Tema 6: Representación del conocimiento mediante reglas

C. Graciani Díaz
F. J. Martín Mateos
J. L. Ruiz Reina

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Sistemas basados en el conocimiento

- Sistemas basados en el conocimiento:
  - Programas que resuelven problemas usando un determinado dominio de conocimiento
  - A veces son llamados *sistemas expertos*
- Ventajas:
  - Fácil acceso y disponibilidad de conocimiento (experto)
  - Coste reducido
  - Permanencia
  - Fiabilidad y rapidez
  - Respuestas no subjetivas
  - Explicación del razonamiento
  - Herramientas de aprendizaje
  - Competitivos con expertos humanos

Sistemas basados en el conocimiento

- Componentes principales de un SBC
  - Conocimiento, que necesita ser *representado*
  - Mecanismos que permitan *inferir* nuevo conocimiento
  - Estos componentes deberían constituir *módulos independientes*
- Otros componentes:
  - Interfaz de usuario
  - Subsistema de explicación del conocimiento inferido
  - Subsistema de adquisición de nuevo conocimiento
  - Herramientas de aprendizaje

Representación del conocimiento

- Requisitos de los formalismos de representación del conocimiento:
  - Potencia expresiva
  - Facilidad de interpretación
  - Eficiencia deductiva
  - Posibilidad de explicación y justificación
- Algunos formalismos de representación:
  - Reglas, redes semánticas, marcos, lógicas de descripción, lógica de primer orden, ...
- Cada formalismo de representación usa un método de inferencia específico:
  - Razonamiento hacia adelante, razonamiento hacia atrás (SLD-resolución), herencia, tableros, resolución, ...

Representación del conocimiento mediante reglas

Reglas

```
SI la luz del semáforo es verde
 Y no hay peatones cruzando
ENTONCES continúa la marcha

SI x es número natural
 Y x es par
ENTONCES x es divisible por 2

SI el reactivo toma color azul
 Y la morfología del organismo es alargada
 Y el paciente es un posible receptor
ENTONCES existe una evidencia (0.7) de que
         la infección proviene de pseudomonas.
```

Hechos

```
La luz del semáforo es verde

El reactivo toma color azul
```

Representación del conocimiento mediante reglas

- Esencialmente, existen dos mecanismos para inferir nuevo conocimiento a partir de un conjunto de reglas
  - Razonamiento hacia atrás (*backward chaining*)
  - Razonamiento hacia adelante (*forward chaining*)
- Se estudian mejor desde un punto de vista formal mediante su formulación *lógica*

Lógica de primer orden

- Lenguaje de la lógica de primer orden
  - Símbolos de *variable*: $\mathbf{x}$, $\mathbf{y}$, ...
  - Símbolos de *función* (con su *aridad*): $\mathbf{f}$, $\mathbf{g}$, ... (las *constantes* son símbolos de función de aridad $\mathbf{0}$)
  - Símbolos de predicado (con su *aridad*): $\mathbf{p}$, $\mathbf{q}$, ...
  - Conectivas: $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$
  - Cuantificadores: $\forall$, $\exists$

Lógica de primer orden

- Términos de la lógica de primer orden
  - Las variables son términos
  - Si $\mathbf{t_1}, \ldots, \mathbf{t_n}$ son términos y $\mathbf{f}$ es un símbolo de función de aridad $\mathbf{n}$, entonces $\mathbf{f(t_1, \ldots, t_n)}$ es un término
- Fórmulas de la lógica de primer orden
  - Si $\mathbf{t_1}, \ldots, \mathbf{t_n}$ son términos y $\mathbf{p}$ es un símbolo de predicado de aridad $\mathbf{n}$, entonces $\mathbf{p(t_1, \ldots, t_n)}$ es una fórmula (atómica)
  - Si $\mathbf{F_1}$ y $\mathbf{F_2}$ y son fórmulas y $\mathbf{x}$ es una variable, entonces $\mathbf{F_1} \vee \mathbf{F_2}$, $\mathbf{F_1} \wedge \mathbf{F_2}$, $\mathbf{F_1} \rightarrow \mathbf{F_2}$, $\neg \mathbf{F_1}$, $\mathbf{F_1} \leftrightarrow \mathbf{F_2}$, $\forall \mathbf{x} \mathbf{F_1}$ y $\exists \mathbf{x} \mathbf{F_1}$ también son fórmulas

