Extracción Automática de Conocimiento en Bases de Datos e Ingeniería del Software

T.1b INTRODUCCIÓN (Inducción Declarativa y de Árboles de Decisión)

José Hernández Orallo

Programa: Programación Declarativa e Ingeniería de la Programación

Temario

1.1. El Problema de la Extracción Automática de Conocimiento.
1.2. Técnicas de Aprendizaje Automático.
1.3. Evaluación de Hipótesis
1.4. Programación Lógica Inductiva (ILP).
1.5. Programación Lógico-Funcional Inductiva (IFLP).
1.6. Árboles de Decisión.

ILP

Programación Lógica Inductiva: “Intersección entre el aprendizaje automático y la programación lógica” (*S. Muggleton).*

De una manera más precisa:

ILP: aprendizaje de teorías lógicas de primer orden

Se representa en Lógica de primer orden:

- Hipótesis: H,
- Evidencias positiva y negativa: E^+ y E^-
- Conocimiento previo B.

REPRESENTACIÓN UNIFORME

ILP

Especificación del aprendizaje supervisado en ILP (también mal llamado explicativo). Los ejemplos deben ser implicados por la teoría.

- La especificación del problema es muy clara:
  - Satisfacibilidad *a priori*: $\mathbf{B \wedge E^- \not\models False}$
  - Necesidad *a priori*: $\mathbf{B \not\models E^+}$
  - Satisfacibilidad *a posteriori o consistencia*: $\mathbf{B \wedge H \wedge E^- \not\models False}$
  - Suficiencia *a posteriori o completitud*: $\mathbf{B \wedge H \models E^+}$
- La hipótesis a aprender (si sólo un predicado) está entre:

  $H_{\text{más general}} = \top = p(x_1,\ldots,x_n)$ y $H_{\text{más específica}} = \bot = E^+$

- Hay que buscar y *elegir* una H que cumpla las condiciones anteriores.

ILP

ILP: especificación del aprendizaje no supervisado (también llamado descriptivo o a veces mal llamado no-monótono). Los ejemplos son *interpretaciones / modelos*.

- La especificación del problema es también muy clara:
  - Satisfacibilidad *a priori*: $\mathbf{B \wedge E^- \not\models False}$
  - Necesidad *a priori*: $\mathbf{B \not\models E^+}$**.**
  - Satisfacibilidad *a posteriori o consistencia*: $\mathbf{B \wedge H \wedge E^- \not\models False}$
  - $\mathbf{E^+}$ es un modelo para $\mathbf{B \wedge H}$**.**

    Es decir, B ^ H son ciertos para cada ejemplo positivo.

Los criterios de selección son más importantes aquí, y se suelen hacer búsquedas top-down a partir de hipótesis demasiado generales (p.ej. $p(X_1, X_2, ...)$)

ILP

ILP: Ejemplo supervisado:

B= { parent(eve, sue). parent(ann, tom). parent(pat,ann). parent(tom,sue).
female(ann). female(sue). female(eve). }

E^+= { daughter(sue,eve). daughter(ann,pat). }

E^-= { daughter(tom,ann). daughter(eve,ann). }

- Satisfacibilidad *a priori*: $\mathbf{B \wedge E^- \not\models False}$
- Necesidad *a priori*: $\mathbf{B \not\models E^+}$

H = { daughter(X,Y) :- female(X), parent(Y,X). }

- Satisfacibilidad *a posteriori o consistencia*: $\mathbf{B \wedge H \wedge E^- \not\models False}$
- Suficiencia *a posteriori o completitud*: $\mathbf{B \wedge H \models E^+}$

ILP

ILP: Ejemplo no supervisado:

B= { duck(lucas). ¬(black(X) ^ white(X)). }

E^+= { black(tweety). raven(tweety). black(lucas). swan(sue).
black(bobby). raven(bobby). black(blacky). raven(blacky). }

- Satisfacibilidad *a priori*: $\mathbf{B \wedge E^- \not\models False}$
- Necesidad *a priori*: $\mathbf{B \not\models E^+}$

H = { black(X) :- raven(X). }

- Satisfacibilidad *a posteriori o consistencia*: $\mathbf{B \wedge H \wedge E^- \not\models False}$

No cubre la evidencia, pero la evidencia es un modelo para $\mathbf{B \wedge H}$**.**

Historia de ILP

- Prehistoria (1970)
  - Plotkin - LGG, RLGG -1970, 1971
- Entusiasmo Inicial (1975-1983)
  - Vere 1975, 1977; Shapiro 1981, 1983.
- Decadencia (1983-1987)
- Renacimiento (1987- ...)
  - MARVIN - Sammut & Banerji 1986, DUCE - Muggleton 1987, Helft 1987, QuMas - Mozetic 1987, LINUS - Lavrac et al. 1991
  - CIGOL - Muggleton & Buntine 1988, ML-SMART - Bergadano et al. 1988, CLINT - De Raedt & Bruynooghe 1988, BLIP, MOBAL - Morik, Wrobel et al. 1988, 1989.
  - GOLEM - Muggleton & Feng 1990
  - FOIL - Quinlan 1990, mFOIL - Dzeroski 1991, FOCL - Brunk & Pazzani 1991, FFOIL - Quinlan 1996.
  - CLAUDIEN - De Raedt & Bruynooghe 1993.
  - PROGOL - Muggleton 1995, FORS - Karalic & Bratko 1996, TILDE - Blockeel & De Raedt 1997
  - ...

