Inferencia en Lógica de Primer Orden I

BLOQUE TEMÁTICO:	**Representación del Conocimiento y Razonamiento**
TEMA 3.2:	**Representación y Razonamiento mediante Lógica**
LECCIÓN 16:	**Inferencia en Lógica de Predicados I**

DESARROLLO DEL TEMA:

1. Reducir inferencia de primer orden a inferencia proposicional.
2. Unificación.
3. Modus Ponens generalizada.
4. Encadenamiento hacia delante

Breve historia del razonamiento

450 B.C.	Stoics	lógica proposicional, inferencia (quizás)
322 B.C.	Aristóteles	“silogismos” (reglas de inferencia), cuantificadores
1565	Cardano	teoría de la probabilidad (lógica proposicional + incertidumbre)
1847	Boole	lógica proposicional (otra vez)
1879	Frege	lógica de primer orden
1922	Wittgenstein	prueba para tablas de verdad
1930	Gödel	$\exists$ algoritmo completo para LPO
1930	Herbrand	algoritmo completo para LPO (reducción a proposicional)
1931	Gödel	$\neg\exists$ algoritmo completo para aritmética
1960	Davis/Putnam	algoritmo “práctico” para lógica proposicional (DPLL)
1965	Robinson	algoritmo “práctico” para LPO, resolución

Especificación Universal (EU)

Podemos inferir cualquier sentencia obtenida por sustitución de la variable por un término base (un término sin variables).
Denotamos el resultado de aplicar una sustitución θ a una sentencia α mediante $SUST(\theta, \alpha)$.
Cada especificación de una sentencia cuantificada universalmente es implicada por ella:

$$\frac{\forall v \alpha}{SUST(\{v/g\},\alpha)}$$

para cualquier variable v y término base g.

Ejemplo, $\forall x Rey(x) \wedge Codicioso(x) \Rightarrow Malvado(x)$ infiere cualquiera de las siguientes sentencias:
$Rey(Juan) \wedge Codicioso(Juan) \Rightarrow Malvado(Juan)$
$Rey(Ricardo) \wedge Codicioso(Ricardo) \Rightarrow Malvado(Ricardo)$
$Rey(Padre(Juan)) \wedge Codicioso(Padre(Juan)) \Rightarrow Malvado(Father(Juan))$
$\vdots$

Especificación Existencial (EE)

Para cualquier sentencia α, variable v, y símbolo constante k, *que no aparezca en ninguna otra parte de la base de conocimiento*:

$$\frac{\exists v \alpha}{SUST(\{v/k\},\alpha)}$$

Ejemplo, $\exists x Corona(x) \wedge SobreCabeza(x, Juan)$ implic

$$Corona(C_1) \wedge SobreCabeza(C_1, Juan)$$

mientras que C_1 sea un nuevo símbolo constante, llamado Constante de Skolem.

Otro ejemplo: de $\exists x \; d(x^y)/dy = x^y$, obtenemos

$$d(e^y)/dy = e^y$$

mientras que e sea un símbolo constante nuevo.

Especificación Existencial (cont.)

La Especificación Universal puede ser aplicada muchas veces y *añadir* nuevas sentencias. La nueva base de conocimiento es **lógicamente equivalente** a la original.

Sin embargo, la Especificación Existencial puede ser aplicada una única vez y entonces se puede *descartar* la sentencia cuantificada existencialmente. La nueva base de conocimiento no es equivalente a la anterior, pero se puede demostrar que es **equivalente inferencialmente** en el sentido de que es satisfacible justamente cuando lo es la base de conocimiento original.

Reducción a la inferencia proposicional

Una vez que tenemos las reglas para inferir sentencias no cuantificadas a partir de sentencias cuantificadas, nos es posible reducir la inferencia de primer orden a la inferencia proposicional.

Supongamos que la BC contiene sólo lo siguiente;

$\forall x Rey(x) \wedge Codicioso(x) \Rightarrow Malvado(x)$
$Rey(Juan)$
$Codicioso(Juan)$
$Hermano(Ricardo, Juan)$

Especificando la sentencia universal de *todas las maneras posibles*, tenemos:

$Rey(Juan) \wedge Codicioso(Juan) \Rightarrow Malvado(Juan)$
$Rey(Ricardo) \wedge Codicioso(Ricardo) \Rightarrow Malvado(Ricardo)$
$Rey(Juan)$
$Codicioso(Juan)$
$Hermano(Ricardo, Juan)$

La nueva BC está proposicionalizada. Símbolos proposición son: $Rey(Juan), Codicioso(Juan), Malvado(Juan), Rey(Ricardo), etc.$

Reducción (cont.)