Lógica de primer orden

- Una *interpretación* es un conjunto (*universo*) junto con una asignación de funciones y relaciones concretas, en ese universo, a los símbolos de función y predicado
- Una interpretación asigna un valor de verdad a cada fórmula (**V** ó **F**)
- Decimos que una fórmula $\mathbf{G}$ es *consecuencia lógica* de un conjunto de fórmulas $\{\mathbf{F_1}, \mathbf{F_2}, \ldots, \mathbf{F_n}\}$ (y lo escribiremos $\{\mathbf{F_1}, \mathbf{F_2}, \ldots, \mathbf{F_n}\} \models \mathbf{G}$) si para toda interpretación $\mathbf{I}$ tal que $\mathbf{I(F_i) = V}$ para todo $\mathbf{i = 1, \ldots, n}$, entonces se tiene que $\mathbf{I(G) = V}$

Reglas, hechos y bases de conocimiento

- Una *regla* es una fórmula de primer orden de la forma

$$\mathbf{P_1} \wedge \mathbf{P_2} \wedge \ldots \wedge \mathbf{P_n} \rightarrow \mathbf{Q}$$

  donde $\mathbf{P_1}, \ldots, \mathbf{P_n}, \mathbf{Q}$ son fórmulas atómicas
  - Las variables de una regla se interpretan universalmente cuantificadas
  - Cabeza de la regla: $\mathbf{Q}$
  - Cuerpo de la regla: $\mathbf{P_1} \wedge \mathbf{P_2} \wedge \ldots \wedge \mathbf{P_n}$
- Un *hecho* es una fórmula atómica
  - Un hecho puede verse como una regla sin cuerpo
- Una *base de conocimiento* es un conjunto de hechos y reglas

Reglas, hechos y bases de conocimiento

Una pequeña base de conocimiento

R1: bueno(x) $\wedge$ rico(y) $\wedge$ quiere(x,y) $\rightarrow$ hereda-de(x,y)
R2: amigo(x,y) $\rightarrow$ quiere(x,y)
R3: antecesor(y,x) $\rightarrow$ quiere(x,y)
R4: progenitor(x,y) $\rightarrow$ antecesor(x,y)
R5: progenitor(x,z) $\wedge$ progenitor(z,y) $\rightarrow$ antecesor(x,y)
H1: progenitor(padre(x),x)
H2: rico(Pedro)
H3: rico(padre(padre(Juan)))
H4: amigo(Juan,Pedro)
H5: bueno(Juan)

Deducción en una base de conocimiento

- Sustituciones: $\theta = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$
  - Si Q es una fórmula atómica, $\theta(Q)$ es la fórmula obtenida sustituyendo cada aparición de x_i en Q por t_i
- Regla de inferencia: *Modus Ponens Generalizado*

$$\frac{P_1 \quad P_2 \quad \ldots \quad P_n \quad P'_1 \wedge P'_2 \wedge \ldots \wedge P'_n \rightarrow Q}{\theta(Q)}$$

  donde θ es una sustitución tal que $\theta(P_i) = \theta(P'_i)$, para todo i
- Dada una base de conocimiento BC y una fórmula atómica P, decimos que $BC \vdash P$ si existe una secuencia de fórmulas $F_1, \ldots, F_n$ tales que $F_n = P$ y cada F_i está en BC o se obtiene de fórmulas anteriores mediante aplicación de Modus Ponens Generalizado
  - Detalle técnico: cada vez que se usa una fórmula, se pueden *renombrar* sus variables (de manera consistente)

Deducción en una base de conocimiento

Una pequeña base de conocimiento

R1: $\texttt{bueno(x)} \wedge \texttt{rico(y)} \wedge \texttt{quiere(x,y)} \rightarrow \texttt{hereda-de(x,y)}$
R2: $\texttt{amigo(x,y)} \rightarrow \texttt{quiere(x,y)}$
R3: $\texttt{antecesor(y,x)} \rightarrow \texttt{quiere(x,y)}$
R4: $\texttt{progenitor(x,y)} \rightarrow \texttt{antecesor(x,y)}$
R5: $\texttt{progenitor(x,z)} \wedge \texttt{progenitor(z,y)} \rightarrow \texttt{antecesor(x,y)}$
H1: `progenitor(padre(x),x)`
H2: `rico(Pedro)`
H3: `rico(padre(padre(Juan)))`
H4: `amigo(Juan,Pedro)`
H5: `bueno(Juan)`

- En este caso, a partir de $\texttt{antecesor(y,x)} \rightarrow \texttt{quiere(x,y)}$ y de `antecesor(Pedro,Juan)` se deduce `quiere(Juan,Pedro)` (mediante la sustitución $\theta = \{\texttt{x} \mapsto \texttt{Juan}, \texttt{y} \mapsto \texttt{Pedro}\}$)