IFP: (LISP) Summers 1977;
Kodratoff and Jouanaud 1979;

ILP. Restricciones de Representación

Representación ILP ≠ LÓGICA DE PRIMER ORDEN

La mayoría basados en subconjuntos de la lógica de primer orden:

- La teoría de conocimiento previo *B* se suele restringir:
  - programas normales (sólo un átomo en cabeza).
  - cláusulas de Horn (no negación en cuerpo).
  - programas definidos (no negación en cuerpo y cabeza no vacía).
  - sólo literales ground.
- La evidencia *E* se restringe generalmente a:
  - sólo literales ground o incluso átomos ground.
  - incluso a sólo un predicado al mismo tiempo.
- La hipótesis *H* tiene diversas restricciones:
  - casi siempre programas definidos (evitan hipótesis negativas).
  - sólo un predicado al mismo tiempo.
  - muchos son no recursivos.

ILP. Restricciones de Representación

Además suele haber otras restricciones:

- ij-determinaciones entre los atributos del predicado.
- límite del número de atributos por predicado.
- límite en el uso de variables.
- restricción en el uso de funciones (aplanamiento a DATALOG).

De los sistemas actuales (sin uso de esquemas), el más expresivo es PROGOL, que soporta:

- *B = programas normales,*
- *E = cláusulas de Horn,*
- *H = programas definidos con recursión.*

ILP. Espacio de Búsqueda.

Dada una evidencia *E* y *B*, se definen

Hipótesis más específicas: Una hipótesis más específica *H* es aquella que cumple $\mathbf{B} \wedge \mathbf{H} \models \mathbf{E}^+$ y además no existe otra *H'* estrictamente más específica (*H* $\models$ *H'* pero no *H'* $\not\models$ *H*) que también cumple $\mathbf{B} \wedge \mathbf{H'} \models \mathbf{E}^+$.

Si sólo existe una hipótesis más específica, se denota por $\bot$**.**

Generalmente, la hipótesis más específica es *H* = *E*.

Hipótesis más generales: Una hipótesis más general *H* es aquella que cumple $\mathbf{B} \wedge \mathbf{H} \models \mathbf{E}^+$ y además no existe otra *H'* estrictamente más general (*H'* $\models$ *H* pero no *H* $\not\models$ *H'*) que también cumple $\mathbf{B} \wedge \mathbf{H'} \models \mathbf{E}^+$.

Si sólo existe una hipótesis más general, se denota por $\top$**.**

Generalmente, la hipótesis más general si sólo hay un predicado *p* en la evidencia es $H = p(X_1, X_2, X_3, ...)$ y no hay evidencia negativa.

ILP. Métodos

¿Cómo buscar en el retículo* entre $\top$ y $\bot$?

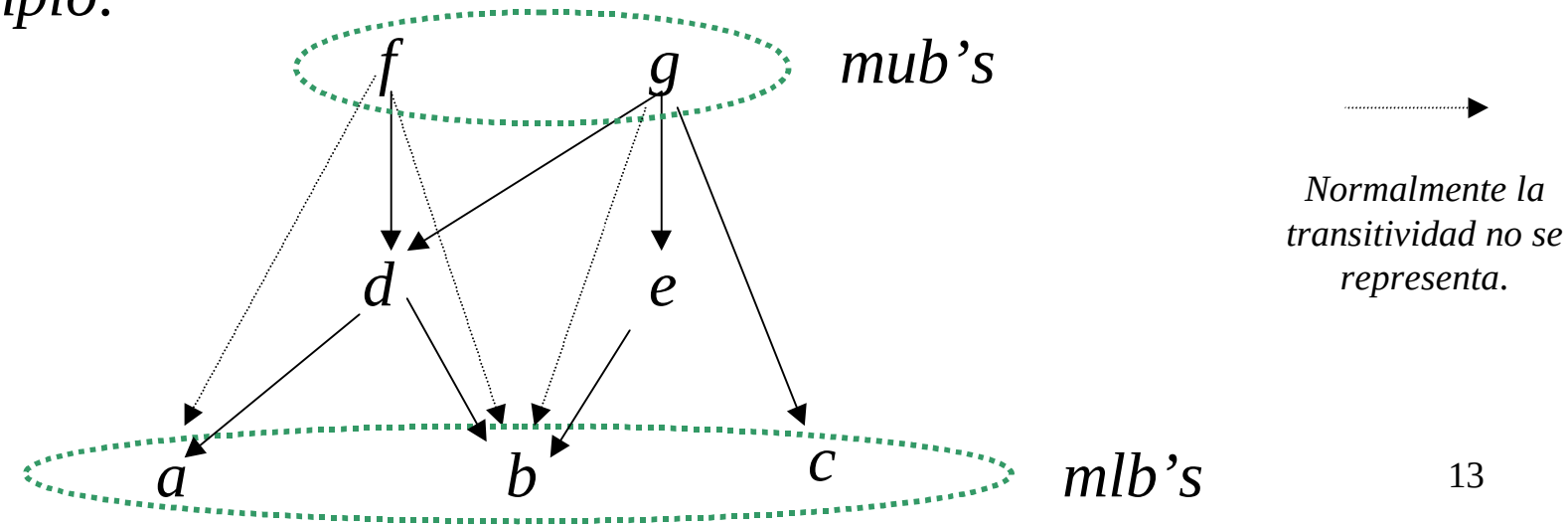
Métodos más importantes:

- Bottom-up o basados en generalización: métodos más clásicos, inversión de operadores deductivos.
- Top-down o basados en especialización: se añaden condiciones, pick-and-mix, árboles de decisión.
- Bidireccionales: se utilizan los dos tipos de operadores de generalización y especialización hasta que se acota el espacio.
- Ni bottom-up ni top-down. P.ej. genéticos.
- Guiados por la longitud de la hipótesis.
- Guiados por esquemas, por background knowledge...