Una sentencia base es implicada por la nueva BC si y sólo si es implicada por la BC original.

Toda BC de LPO se puede transformar a forma proposicional de tal manera que se mantenga la relación de implicación.

Idea: proposicionalizar la BC y la petición, aplicar resolución y devolver el resultado.

¡Hay un problema!
Problema: cuando la base de conocimiento incluye un símbolo de función, el conjunto de sistituciones de los términos base es infinito.

Ejemplo, $Padre(Padre(Padre(Juan)))$ (términos anidados infinitamente).

Reducción (cont.)

Teorema: Herbrand (1930). Si una sentencia se implica de una BC de primer orden, entonces hay una demostración que involucra tan sólo a un subconjunto finito de la BC transformada a proposicional.

Idea: Para $n = 0$ a ∞ hacer:
Crear una BC proposicional especificando los términos con profundidad n.
Ver si α es implicada por esta BC.

Problema: funciona si α es implicada, bucle si no lo es.

Teorema: Turing (1936), Church (1936). El problema de la implicación en LPO es semidecidible, es decir, existen algoritmos que responden afirmativamente para cada sentencia implicada, pero no existe ningún algoritmo que también responda ante una sentencia no implicada.

Problemas con la proposicionalización

La Proposicionalización parace generar muchas sentencias irrelevantes. Por ejemplo:

$\forall x Rey(x) \wedge Codicioso(x) \Rightarrow Malvado(x)$
$Rey(Juan)$
$\forall y Codicioso(y)$
$Hermano(Ricardo, Juan)$

Parace obvio que $Malvado(Juan)$, pero la proposicionalización produce un montón de hechos tales como $Codicioso(Ricardo)$, que son irrelevantes.

Con p k-ésimos predicados y n constantes, hay $p \cdot n^k$ especificaciones.

Con los símbolos de función, es aún pero!

Unificación

Podemos obtener la inferencia inmediatamente si encontramos una sustitución θ, tal que $Rey(x)$ y $Codicioso(x)$ sean iguales a $Rey(Juan)$ y $Codicioso(y)$.

En este caso, la sustitución $\{x/Juan, y/Juan\}$ logra el objetivo.

UNIFICAR$(\alpha, \beta) = \theta$ si $\alpha\theta = \beta\theta$
El algoritmo *UNIFICA* toma dos sentencias y devuelve un **unificador** para ellas, si éste existe.

Supongamos que tenemos una petición $Conoce(Juan, x)$: ¿A quién conoce Juan?

p	q	θ
$Conoce(Juan, x)$	$Conoce(Juan, Juana)$	
$Conoce(Juan, x)$	$Conoce(y, OJ)$	
$Conoce(Juan, x)$	$Conoce(y, Madre(y))$	
$Conoce(Juan, x)$	$Conoce(x, OJ)$	

Unificación

Podemos obtener la inferencia inmediatamente si encontramos una sustitución θ, tal que $Rey(x)$ y $Codicioso(x)$ sean iguales a $Rey(Juan)$ y $Codicioso(y)$.

En este caso, la sustitución $\{x/Juan, y/Juan\}$ logra el objetivo.

UNIFICAR$(\alpha, \beta) = \theta$ si $\alpha\theta = \beta\theta$

Supongamos que tenemos una petición $Conoce(Juan, x)$: ¿A quién conoce Juan?

p	q	θ
$Conoce(Juan, x)$	$Conoce(Juan, Juana)$	{x/Juana}
$Conoce(Juan, x)$	$Conoce(y, OJ)$	
$Conoce(Juan, x)$	$Conoce(y, Madre(y))$	
$Conoce(Juan, x)$	$Conoce(x, OJ)$	

Unificación

Podemos obtener la inferencia inmediatamente si encontramos una sustitución θ, tal que $Rey(x)$ y $Codicioso(x)$ sean iguales a $Rey(Juan)$ y $Codicioso(y)$.

En este caso, la sustitución $\{x/Juan, y/Juan\}$ logra el objetivo.