Deducción en una base de conocimiento

- Teorema: $\mathbf{BC} \models \mathbf{P} \iff \mathbf{BC} \vdash \mathbf{P}$
  - Este teorema nos sugiere cómo implementar algoritmos para responder preguntas a una base de conocimiento
  - Esencialmente, se trata de implementar procesos que *buscan* una secuencia de aplicaciones válidas de la regla de Modus Ponens que obtenga **P** como fórmula final
  - Por tanto, algoritmos de *búsqueda*
- Esta búsqueda puede realizarse:
  - *Hacia atrás*: partiendo de **P**
  - *Hacia adelante*: partiendo de **BC**

Razonamiento hacia atrás con reglas de primer orden

- Podemos diseñar un algoritmo de encadenamiento hacia atrás basado en la regla de Modus Ponens Generalizado
- Dado un objetivo $\mathbf{Q}$ y una regla $\mathbf{Q_1 \wedge \ldots \wedge Q_n \rightarrow P}$, los nuevos objetivos podrían ser $\{\boldsymbol{\theta}\mathbf{(Q_1)}, \ldots, \boldsymbol{\theta}\mathbf{(Q_n)}\}$ si encontramos una sustitución $\boldsymbol{\theta}$ tal que $\boldsymbol{\theta}\mathbf{(P)} = \boldsymbol{\theta}\mathbf{(Q)}$
  - Dicha sustitución se denomina *unificador* de $\mathbf{P}$ y $\mathbf{Q}$
- Por tanto, se necesita un algoritmo que dado el objetivo y la cabeza de la regla, encuentre un unificador, si es que existe
  - Dicho algoritmo se denomina *algoritmo de unificación*

Unificación

- Ejemplos:

Objetivo	Cabeza de regla
amigo(x,Juan)	**amigo(Antonio,y)**
Unificador: {**x** $\mapsto$ **Antonio**, **y** $\mapsto$ **Juan**}	
progenitor(padre(x),x)	**progenitor(y,Juan)**
Unificador: {**x** $\mapsto$ **Juan**, **y** $\mapsto$ **padre(Juan)**}	
quiere(x,z)	**amigo(Antonio,y)**
Unificador: No existe	
quiere(x,z)	**quiere(padre(y),x)**
Unificador: {**x** $\mapsto$ **padre(Juan)**, **y** $\mapsto$ **Juan**, **z** $\mapsto$ **padre(Juan)**}	
quiere(x,z)	**quiere(padre(y),x)**
Unificador: {**x** $\mapsto$ **padre(y)**, **z** $\mapsto$ **padre(y)**}	

- Dos átomos cualesquiera podrían no ser unificables, o serlo y tener más de un unificador
  - Se puede demostrar que si son unificables entonces existe un unificador *más general* que cualquier otro

Algoritmo de unificación

Algoritmo de unificación

```
FUNCIÓN UNIFICA(S,T)
  Devolver UNIFICA-REC(S,T,{})

FUNCIÓN UNIFICA-REC(S,T,θ)
1 Si S = T, devolver {}
2 Si no, si S es variable, entonces:
  2.1 Si S ocurre en T, devolver FALLO
  2.2 Si S no ocurre en T, devolver {S↦T}
3 Si no, si T es variable, entonces devolver
  UNIFICA-REC(T,S,θ)
4 Si no, sean S = f(S1,...,Sn) y T = g(T1,...,Tm)
  4.1 Si f ≠ g ó n ≠ m, devolver FALLO
  4.2 En otro caso, devolver
      UNIFICA-REC-LISTA([S1,...,Sn],[T1,...,Tn],θ)

FUNCIÓN UNIFICA-REC-LISTA(LS,LT,θ)
1 Si LS es la lista vacía, devolver θ
2 Si no, sea σ = UNIFICA-REC(PRIMERO(LS),PRIMERO(LT),θ)
  2.1 Si σ es igual a FALLO, devolver FALLO
  2.2 En caso contrario, devolver
      UNIFICA-REC-LISTA(σ(RESTO(LS)),σ(RESTO(LT)),COMPONER(θ,σ))
```

- Nota: **COMPONER(S1,S2)** obtiene la sustitución que actúa como si se aplicara **S1** y a continuación **S2**

