vee \psi$ *porque* $\varphi \rightarrow \varphi \vee \psi$)
  - Intentar resolver con una de las cláusulas de la fórmula que estamos intentando refutar, o con alguna cláusula que se haya obtenido por resolución a partir de una de ellas
    (intuición: la contradicción tiene que salir de ahí)
  - Resolver con cláusulas que tengan un solo literal
    (idea: disminuye el tamaño de las cláusulas generadas)

Contestar a preguntas

- ¿Tiene esqueleto *flipper*?
  - Hay dos opciones para saber si algo se deduce o no de la BC

  1) Demostrar **Tiene_esqueleto(flipper)**

     ➔ añadir **¬Tiene_esqueleto(flipper)**

  2) Demostrar **¬Tiene_esqueleto(flipper)**

     ➔ añadir **Tiene_esqueleto(flipper)**

  - La opción correcta depende de cómo son los predicados de la BC

    Ejemplar(flipper, Delfín)

    Subclase(Delfín, Vertebrado)

    ∀x (Ejemplar(x, Vertebrado) ➔ Tiene_Esqueleto(x))

    ∀x∀y∀z (Ejemplar(x,y) ∧ Subclase(y,z) ➔ Ejemplar(x,z))

    ∀x∀y∀z (Subclase(x,y) ∧ Subclase(y,z) ➔ Subclase(x,z))

Contestar a preguntas

- ¿Existe algún delfín?
  - Demostrar **∃x Ejemplar(x, Delfín)**
    - ➔ añadir **¬∃x Ejemplar(x, Delfín)** en forma clausal
    - ➔ **~ ∀x ¬Ejemplar(x, Delfín)** en forma clausal
    - ➔ añadir **¬Ejemplar(t, Delfín)** variante
  - Cómo preguntar depende de cómo son los predicados de la BC

    Ejemplar(flipper, Delfín)

    Subclase(Delfín, Vertebrado)

    ∀x (Ejemplar(x, Vertebrado) ➔ Tiene_Esqueleto(x))

    ∀x∀y∀z (Ejemplar(x,y) ∧ Subclase(y,z) ➔ Ejemplar(x,z))

    ∀x∀y∀z (Subclase(x,y) ∧ Subclase(y,z) ➔ Subclase(x,z))
  - Si se requiere usar unificación, la resolución devolverá los valores que hacen cierta la pregunta: **{t = flipper}**

Problemas de la representación con LPO

- La resolución no es la forma en que una persona piensa
  - No es válida en sistemas de enseñanza o de diagnóstico médico en los que el sistema debe explicar su razonamiento
  - Proceso de búsqueda no guiado por el razonamiento humano
    - La resolución, con la transformación a forma clausal, no es lo más indicado para que una persona interactúe con la máquina, en procesos semiautomáticos
- Demostración por búsqueda → problemas de eficiencia
  - Necesidad de heurísticas, indexación de la base de conocimiento
  - BC plana: todas las cláusulas tienen la misma importancia
    → añadir meta-conocimiento para dirigir las búsquedas
    - No organizable, imposibilidad de priorizar: explosión combinatoria
- Es difícil representar los distintos tipos de conocimiento
  - Excepciones como cláusulas
  - Conocimiento impreciso → ampliación con otros cuantificadores
  - Conocimiento heurístico → difícil de representar

Representación de conocimiento con Prolog

Prolog

- Sistema de Programación Lógica
  - Conjunto de fórmulas = programa
- Representación del conocimiento situada entre
  - una representación puramente declarativa
    - la lógica de predicados o de primer orden
  - y una representación procedimental
    - Prolog no es declarativo puro porque tiene fijado el mecanismo de inferencia, aparte de por otras cuestiones
- Prolog se suele clasificar dentro de los sistemas de producción con encadenamiento hacia atrás

Representación declarativa vs. procedimental

- Representación declarativa
  - Representamos el conocimiento pero no la forma de utilizarlo
  - Una representación declarativa debe acompañarse con algún programa que especifique qué hacer con el conocimiento y cómo
    - Por ejemplo, fórmulas lógicas + deducción o resolución
- Representación procedimental
  - Otro punto de vista: las fórmulas lógicas no son los datos que se suministran a un programa, son el programa en sí mismo
  - Las implicaciones establecen la forma de razonar (los caminos legítimos para hacerlo) y las fórmulas atómicas nos dan los puntos de partida, razonando hacia delante, o, si razonamos hacia atrás, los puntos de llegada de esos caminos
  - La información de control para usar el conocimiento forma parte de la propia representación del conocimiento

Representación declarativa vs. procedimental

- No determinismo
  - Si hay varias alternativas, como un programa necesita tener determinada una forma de proceder, el intérprete tendrá que tener fijada una estrategia concreta para realizar estas elecciones
  - Por ejemplo, se pueden examinar las fórmulas en el orden textual en el que aparecen en el programa y la búsqueda puede hacerse primero en profundidad
  - Esta estrategia forma parte del sistema y es lo que lo convierte en procedimental
  - Un punto de vista declarativo consideraría todas las alternativas

- Siempre ha habido mucha controversia en IA sobre qué tipo de representación es mejor en la práctica
  - La representación declarativa es más pura pero dificulta codificar conocimiento heurístico

Prolog

- Conocimiento expresable sólo como cláusulas de Horn
  (Prolog puro; la parte impura de Prolog se desvía)
  - Subconjunto decidible de la LPO
  - Cláusulas que tienen como mucho un literal positivo
    - Restricción que lleva a una representación uniforme del conocimiento
    - Esto posibilita la implementación de un intérprete sencillo y eficiente
- Razonamiento hacia atrás *(a partir de un objetivo)*
- Exploración de la BC en orden prefijado *(de arriba a abajo)*
- Búsqueda primero en profundidad con *backtracking*
  - Resolución SLD (estrategia de resolución fija)
    - *Selecting a literal, using a linear strategy, restricted to definite clauses*
  - En profundidad y el subobjetivo más a la izquierda
- Principal ventaja e inconveniente: estrategia de control fija