UNIFICAR$(\alpha, \beta) = \theta$ si $\alpha\theta = \beta\theta$

Supongamos que tenemos una petición $Conoce(Juan, x)$: ¿A quién conoce Juan?

p	q	θ
$Conoce(Juan, x)$	$Conoce(Juan, Juana)$	{x/Juana}
$Conoce(Juan, x)$	$Conoce(y, OJ)$	{x/OJ,y/Juan}
$Conoce(Juan, x)$	$Conoce(y, Madre(y))$	
$Conoce(Juan, x)$	$Conoce(x, OJ)$	

Unificación

Podemos obtener la inferencia inmediatamente si encontramos una sustitución θ, tal que $Rey(x)$ y $Codicioso(x)$ sean iguales a $Rey(Juan)$ y $Codicioso(y)$.

En este caso, la sustitución $\{x/Juan, y/Juan\}$ logra el objetivo.

UNIFICAR$(\alpha, \beta) = \theta$ si $\alpha\theta = \beta\theta$

Supongamos que tenemos una petición $Conoce(Juan, x)$: ¿A quién conoce Juan?

p	q	θ
$Conoce(Juan, x)$	$Conoce(Juan, Juana)$	{x/Juana}
$Conoce(Juan, x)$	$Conoce(y, OJ)$	{x/OJ,y/Juan}
$Conoce(Juan, x)$	$Conoce(y, Madre(y))$	{y/Juan,x/Madre(Juan)}
$Conoce(Juan, x)$	$Conoce(x, OJ)$	

Unificación

Podemos obtener la inferencia inmediatamente si encontramos una sustitución θ, tal que $Rey(x)$ y $Codicioso(x)$ sean iguales a $Rey(Juan)$ y $Codicioso(y)$.

En este caso, la sustitución $\{x/Juan, y/Juan\}$ logra el objetivo.

UNIFICAR$(\alpha, \beta) = \theta$ si $\alpha\theta = \beta\theta$

Supongamos que tenemos una petición $Conoce(Juan, x)$: ¿A quién conoce Juan?

p	q	θ
$Conoce(Juan, x)$	$Conoce(Juan, Juana)$	{x/Juana}
$Conoce(Juan, x)$	$Conoce(y, OJ)$	{x/OJ,y/Juan}
$Conoce(Juan, x)$	$Conoce(y, Madre(y))$	{y/Juan,x/Madre(Juan)}
$Conoce(Juan, x)$	$Conoce(x, OJ)$	fallo

Unificación

p	q	θ
$Conoce(Juan, x)$	$Conoce(Juan, Juana)$	{x/Juana}
$Conoce(Juan, x)$	$Conoce(y, OJ)$	{x/OJ,y/Juan}
$Conoce(Juan, x)$	$Conoce(y, Madre(y))$	{y/Juan,x/Madre(Juan)}
$Conoce(Juan, x)$	$Conoce(x, OJ)$	fallo

La última sentencia falla porque x no puede tomar los valores $Juan$ y OJ al mismo tiempo.
El problema se presenta sólo cuando las dos sentencias tienen que utilizar el mismo nombre de variable, x. Este problema se puede evitar utilizando la estandarización de las variables de las sentencias que van a ser unificadas, que consiste en renombrer sus variables para evitar conflictos de nombre. Por ejemplo:

$$Conoce(z_{17}, OJ)$$

Ahora la unificación tendrá éxito.

$$\textit{UNIFICA}(Conoce(Juan, x), Conoce(Z_{17}, OJ)) = \{x/OJ, z_{17}/Juan\}$$

Modus Ponens Generalizado (MPG)

$$\frac{p_1{}',\ \ p_2{}',\ \ \ldots,\ \ p_n{}',\ \ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$

donde $p_i{}'\theta = p_i\theta$ para todo i.

p_1' *es Rey*(*Juan*)	p_1 *es Rey*(x)
p_2 *es* Codicioso(y)	p_2 *es* Codicioso(x)
θ *es* $\{x/Juan, y/Juan\}$	q *es Malvado*(x)
$q\theta$ *es* Malvado(Juan)	

Solidez del MPG

MPG es una regla de inferencia sólida. Asumimos que todas las variables están universalmente especificadas.
Necesitamos mostrar que

$$p_1', \ \ldots, \ p_n', \ \ (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models q\theta$$

mientras que $p_i'\theta = p_i\theta$ para todo i.

Lema: Para cualquier cláusula p, tenemos $p \models p\theta$ por especificación universal.

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta)$
2. $p_1', \ \ldots, \ p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\theta \wedge \ldots \wedge p_n'\theta$
3. De 1 y 2, $q\theta$ se sigue del Modus Ponens ordinario.