Algoritmo de unificación

Traza del algoritmo

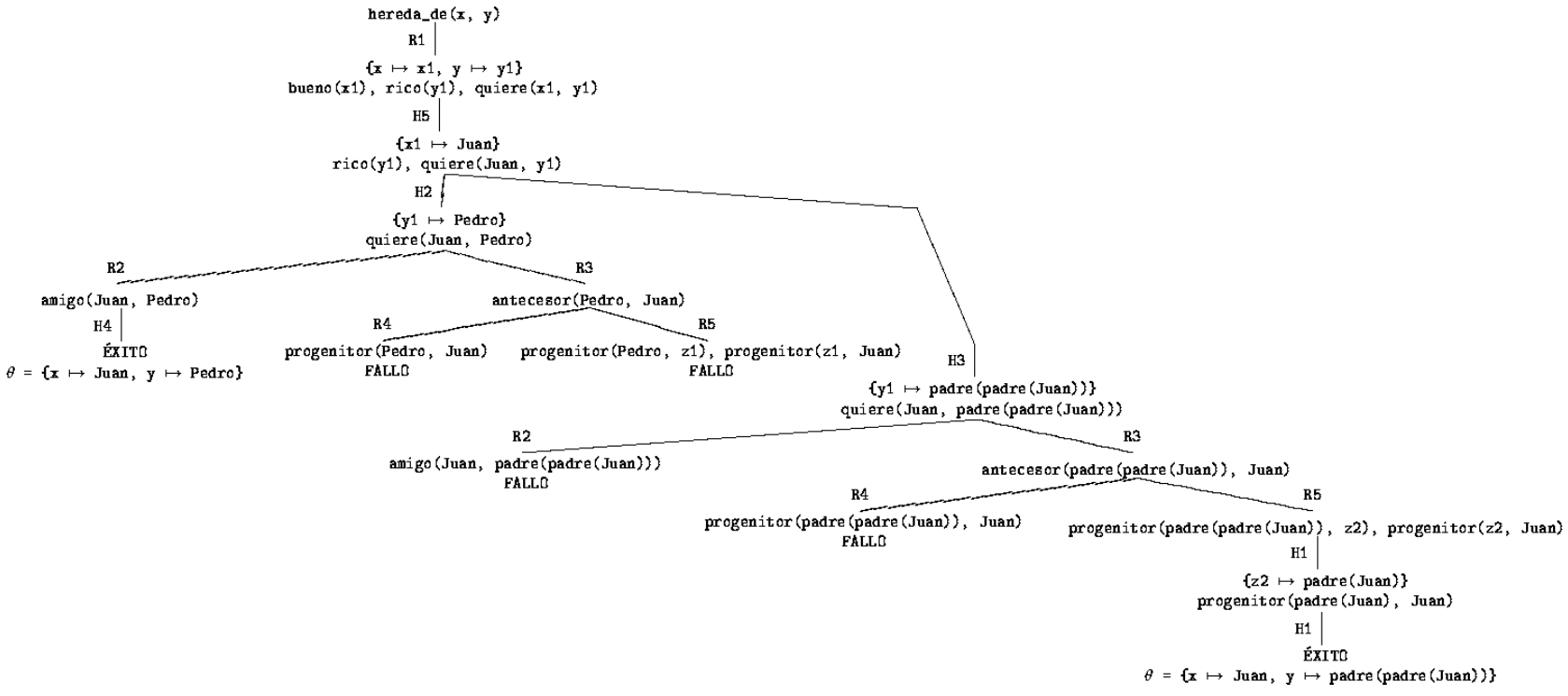
```
UNIFICA(quiere(x,z),quiere(padre(y),x)) ⇒
  UNIFICA-REC(quiere(x,z),quiere(padre(y),x),{}) ⇒
    UNIFICA-REC-LISTA({x,z},{padre(y),x},{}) ⇒
      [UNIFICA-REC(x,padre(y),{}) = {x↦padre(y)}]
    UNIFICA-REC-LISTA({z},{padre(y)},{x↦padre(y)}) ⇒
      [UNIFICA-REC(z,padre(y),{x↦padre(y)}) = {z↦padre(y)}]
    UNIFICA-REC-LISTA({},{},{x↦padre(y),z↦padre(y)}) ⇒
      {x↦padre(y),z↦padre(y)}
```

- Teorema: **UNIFICA(S,T)** no devuelve **FALLO** si y sólo si **S** y **T** se pueden unificar; en ese caso, devuelve un unificador de **S** y **T** más general que cualquier otro unificador

SLD-resolución

- Combinando unificación y encadenamiento hacia atrás podemos obtener un procedimiento que permite responder a preguntas del tipo ¿$\mathbf{BC} \models \mathbf{P}$?
  - En concreto, el algoritmo encontrará sustituciones θ tal que $\mathbf{BC} \models \theta(\mathbf{P})$. A estas sustituciones se las llama *respuestas*
  - Este algoritmo se denomina *SLD-resolución*
- El algoritmo realiza una búsqueda en profundidad (backtracking)
  - Como es habitual, esta búsqueda se puede representar mediante un árbol