Representación Prolog: Herencia de predicados de propiedad

- Por ejemplo, dada la base de conocimiento

```
es_un(portaaviones, buque_de_guerra).
es_un(buque_de_guerra, barco).
es_un(barco, vehículo).
propósito(vehículo, transporte).
```

y la regla de herencia para la propiedad *propósito* a través de la relación *es_un*

```
propósito(X, P):- es_un(X, Y), propósito(Y, P).
```

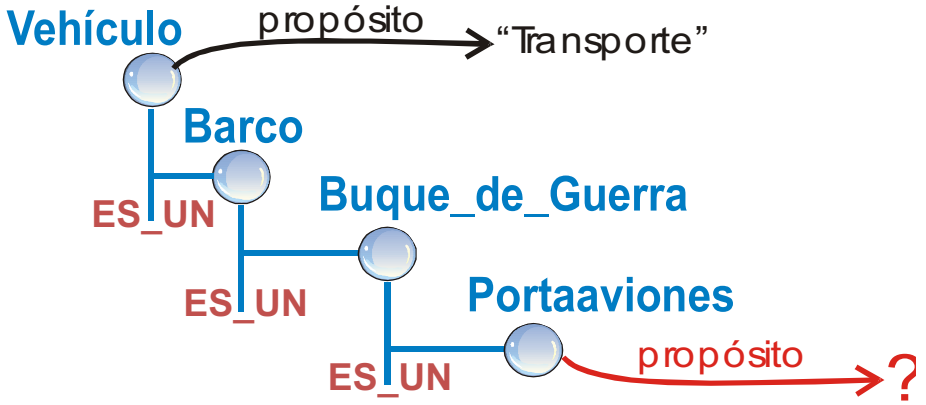
- ¿Qué respondería el sistema ante la siguiente consulta?

```
?- propósito(portaaviones, P).
```

Representación Prolog: Herencia de predicados de propiedad

?- propósito(portaaviones, P).

- El intérprete ascendería por la jerarquía *es_un*, aplicando 3 veces la regla de herencia para obtener **P=transporte**




```
es_un(portaaviones, buque_de_guerra).
es_un(buque_de_guerra, barco).
es_un(barco, vehículo).
propósito(vehículo, transporte).
propósito(X, P):- es_un(X, Y), propósito(Y, P).
```

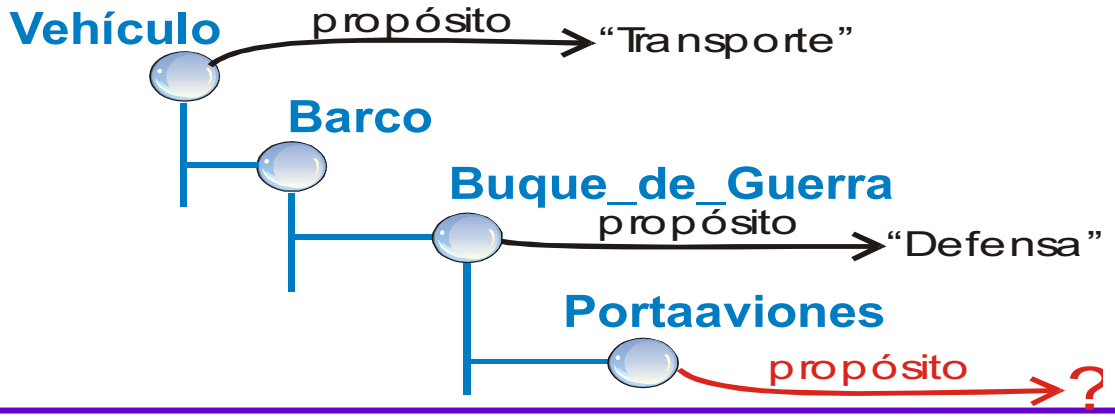
Representación Prolog: Herencia de predicados de propiedad

?- propósito(portaaviones, P).

- Si hubiéramos tenido un hecho

  propósito(buque_de_guerra, defensa).

  el sistema hubiera devuelto primero **P=defensa**



es_un(portaaviones, buque_de_guerra).

es_un(buque_de_guerra, barco).

es_un(barco, vehículo).

propósito(buque_de_guerra, defensa).

propósito(vehículo, transporte).

propósito(X, P):- es_un(X, Y), propósito(Y, P).

SWI Prolog

Robust, mature, free. **Prolog for the real world.**

HOME | DOWNLOAD | DOCUMENTATION | TUTORIALS | COMMUNITY | USERS | WIKI

SWI-Prolog offers a comprehensive free Prolog environment. Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications. Join over a million users who have downloaded SWI-Prolog. more ...

Download SWI-Prolog Get Started Try SWI-Prolog online

https://www.swi-prolog.org/

Bibliografía Prolog

- **Sterling, Leon y Shapiro, Ehud**
  The Art of Prolog : advanced programming techniques
  MIT Press Colección: Logic programming, 2001.
- **Clocksin, William F. y Mellish, Christopher S.**
  Programación en Prolog (*Programming in Prolog*)
  Gustavo Gili (*Springer*) Colección: Ciencia Informática,1994
- **Rowe, Neil C.**
  Artificial Intelligence through Prolog.
  Prentice-Hall, 1988.
- **Bramer, Max**
  Logic Programming with Prolog
  [Recurso electrónico BUCM] Max Bramer, 2005.