Ejemplo de base de conocimiento

La ley dice que para un Americano es un crimen vender armas a naciones hostiles. Cierto país, enemigo de América, tiene algunos misiles y todos sus misiles fueron vendidos por un Coronel americano.

Probar que el coronel es un criminal.

Ejemplo de base de conocimiento

... para un americano es un crimen vender armas a naciones hostiles: $Americano(x) \wedge Arma(y) \wedge Vende(x, y, z) \wedge Hostil(z) \Rightarrow Criminal(x)$

País ... tiene algunos misiles, es decir, $\exists\, x\ Tiene($*País*$,x) \wedge Misil(x)$:

$Tiene($*País*$,M_1)$ y $Misil(M_1)$

... todos los misiles del país le fueron vendidos por el coronel.

$\forall x Misil(x) \wedge Tiene($*País*$,x) \Rightarrow Vende(Coronel, x,$*País*$)$

Los misiles son armas:

$Misil(x) \Rightarrow Arma(x)$

Ejemplo de base de conocimiento

Un enemigo de américa cuenta como hostil:

$Enemigo(x,$*América*$) \Rightarrow Hostil(x)$

El Coronel, que es americano, . . .

$Americano(Coronel)$

El País, un enemigo de América . . .

$Enemigo($*País*,*América*$)$

Algoritmo sencillo de encadenamiento hacia delante

función PREGUNTA-EHD-LPO (BC,α) devuelve una sustitución o *falso*
entradas: BC, un conjunto de cláusulas positivas de primer orden
α, la petición, una sentencia atómica
variables locales: *nuevas*, las nuevas sentencias inferidas en cada iteración
repetir hasta *nuevo* está vacío
nuevo $\leftarrow \{\}$
para cada sentencia *r* en la BC, hacer
$(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ ESTANDARIZAR-VAR(r)
para cada θ tal que SUST$(\theta, p_1 \wedge \ldots \wedge p_n) = SUST(p'_1 \wedge \ldots \wedge p'_n)$
para algún $p'_1, \ldots, p'_n$ en BC
$q' \leftarrow SUST(\theta, q)$
si q' no es el renombramiento de una sentencia de BC o de *nuevo*,
entonces hacer
añadir q' a *nuevo*
$\phi \leftarrow UNIFICA(q', \alpha)$
si ϕ no es *fallo* entonces devolver ϕ
añadir *nuevo* a BC
devolver *falso*

Algoritmo sencillo de encadenamiento hacia delante

Este algoritmo de encadenamiento hacia delante es conceptualmente sencillo, pero muy ineficiente.
Comenzando con los hechos conocidos, el proceso dispara todas las reglas cuyas premisas se satisfacen, añadiendo sus conclusiones al conjunto de hechos conocidos.
El proceso se repite hasta que la petición es respondida o no se pueden añadir más hechos.

Ejemplo de EHD

Usaremos nuestro problema sobre el crimen para ilustrar cómo funciona el algoritmo PREGUNTA-EHD-LPO.
Las sentencias de implicación son:

- $Americano(x) \wedge Arma(y) \wedge Vende(x, y, z) \wedge Hostil(z) \Rightarrow Criminal(x)$
- $\forall x Misil(x) \wedge Tiene(\textsf{País}, x) \Rightarrow Vende(Coronel, x, \textsf{País})$
- $Misil(x) \Rightarrow Arma(x)$
- $Enemigo(x, \textsf{América}) \Rightarrow Hostil(x)$

Ejemplo de EHD

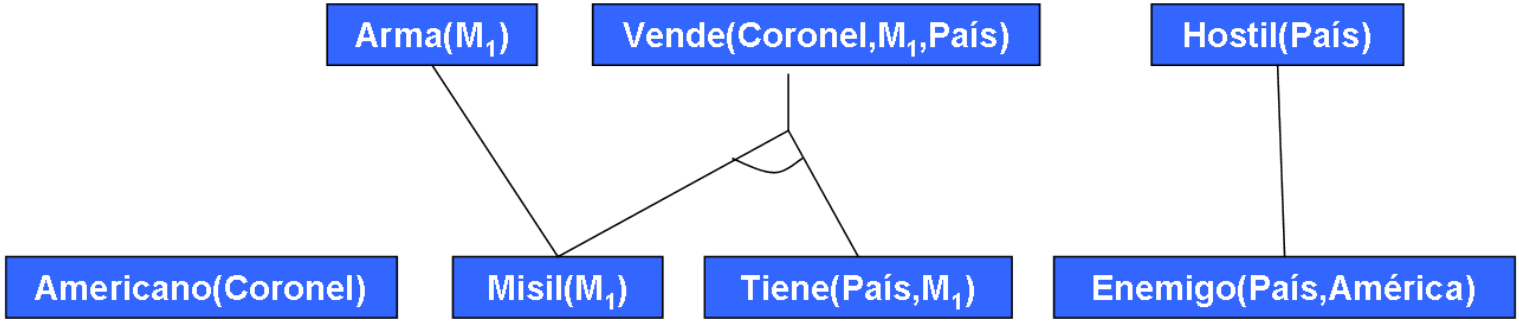
Americano(Coronel)