Un ejemplo de árbol SLD



Árboles SLD

- Un árbol SLD representa el proceso de búsqueda
  - Los nodos de un árbol representan los objetivos pendientes (átomos por deducir)
- Nodo raíz: objetivo inicial
- Nodos finales:
  - Un nodo de *fallo* se obtiene cuando su primer átomo no es unificable con la cabeza de ninguna regla o hecho
  - Un nodo de *éxito* es aquél que no tiene objetivos pendientes
- Ramas de éxito
  - Una *rama de éxito* es aquella que termina en un nodo de *éxito*
  - Cada rama de éxito "construye" una sustitución (*respuesta*) que se va "concretando" mediante los sucesivos pasos de unificación
  - Cada rama de éxito representa una *SLD-refutación*
- Renombrado de reglas y hechos

Algoritmo de SLD-resolución

Algoritmo de SLD-resolución

FUNCIÓN SLD-RESOLUCIÓN(BC,Q)
** Devolver SLD-RESOLUCIÓN-REC(BC,{Q},{})**

FUNCIÓN SLD-RESOLUCIÓN-REC(BC,OBJETIVOS,θ)
1 Si OBJETIVOS está vacío, devolver la lista unitaria {θ}
2 Hacer RESPUESTAS igual a la lista vacía
** Hacer ACTUAL igual θ (SELECCIONA-UNO(OBJETIVOS))**
3 Para cada regla $P_1,\ldots,P_n \rightarrow P$ en BC (renombrada con
** variables nuevas) tal que σ = UNIFICA(P,ACTUAL) es distinto**
** de FALLO**
** 3.1 Hacer NUEVOS-OBJETIVOS igual a**
** $\{P_1,\ldots,P_n\} \cup$ (OBJETIVOS$\backslash${ACTUAL})**
** 3.2 Añadir a RESPUESTAS el resultado de**
** SLD-RESOLUCIÓN-REC(BC,NUEVOS-OBJETIVOS,COMPONER(θ,σ))**
4 Devolver RESPUESTAS

Propiedades del algoritmo de SLD-resolución

- Teorema (*corrección*): Si $\theta \in$ **SLD-RESOLUCION(BC,P)** entonces $\mathbf{BC} \vdash \theta(\mathbf{P})$
- Teorema: Si $\mathbf{BC} \vdash \sigma(\mathbf{P})$, entonces existe una SLD-refutación a partir de **P** que construye una respuesta θ tal que σ es "un caso particular" de θ
  - Problema: el algoritmo de SLD-resolución podría no encontrar tal SLD-refutación debido a la existencia de ramas infinitas
- Estrategia de búsqueda:
  - En nuestro pseudocódigo, búsqueda en profundidad
  - Selección del objetivo a resolver: cualquier función de selección serviría
  - Orden en el que se usan las reglas y hechos de la BC

Programación lógica

- El algoritmo de SLD-resolución es el elemento en torno al cual se desarrolla el paradigma declarativo de *programación lógica*
  - Programa lógico: conjunto de reglas y hechos
  - Declarativo: "qué es" en lugar de "cómo se hace"
- Una base de conocimiento puede verse como un *programa lógico*
- Las respuestas que calcula el algoritmo de SLD-resolución pueden verse como la salida que calcula el programa
- PROLOG es el lenguaje de programación más conocido basado en el paradigma de la programación lógica
- Debido a que el encadenamiento hacia atrás forma parte del propio intérprete, PROLOG es un lenguaje especialmente adecuado para la implementación de SBCs basados en reglas con razonamiento hacia atrás

Deducción hacia adelante

- En contraposición al encadenamiento hacia atrás, podemos aplicar la regla de Modus Ponens Generalizado hacia adelante
  - A partir de los hechos, y usando las reglas, obtener nuevos hechos
  - De la misma manera, los hechos deducidos permiten obtener nuevos hechos
- Problemas adecuados para razonar hacia adelante
  - Monitorización y control
  - Problemas dirigidos por los datos
  - Sin necesidad de explicación
- Problemas adecuados para razonar hacia atrás
  - Diagnóstico
  - Problemas dirigidos por los objetivos
  - Interacción/Explicación al usuario

Deducción hacia adelante

Algoritmo de encadenamiento hacia adelante

FUNCIÓN ENCADENAMIENTO-HACIA-ADELANTE(BC,P)
1 Hacer DEDUCIDOS igual a vacío
2 Para cada regla $P_1, \ldots, P_n \rightarrow Q$ en BC (renombrada con variables nuevas)
2.1 Para cada θ tal que $\theta(P_i) = \theta(R_i)$ para ciertos R_i en BC, i=1,...,n,
2.1.1 Hacer $S = \theta(Q)$
2.1.2 Si S (o un renombrado) no está ni en BC ni en DEDUCIDOS, añadir S a DEDUCIDOS
2.1.3 Si σ=UNIFICA(P,S) es distinto de FALLO, devolver σ y terminar
3 Si DEDUCIDOS es vacío, devolver FALLO y terminar.
en caso contrario, añadir DEDUCIDOS a BC y volver al punto 1