** Será un retículo si no hay recursividad (entre otras cosas).*

ILP. Métodos

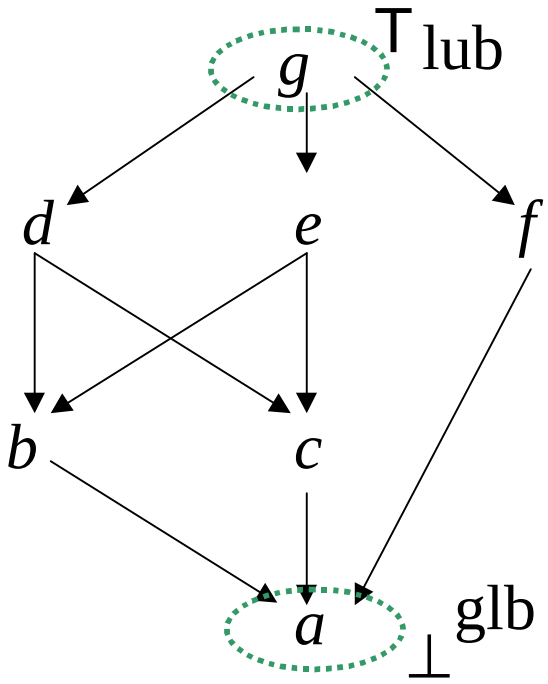
En ILP se suele trabajar con operadores de generalización / especialización que construyan cuasi-órdenes (ref,trans) u órdenes parciales (ref,trans,anti-t) (utilizando clases de equivalencia) entre átomos, cláusulas o programas.
Si no tienen lub o glb no forman retículo (solamente un grafo orientado no cíclico transitivo). Tienen varios mub o mlb.
Ejemplo:



ILP. Métodos

Generalmente, si existe lub o glb se denotan por ⊤ y ⊥ y tenemos un retículo (lattice).

Ejemplo:



ILP. Métodos Bottom-Up

De más específico (bottom) a más general. Parten de los ejemplos y empiezan a generalizar sin invadir el terreno de los ejemplos negativos.

- Operadores de generalización:
  - Operadores clásicos (inspirados en ML):
    - eliminación de condiciones.
    - transformación de constantes en variables.
    - transformación de conjunciones en disyunciones.
  - Inversión de reglas deductivas:
    - antiunificación.
    - resolución inversa (operadores V y W).
    - implicación inversa.

ILP. Métodos Bottom-Up

¿Cuándo paramos de generalizar?

Si no tenemos ejemplos negativos...

- Generalizaremos hasta llegar a T. Esfuerzo inútil.
- Ejemplo:

  E^+= { tiene_alas(tweety):-pajaro(tweety). tiene_alas(birdy):-pajaro(birdy) }.

  H = { tiene_alas(X) }.

Incluso si tenemos ejemplos negativos...

- Tenderemos a sobregeneralizar (underfit) y llegar a la hipótesis consistente más general.
- Ejemplo:

  B= { parent(eve, sue). parent(ann, tom). parent(pat,ann).
  parent(tom,sue). female(ann). female(sue). female(eve). }

  E^+= { daughter(sue,eve). daughter(ann,pat). }

  E^-= { daughter(tom,ann). daughter(eve,ann). }

  H = { daughter(X,Y) :- Y <> ann. }

Generalización Menos General

Definición: Dadas dos cláusulas* C y D, C θ-subsume a D, $C =\prec D$, si y sólo si existe una sustitución θ tal que $C\theta \subseteq D$.

Propiedades:

- $C \equiv D$ si y sólo si $C =\prec D$ y , $D =\prec C$
- [C] denota el conjunto de cláusulas equivalentes a C
- $[C] =\prec [D]$ si y sólo si $C =\prec D$
- $=\prec$ es un orden parcial sobre la clase de equvlncia. de las cláusulas
- **SUBSUNCIÓN e IMPLICACIÓN**
  - ♦ Teorema: $C =\prec D$ entonces $C \rightarrow D$

pero $C \rightarrow D$ no implica $C =\prec D$

p.ej. $C = nat(s(x)) \leftarrow nat(x)$

$D = nat(s(s(x))) \leftarrow nat(x)$

* *por definición sin negación en cuerpo o en cabeza. Una cláusula es un conjunto de literales que se interpretan como una disyunción.* $C\theta \subseteq D$ es lo mismo que $\forall(C\theta \xrightarrow{17} D)$

Generalización Menos General

Definición: C es la **generalización menos general** (lgg) de D bajo θ-subsunción si C =≺ D y para cada E tal que E =≺ D se cumple E =≺ C. (es el lub de la subsunción)