Misil(M_1)

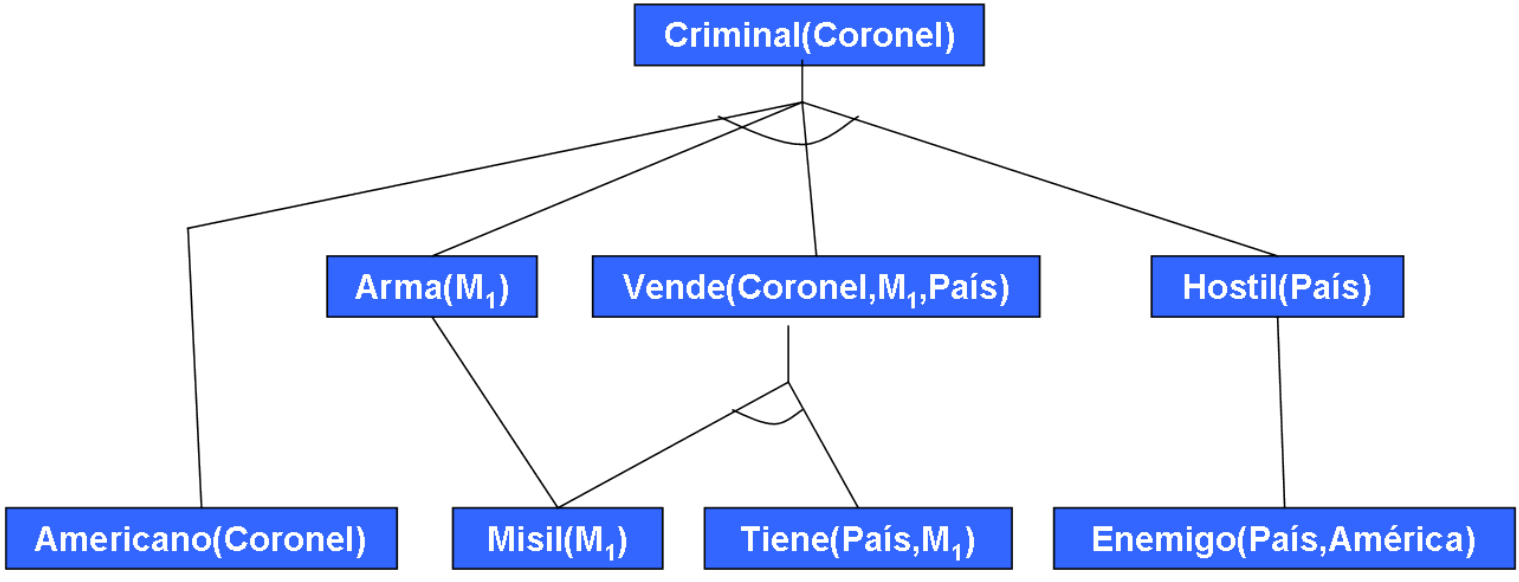
Tiene(País,M_1)

Enemigo(País,América)

Ejemplo de EHD



Ejemplo de EHD



Propiedades del Encadenamiento Hacia Delante

- El algortimo PREGUNTA-EHD-LPO es **sólido**, porque cada inferencia es justo una aplicación del Mo dus Ponens Generalizado, que es sólido.
- El algortimo PREGUNTA-EHD-LPO es **completo** para las BC de cláusulas positivas.
- En las BC Datalog, que no contienen símbolos de función, la demostración de su completitud es bastante fácil. Sea k la aridad máxima (número de argumentos) de cada predicado, p el número de predicados y n el número de símbolos de constante. No puede haber más de $p \cdot n^k$ hechos base, así que después de estas muchas iteraciones el algoritmo debe encontrar un punto fijo.
- Con las cláusulas positivas que contienen símbolos de función, PREGUNTA-EHD-LPO puede generar muchos hechos nuevos infinitamente, por lo que necesitamos tener mucho cuidado.