Deducción hacia adelante

Ejemplo: una pequeña base de conocimiento

R1: bueno(x) $\wedge$ rico(y) $\wedge$ quiere(x,y) $\rightarrow$ hereda-de(x,y)
R2: amigo(x,y) $\rightarrow$ quiere(x,y)
R3: antecesor(y,x) $\rightarrow$ quiere(x,y)
R4: progenitor(x,y) $\rightarrow$ antecesor(x,y)
R5: progenitor(x,z) $\wedge$ progenitor(z,y) $\rightarrow$ antecesor(x,y)
H1: progenitor(padre(x),x)
H2: rico(Pedro)
H3: rico(padre(padre(Juan)))
H4: amigo(Juan,Pedro)
H5: bueno(Juan)

Ejemplo: una pequeña base de conocimiento

Traza del algoritmo

```
ENCADENAMIENTO-HACIA-ADELANTE(BC,hereda-de(x,y))
 H6: quiere(Juan, Pedro)
   | R2 {x1↦Juan, y1↦Pedro} H4
 H7: antecesor(padre(x), x)
   | R4 {x1↦padre(x), y1↦x}, H1
 H8: antecesor(padre(padre(x)), x)
   | R5 {x1↦padre(padre(x2)), z1↦padre(x2), y1↦x2},
   | H1 {x↦padre(x2)}, H1 {x↦x2}
 H9: hereda-de(Juan, Pedro)
   | R1 {x1↦Juan, y1↦Pedro}, H5, H2, H6
   θ = {x↦Juan, y↦Pedro}
H10: quiere(x, padre(x))
   | R3 {x1↦x, y1↦padre(x)} H7
H11: quiere(x, padre(padre(x)))
   | R3 {x1↦x, y1↦padre(padre(x))} H8
H12: hereda-de(Juan, padre(padre(Juan)))
   | R1 {x1↦Juan, y1↦padre(padre(Juan))},
   | H5, H3, H11 {x↦Juan}
   θ = {x↦Juan, y↦padre(padre(Juan))}
```

Sistemas de producción

- Un *sistema de producción* es un mecanismo computacional basado en *reglas de producción* de la forma: "Si se cumplen las *condiciones* entonces se ejecutan las *acciones*"
- El conjunto de las reglas de producción forma la *base de conocimiento* que describe como evoluciona un sistema
  - Las reglas de producción actúan sobre una *memoria de trabajo* o *base de datos* que describe el estado actual del sistema
  - Si la condición de una regla de producción se satisface entonces dicha regla está *activa*
  - El conjunto de reglas de producción activas en un instante concreto forma el *conjunto de conflicto* o *agenda*
  - La *estrategia de resolución* selecciona una regla del conjunto de conflicto para ser ejecutada o *disparada* modificando así la memoria de trabajo

Sistemas de producción

- Paradigma de los sistemas de producción
  - Hechos: pieza básica de información
  - Reglas: describen el comportamiento del programa en función de la información existente
- Modelo de hecho:

```
<índice>: <símbolo>(<elemento>*)
```

- Modelo de regla:

```
REGLA <Nombre>:
SI <Condición>*
ENTONCES
   <Acción>*
```

Elementos de una regla

- Condiciones:
  - Existencia de cierta información: **<patrón>**
  - Ausencia de cierta información: **no(<patrón>)**
  - Relaciones entre datos
- Acciones:
  - Incluir nueva información: **INCLUIR: <hecho>)**
  - Eliminar información: **ELIMINAR: <hecho>)**