EJEMPLO 1

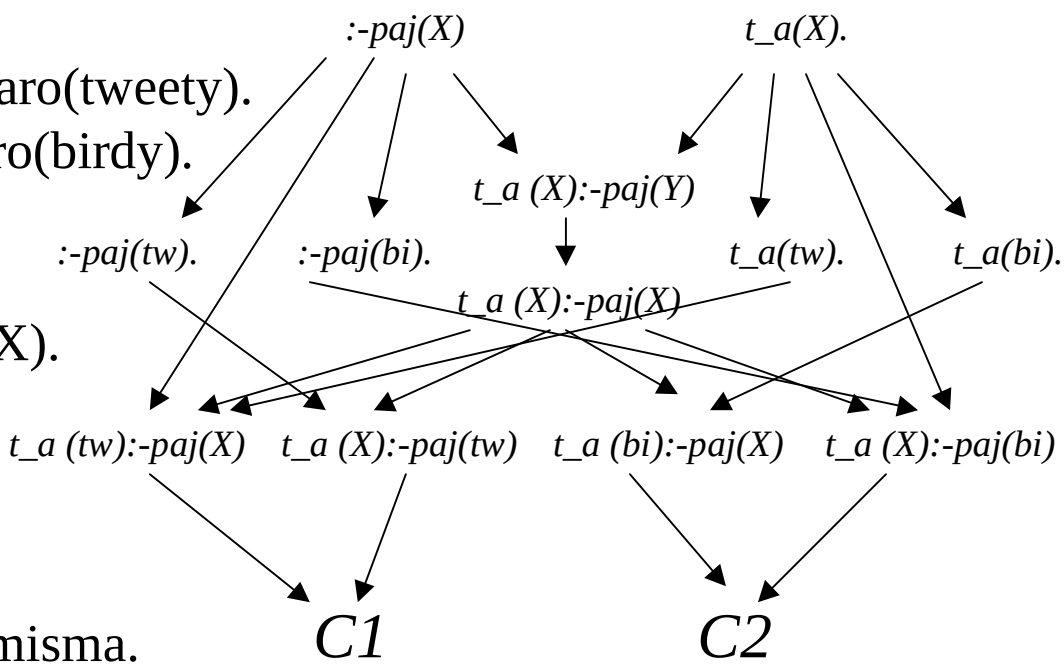
C1: tiene_alas(tweety):-pájaro(tweety).

C2: tiene_alas(birdy):-pájaro(birdy).

Generalizaciones:

C3: tiene_alas(X).

C4: tiene_alas(X):- pájaro(X).



El lgg de una cláusula es ella misma.

En general, si exceptuamos ella misma no existe otra cláusula que cumpla la propiedad anterior, por lo que para una cláusula es inútil.

Generalización Menos General

Definición: G es la **generalización menos general** (lgg) de C y D bajo θ-subsunción si G =≺ C, G =≺ D y para cada E tal que E =≺ C, E =≺ D se cumple E =≺ G. (Plotkin 1971a, 1971b)

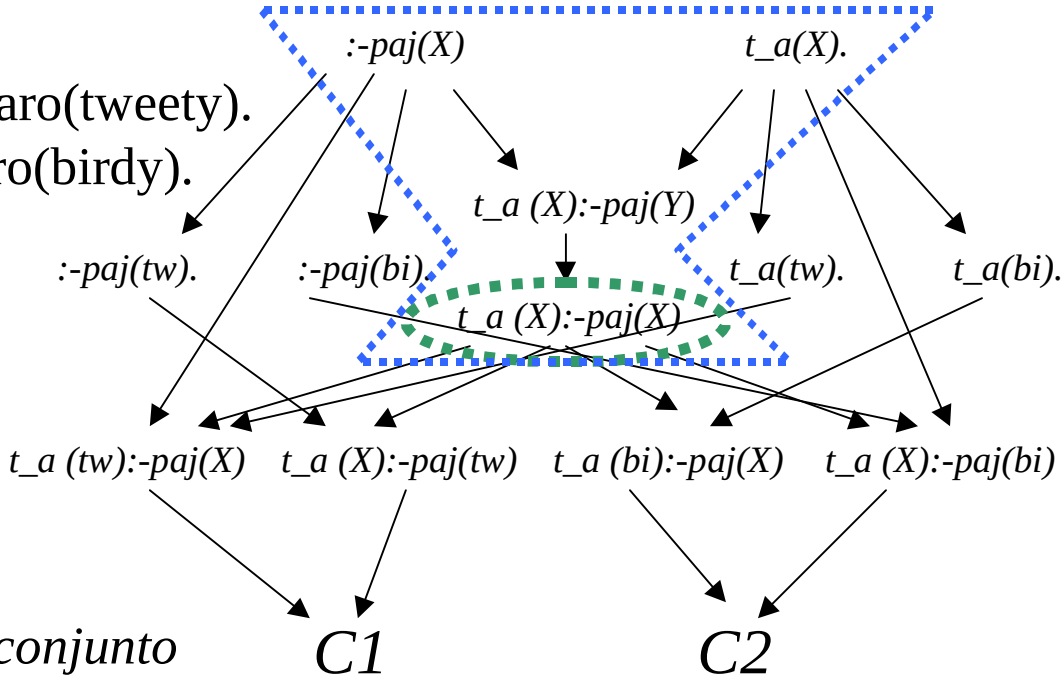
EJEMPLO 1

C1: tiene_alas(tweety):-pájaro(tweety).

C2: tiene_alas(birdy):-pájaro(birdy).

lgg(C1, C2) =

{ tiene_alas(X):- pájaro(X) }



Plotkin demostró que cualquier conjunto finito de cláusulas tiene lgg.

Generalización Menos General

Computación de la **generalización menos general** (lgg):

- Sean C y D dos cláusulas. El lgg de C y D en el orden de subsunción se calcula como:

  $lgg(C,D) = \{lgg(l,m) \mid l \in C \text{ y } m \in D \text{ y } l \text{ y } m \text{ son compatibles}\}$

  entendiendo por compatible literales con el mismo signo y el mismo símbolo de predicado.

- Posteriormente se eliminan literales redundantes:

  $l \in C$ es redundante bajo θ-subsunción si $C - \{l\} \equiv_\theta C$.

El lgg de dos átomos compatibles es el dual de la unificación:

EJEMPLO:

l = member(3,cons(2,cons(3,nil)))

m = member(3,cons(4,cons(5,cons(6,nil))))

anti-unif. = member(3,cons($v_{2,4}$,cons($v_{3,5}$,$v_{nil,cons(6,nil)}$)))

lgg(l,m) = member(3,cons(X,cons(Y,Z)))

Generalización Menos General Relativa

La generalización menos general no tiene en cuenta *B* y puede llegar a generalizar demasiado:

EJEMPLO:

C: uncle(X,Y):- brother(X,father(Y)).