Eficiencia del encadenamiento hacia delante

En el algoritmo simple de encadenamiento hacia delante presentado hay tres **fuentes de complejidad** posibles:

1. El bucle interno del algortimo requiere que se encuentren todos los unificadores posibles de manera que la premisa se unifique con un conjunto adecuado de hechos de la BC (**emparejamiento de patrones**).
2. El algoritmo vuelve a comprobar cada regla en cada iteración para ver si se satisfacen sus premisas (EHD incremental).
3. El algoritmo podría generar muchos hechos que son irrelevantes para el algoritmo (hechos irrelevantes).

Vamos a abordar cada uno de estos problemas por separado.

Emparejar reglas con los hechos conocidos

Problema de ordenación de los conjuntores.

$$Misil(x) \wedge Tiene(Pais, x) \Rightarrow Vende(Coronel, x, Pais)$$

Encontramos todos los objetos que tiene el País en tiempo constante; y comprobar para cada objeto si es misil.

Muchos objetos en poseción del País y unos pocos misiles $\Rightarrow$ mejor encontrar los misiles primero y luego comprobar si son del País.

Encontrar la **ordenación de los conjuntores** de las premisas de una regla tal que se minimice el coste es un **problema NP-duro**. $\Rightarrow$ Usar **heurísticas**, por ej. la de la variable más restringida que estudiamos en los PSRs.

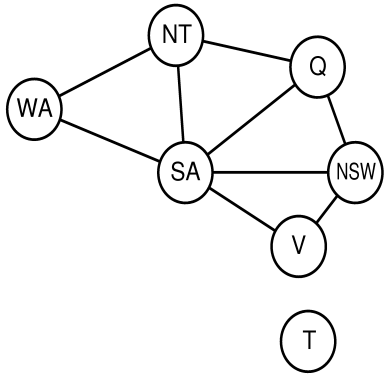
Emparejar reglas con los hechos conocidos

Emparejamiento de patrones $\leftrightarrow$ Satisfacción de restricciones

Ver cada conjuntor como una restricción sobre las variables que contiene.

Podemos expresar cada PSR de dominio finito como una única cláusula positiva junto a algunos hechos base asociados a ella.

Emparejar reglas con los hechos conocidos



$$Dif(wa, nt) \wedge Dif(wa, sa) \wedge$$
$$Dif(nt, q) Dif(nt, sa) \wedge$$
$$Dif(q, nsw) \wedge Dif(q, sa) \wedge$$
$$Dif(nsw, v) \wedge Dif(nsw, sa) \wedge$$
$$Dif(v, sa) \Rightarrow Colorable()$$

$$Dif(Rojo, Azul) \quad Dif(Rojo, Verde)$$
$$Dif(Verde, Rojo) \quad Dif(Verde, Azul)$$
$$Dif(Azul, Rojo) \quad Dif(Azul, Verde)$$

$Colorable()$ se infiere si y sólo si el Problema de Satisfacción de Resctricciones (PSR) tiene solución.

Los PSRs incluyen al 3SAT como un caso especial, por lo que podemos concluir que emparejar una cláusula positiva con un conjunto de hechos es un problema NP-duro.

Emparejar reglas con los hechos conocidos

Conclusión: el algoritmo de encadenamiento hacia delante visto tiene un problema de emparejamiento NP-duro en su bucle interno.

Sin embargo:

- La complejidad de los datos del EHD es polinómica.
- Considerar subclases de reglas para las que el encadenamiento sea eficiente.
- Eliminar los intentos de emparejamiento de reglas redundantes.

Encadenamiento hacia delante incremental

Evitar el emparejamiento de reglas redundantes.

Cada hecho nuevo inferido en una iteración t debe ser derivado de al menos un hecho nuevo inferido en la iteración $t-1$.

Esta observación nos lleva a un algoritmo de EHD incremental, en el que en cada iteración t sólo si su premisa incluye algún valor conjuntor p_i que se unifique con un hecho p_i' que se haya inferido en la iteración $t-1$.

Hechos irrelevantes

El EHD realiza todas las inferencias permitidas basadas en los hechos conocidos, aunque éstos sean irrelevantes respecto al objetivo.

Usar el Encademanciento Hacia Atrás (EHA), que veremos en la próxima lección.