Ejemplo de sistema de producción

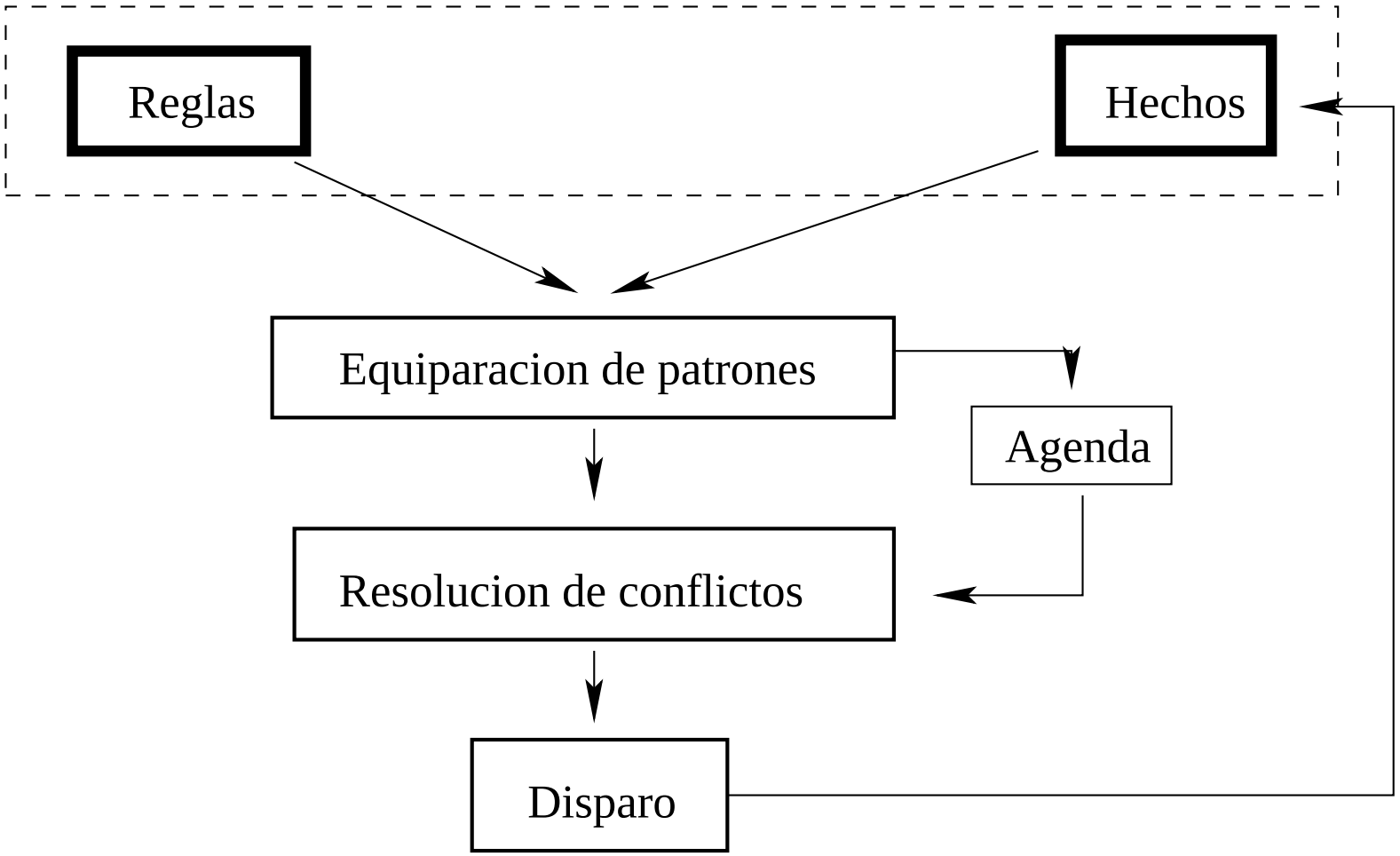
Ejemplo

```
Base de Hechos:
    1: Tiene(pelos)
    2: Tiene(pezuñas)
    3: Tiene(rayas-negras)

Base de Reglas:
    REGLA Jirafa:
       Es(ungulado)
       Tiene(cuello-largo)
       =>
       INCLUIR: Es(jirafa)

    REGLA Cebra:
       Es(ungulado)
       Tiene(rayas-negras)
       =>
       INCLUIR: Es(cebra)

    REGLA Ungulado-1:
       Es(mamífero)
       Tiene(pezuñas)
       =>
       INCLUIR: Es(ungulado)

    REGLA Ungulado-2:
       Es(mamífero)
       Rumia()
       =>
       INCLUIR: Es(ungulado)

    REGLA Mamífero-1:
       Tiene(pelos)
       =>
       INCLUIR: Es(mamífero)

    REGLA Mamífero-2:
       Da(leche)
       =>
       INCLUIR: Es(mamífero)
```

Sistemas de producción

- Componentes:
  - Base de hechos (*memoria de trabajo*). Elemento dinámico
  - Base de reglas. Elemento estático
  - Motor de inferencia (produce los cambios en la memoria de trabajo)
- Elementos adicionales:
  - *Algoritmo de equiparación de patrones*: Algoritmo para calcular **eficientemente** la agenda
  - *Estrategia de resolución de conflictos*: Proceso para decidir en cada momento qué regla de la agenda debe ser disparada

Ciclo de ejecución



Resolución de conflictos

- Una activación sólo se produce una vez
- Estrategias más comunes:
  - Tratar la agenda como una pila
  - Tratar la agenda como una cola
  - Elección aleatoria
  - Regla más específica (número de condiciones)
  - Activación más reciente (en función de los hechos)
  - Regla menos utilizada
  - Mejor (pesos)