D: uncle(X,Y):- brother(X,mother(Y)).

lgg(C,D) = uncle(X,Y):- brother(X,Z).

Generalización Menos General Relativa

- Una cláusula C subsume otra cláusula D con respecto a una teoría B, denotado por $C =\prec_B D$, si $B \models \forall(C\theta \rightarrow D)$.

Definición: G es la **generalización menos general relativa** ($rlgg_B$) de C y D bajo θ-subsunción respecto a una teoría *B* si $G =\prec_B C$, $G =\prec_B D$ y para cada E tal que $E =\prec_B C$, $E =\prec_B D$ se cumple $E =\prec_B G$. (Plotkin 1971a, 1971b)

EJEMPLO:

B: parent(father(X),X).
parent(mother(X),X).
C: uncle(X,Y):- brother(X,father(Y)).
D: uncle(X,Y):- brother(X,mother(Y)).

$rlgg_B$(C,D) = uncle(X,Y):-brother(X,U),parent(U,Y).

Generalización Menos General Relativa

- Problema: no siempre existe rlgg de un conjunto de cláusulas con respecto a una teoría.

  C: q(f(a)).
  D: q(g(a)).
  B: p(f(X),Y).
  p(g(X),Y).

  G : q(X):-p(X,g_1(X)),…,p(X,g_n(X)) es una generalización

- Caso particular: rlgg existe <u>siempre</u> si *B* es básico.
- Esto es en lo que se basa el GOLEM (Muggleton and Feng 1992).

Subsunción Generalizada

rlgg puede conducir a conclusiones contraintuitivas...

EJEMPLO:

C: small(X):-cat(X).
D: cuddlypet(X):- fluffy(X), cat(X).

B: pet(X):- cat(X).
cuddlypet(X):- small(X), fluffy(X), pet(X).

Tenemos: **C** $=\prec_B$ **D** **porque** B $\models \forall(C\theta \rightarrow D)$.

Si asumimos que una cláusula C es más general que otra D si cualquier interpretación de C permite obtener las mismas conclusiones que con D.

En el ejemplo anterior C no sería más general que D.

Subsunción Generalizada

Definición: *Subsunción generalizada (cláusulas definidas)* (Buntime 1986, 1988)

Una cláusula C es más general de Buntime que otra D respecto a un programa B, denotado por $C \leq_B D$, si para cualquier interpretación de Herbrand I modelo de B, entonces $T_D(I) \subseteq T_C(I)$.

En el ejemplo anterior C y D no pueden compararse ya que tienen distintas cabezas.

Subsunción Generalizada

Definición: G es la **generalización menos general relativa de Buntime** (rlggb) de C y D respecto a una teoría *B* si $G \leq_B C$, $G \leq_B D$ y para cada E tal que $E \leq_B C$, $E \leq_B D$ se cumple $E \leq_B G$.

EJEMPLO:

B: parent(father(X),X).
parent(mother(X),X).
C: uncle(X,Y):- brother(X,father(Y)).
D: uncle(X,Y):- brother(X,mother(Y)).

rlggb(C1,C2) = uncle(X,Y):-brother(X,U),parent(U,Y).

Subsunción Generalizada

Visión operacional

Sean C y D cláusulas con variables disjuntas y sea B un programa lógico. Sea θ una sustitución que asigna a las variables de D nuevas constantes (que no aparecen en C, D ni B).

Entonces,

$C \leq_B D$ sii existe una sustitución σ tal que:

$C_{head}\,\sigma = D_{head} \quad \wedge \quad B \cup D_{body} \models \exists\,(C_{body}\,\sigma\theta)$

Subsunción Generalizada

- Subsunción generalizada versus generalización de Plotkin.

$$C \leq_{\varnothing} D \text{ sii } C =\prec D$$

- Subsunción generalizada versus generalización menos general relativa.

$C \leq_B D$ sii C aparece en la raíz de una refutación en la demostración de $B \models (C \rightarrow D)$

$\leq_B$ un caso especial de rlgg

Cuasi-orden de Implicación

Definición: *Más General respecto Implicación Relativa*

Una cláusula C es más general (respecto implicación) que otra D respecto a un programa B, denotado por $C \models_B D$, sii, $C \cup B \models D$.

No existe rlgg en general para un cjto. de cláusulas respecto a implicación relativa.

Si $B = \emptyset$ entonces es implicación no relativa.

Es un problema abierto el saber si existe en general un lgg para todo cjto. de cláusulas respecto a implicación no relativa.

Comparación de cuasi-órdenes entre cláusulas

Existencia de generalizaciones menores (lub)

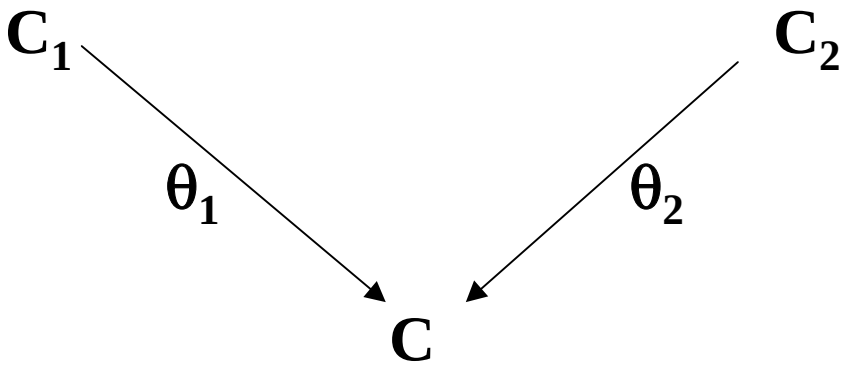
	Cláusulas de Horn	*Cláusulas Generales*
Subsunción ($=\prec$)	*Sí*	*Sí*
Implicación ($\mid=$)	*No*	*? (Th 15.17)*
Subsunción Relativa ($=\prec_B$)	*No (Th 16.9)*	*No (Th 16.8)*
Implicación Relativa ($\mid=_B$)	*No*	*No (Th 16.14)*
Subsunción Generalizada ($\leq_B$)	*No (Th 16.29)*	*no definido*

Los (Th X.Y) se refieren a teoremas sobre resultados afirmativos para casos particulares. Ver (Nienhuys-Cheng & de Wolf 1997).