Tabla de seguimiento

Base de Hechos	E	Agenda	D
1: Tiene(pelos)	0	**Mamífero-1: 1**	1
2: Tiene(pezuñas)	0		
3: Tiene(rayas-negras)	0		
4: Es(mamífero)	1	**Ungulado-1: 4,2**	2
5: Es(ungulado)	2	**Cebra: 5,3**	3
6: Es(cebra)	3		

Variables: simples, múltiples, anónimas, con restricciones

- Variable simple: **`?x, ?y, ?elemento`**
- Variable anónima simple: **`?`**
- Variable múltiple: **`$?x, $?y, $?elemento`**
- Variable anónima múltiple: **`$?`**
- Variables con restricciones: **`?x&a|b, ?x&~a, ?x&:(< ?x 3)`**

Sistema de producción con variables

```
Base de Hechos:
    1: Lista(Mar Ana Luis Pepe)
    2: Alumno(Mar)
    3: Alumno(Ana 2 3 9)
    4: Alumno(Luis)
    5: Alumno(Pepe 3)

Base de Reglas:
    Elimina:
    SI Alumno(?n ? $?)
       Lista($?i ?n $?f)
    ENTONCES
       ELIMINAR: Lista(?i ?n ?f)
       INCLUIR: Lista(?i ?f)
```

Tabla de seguimiento

Base de Hechos	E	S	Agenda	D	S
1:Lista(Mar Ana Luis Pepe)	0	1			
2:Alumno(Mar)	0				
3:Alumno(Ana 2 3 9)	0		**Elimina: 3,1**		
			?n $\mapsto$ **Ana**		
			$?i $\mapsto$ **Mar**		
			$?f $\mapsto$ **Luis Pepe**		1
4:Alumno(Luis)	0				
5:Alumno(Pepe 3)	0		**Elimina: 5,1**		
			?n $\mapsto$ **Pepe**		
			$?i $\mapsto$ **Mar Ana Luis**		
			$?f $\mapsto$	1	
6:Lista(Mar Ana Luis)	1	2	**Elimina: 3,6**		
			?n $\mapsto$ **Ana**		
			$?i $\mapsto$ **Mar**		
			$?f $\mapsto$ **Luis**	2	
7:Lista(Mar Luis)	2				

Equiparación de patrones

- La forma más simple de calcular el conjunto de conflicto supone analizar las condiciones de todas las reglas en la memoria de trabajo actual, este proceso es muy costoso

Algoritmo de fuerza bruta

```
Por cada regla Ri hacer
  Por cada patrón de hecho Pj en Ri hacer
    Por cada hecho Hk en la memoria de trabajo hacer
      Comprobar si Pj equipara con Hk
  Si todos los patrones tienen equiparación incluir
  en la agenda todas las posibles activaciones de la
  regla Ri
```

Equiparación de patrones

- Para hacer más eficiente el proceso de equiparación de patrones se pueden tener en cuenta las siguientes propiedades
  - Redundancia Temporal: el disparo de una regla usualmente cambia pocos hechos y son pocas las reglas afectadas por esos cambios
  - Similitud Estructural: una misma condición aparece frecuentemente en más de una regla
- El algoritmo RETE aprovecha estas características para limitar el esfuerzo requerido para calcular el conjunto de conflicto después de que una regla es disparada

CLIPS

- CLIPS $\equiv$ C Language Integrated Production Systems
  - `http://www.ghg.net/clips/CLIPS.html`
- Lenguaje basado en reglas de producción.
- Desarrollado en el *Johnson Space Center* de la NASA.
- Relacionado con OPS5 y ART
- Sintáxis específica de CLIPS: **`deffacts`**, **`defrule`**, **`assert`**, **`retract`**
- Características:
  - Conocimiento: Reglas, objetos y procedimental
  - Portabilidad: implementado en C
  - Integración y Extensibilidad: C, Java, FORTRAN, ADA
  - Documentación
  - Bajo coste: software libre

Bibliografía

- Russell, S. y Norvig, P.
  Inteligencia artificial: Un enfoque moderno (segunda edición)
  (Prentice Hall, 2004).
  - Cap. 9: "Inferencia en lógica de primer orden"
- Russell, S. y Norvig, P.
  Artificial Intelligence (A Modern Approach)
  (Prentice–Hall, 2010). Third Edition.
  - Cap. 9: "Inference in First-Order Logic".
- Giarrantano, J.C. y Riley, G.
  Expert Systems Principles and Programming (3 ed.)
  (PWS Pub. Co., 1998)