La implicación es mejor que la subsunción, porque la primera maneja mejor las cláusulas recursivas. Sin embargo, la primera es indecidible en general.

Resolución Inversa

Muggleton y Buntine describen dos operadores en los que se basan los pasos de resolución inversa: los operadores V y W.



RESOLUCIÓN: $C = C_1 \bullet C_2$

$\neg l_1 \in C_1 \wedge l_2 \in C_2 \mid C_1$ y C_2 con variables disjuntas

$\wedge\ \theta = mgu(l_1, l_2)$ t.q. $l_1\theta = l_2\theta$

$\theta = \theta_1 \theta_2 \wedge\ l_1\theta_1 = l_2\theta_2$

$C=(C_1 - \neg l_1)\theta_1 \cup (C_2 - l_2)\theta_2$

Resolución Inversa

Inversión de la resolución: obtención de C_1 (o C_2) a partir de C y C_2 (C_1).

No hay una solución única

<u>EJEMPLO</u>:

$C_1 = A \leftarrow B,C,D$

$C_2 = B \leftarrow E,F$

$C = A \leftarrow E,F,C,D$

Dados C y C_2, C_1 es una solución válida, pero también lo son:

$C_1' = A \leftarrow B,C,D,E$

$C_1'' = A \leftarrow B,C,D,F$

$C_1''' = A \leftarrow B,C,D,E,F$

Resolución Inversa

El problema es aún más agudo para cláusulas de primer orden

EJEMPLO:

C_1 = ← más_pesado(martillo,pluma)

C = ← más_denso(martillo,pluma), más_grande(martillo,pluma)

Hay muchas posibles soluciones C2 desde la más específica

C_2 = más_pesado(martillo,pluma) ← más_denso(martillo,pluma),
más_grande(martillo,pluma)

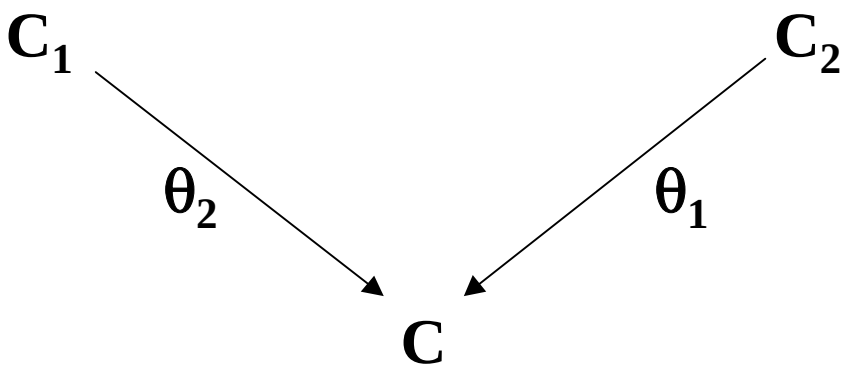
a la más general

C_2 = más_pesado(A,B) ← más_denso(A,B), más_grande(A,B)

Para generar C_2 debemos decidir qué términos se convierten en variables y cómo.

Resolución Inversa: Operador V

OPERADOR V: Produce C_2 a partir de C y C_1



dadas dos cláusulas C_1 y C, el V-operador encuentra C_2 tal que C es una instancia de un resolvente de C_1 y C_2.

Generaliza $\{C_1, C\}$ a $\{C_1, C_2\}$

Resolución Inversa: Operador V

Operador V: Primer Caso. Absorción

el cuerpo de C_1 es absorbido en el cuerpo de C (después de una unificación) y reemplazado por su cabeza.

(*es un unfolding o desplegamiento*)

EJEMPLO:

C:pájaro(tweety):-**tiene_plumas(tweety),**
tiene_alas(tweety),tiene_pico(tweety).

C_1:**vuela(X)**:- **tiene_plumas(X),tiene_alas(X)**.

θ = { X/tweety }

C_2:pájaro(tweety):-**vuela(tweety)**, tiene_pico(tweety).

Resolución Inversa: Operador V

Absorción. <u>EJEMPLO LARGO</u>:

Supongamos que tenemos un lenguaje con tres predicados (nadar, animal y pez) y que estamos aprendiendo a partir de ejemplos.

Nuestra teoría es un programa lógico *P* que de momento consta de las siguientes reglas:

animal(tiburón)

nadar(tiburón)

Obtenemos un nuevo ejemplo **pez(tiburón)** no implicado por P.

- Debemos ajustar la teoría para implicar este hecho.
  - Deberemos añadir las cláusulas necesarias para que **pez(tiburón)** pueda deducirse por resolución SLD.

Resolución Inversa: Operador V

Absorción. <u>EJEMPLO LARGO (cont.)</u>:
Debemos encontrar dos cláusulas que tengan un resolvente del que pez(tiburon) es una instancia

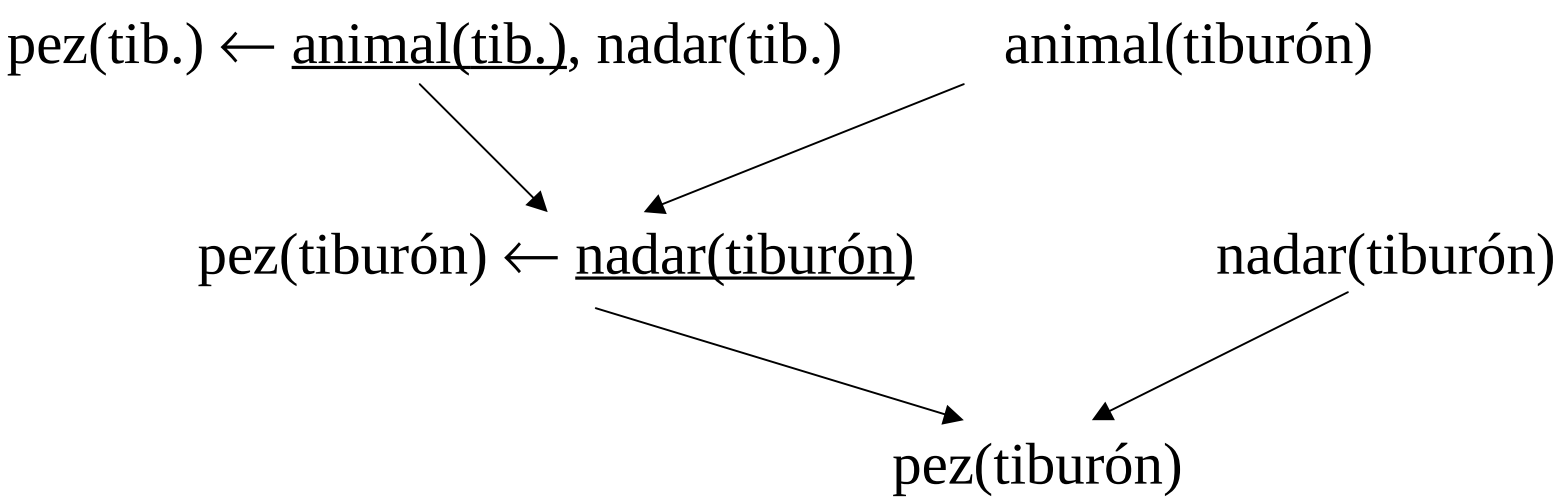
• Cláusula de entrada: una cláusula de P (nadar(tiburón))
• Cláusula central: podemos optar por varias. Por ejemplo, la menos general (pez(tiburón) ← nadar(tiburón)).

Podríamos parar aquí o invertir otro paso de resolución:
• Cláusula de entrada: animal(tiburón)
• Cláusula central: pez(tiburón) ← animal(tiburón),nadar(tiburón)

P'= { animal(tiburón)
nadar(tiburón)
pez(tiburón) ← animal(tiburón), nadar(tiburón)}

Resolución Inversa: Operador V

Absorción. EJEMPLO LARGO (cont.):



Resolución Inversa: Operador V

Operador V: Segundo Caso. Identificación

Supongamos que es C_1 la cláusula que contiene el literal positivo en la ecuación $C=(C_1 - l_1)\theta_1 \cup (C_2 - l_2)\theta_2$. El operador de *identificación* construye C_1 a partir de C y C_2.

La idea es identificar parte del cuerpo de C_2 en el cuerpo de C a través de una sustitución θ.
Supongamos que las cabezas de C y C_2 unifican con θ, y hay un literal *l* en el cuerpo de C_2 que no ocurre en el de C.
La identificación construye C_1 con cabeza *l* y cuerpo la parte de C que no está en C_2.

Resolución Inversa: Operador V

Identificación. <u>EJEMPLO</u>:

C:pájaro(tweety):-tiene_plumas(tweety),
tiene_alas(tweety),tiene_pico(tweety).

C_2:pájaro(tweety):-vuela(tweety), tiene_pico(tweety).

l: vuela(tweety)

C – C_2: { tiene_plumas(tweety), tiene_alas(tweety)}

C_1: vuela(tweety):- tiene_plumas(tweety), tiene_alas(tweety).

$$\frac{q \leftarrow A, B \qquad p \leftarrow A, q}{q \leftarrow B \qquad p \leftarrow A, q}$$

Resolución Inversa: Operador V

Identificación. <u>EJEMPLO</u>:

C:pájaro(tweety):-tiene_plumas(tweety), nada(tweety)
tiene_alas(tweety),tiene_pico(tweety).
C_2:pingüino(tweety):-nada(tweety), pájaro(tweety).

l: pájaro(tweety)

C_1: pájaro(tweety):- tiene_plumas(tweety),
tiene_alas(tweety),tiene_pico(tweety).

$$\frac{q \leftarrow A, B \qquad p \leftarrow A, q}{q \leftarrow B \qquad p \leftarrow A, q}$$

Resolución Inversa: Operador W

El operador W combina dos operadores V.

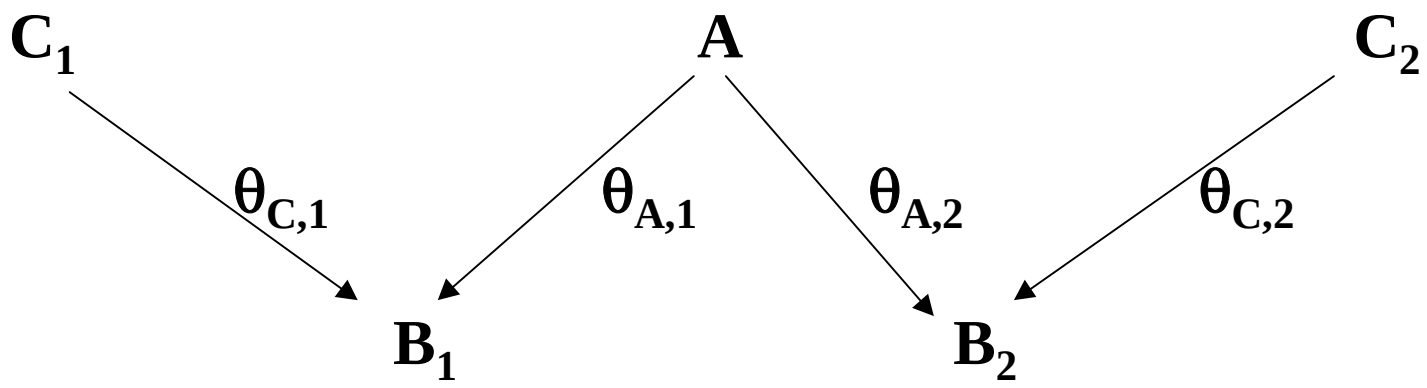
Dadas dos cláusulas $\{B_1,B_2\}$ encontrar $\{C_1,A,C_2\}$ tal que

- B_1 es una instancia de un resolvente de C_1 y A, y
- B_2 es una instancia de un resolvente de A y C_2.

El operador W es capaz de inventar predicados.

Resolución Inversa: Operador W

El operador W construye A, C_1 y C_2 dados B_1 y B_2.
Generaliza $\{B_1,B_2\}$ a $\{C_1,A, C_2\}$



C_1 y C_2 se resuelven sobre un mismo literal *l* de A para obtener B_1 y B_2.

- l negativo: *intraconstrucción*
- l positivo: *interconstrucción*

Resolución Inversa: Operador W

EJEMPLO (intraconstrucción):

B_1: a(X,Z):-p(X,Y), q(Y,Z).
B_2: a(A,C):-p(A,B), r(B,C).

A: s(L,N):-p(L,N).
C_1: a(X,Z):- s(X,Y), q(Y,Z).
C_2: a(A,C):- s(A,B), r(B,C).

EJEMPLO (interconstrucción):

B_1: a(X,Z):-p(X,Y), q(Y,Z).
B_2: a(A,C):-p(A,B), r(B,C).

A: a(L,N):-s(L,N).
C_1: s(X,Z):- p(X,Y), q(Y,Z).
C_2: s(A,C):- p(A,B), r(B,C).

Resolución Inversa: Operador W

Cuando *l* no está ni en B_1 ni en B_2, el operador W se *inventa* predicados.

$$B_1=(A - l)\theta_{A,1} \cup (C_1 - l_1)\theta_{C,1}$$
$$B_2=(A - l)\theta_{A,2} \cup (C_2 - l_2)\theta_{C,2}$$

Se introduce un nuevo concepto (*newp*).

A se genera con los literales en común entre B_1 y B_2, junto con un nuevo predicado *newp*.
Newp se define mediante dos cláusulas que contienen el resto de literales de B_1 y B_2, respectivamente.

Resolución Inversa: Operador W

EJEMPLO:

B_1: abuelo(X,Z):-padre(X,Y), padre(Y,Z).
B_2: abuelo(A,C):-padre(A,B), madre(B,C).

A: abuelo(L,N):-padre(L,M), *newp*(M,N).
C_1: *newp* (Y,Z):- padre(Y,Z).
C_2: *newp* (B,C):- madre(B,C).

Resolución Inversa: Resumen

Absorción:

$$\frac{q \leftarrow A \qquad p \leftarrow A, B}{q \leftarrow A \qquad p \leftarrow q, B}$$

Identificación:

$$\frac{q \leftarrow A, B \qquad p \leftarrow A, q}{q \leftarrow B \qquad p \leftarrow A, q}$$

Intra-construcción:

$$\frac{p \leftarrow A, B \qquad p \leftarrow A, C}{q \leftarrow B \qquad p \leftarrow A, q \qquad q \leftarrow C}$$

Inter-construcción:

$$\frac{p \leftarrow A, B \qquad q \leftarrow A, C}{p \leftarrow r, B \qquad r \leftarrow A \qquad q \leftarrow r, C}$$

Implicación Inversa

La implicación no es decidible en lógica de primer orden.
Sin embargo, existen algunos resultados:

Lema (Lee 1967).
Una cláusula T implica una cláusula C si y sólo si existe una cláusula D en el cierre de resolución de T, tal que θ-subsume C.

Corolario: (Implicación y recursividad)
Sean C, D dos cláusulas. C implica D si y sólo si D es una tautología o θ-subsume D o existe una cláusula E tal que E θ-subsume D donde E se construye auto-resolviendo C repetidamente.

Por tanto, la diferencia entre θ-subsunción e implicación entre las cláusulas C y D sólo es pertinente cuando C puede auto-resolver.

Implicación Inversa

Por tanto, ha habido intentos de:

a) extender resolución inversa para incluir algunos casos.

Han sufrido problemas de terminación.

b) usar una mezcla de resolución inversa y lgg.

Idestam-Almquist (1992-1995) ha investigado esta segunda opción. Sufre el problema de la intractabilidad de cláusulas grandes.

Ambas aproximaciones son incompletas para invertir implicación, aunque la técnica de Idestam-Almquist es completa para una forma restringida de 'entailment' denominada 'T-implication'.

Consecuencia Lógica (entailment) Inversa

Se utiliza para extraer cláusulas de comienzo (starting clauses):
Si *H* y *E* son cláusulas simples, tenemos:

$$B \wedge H \models E$$

se puede reescribir:

$$B \wedge \neg E \models \neg H$$

donde ¬E y ¬H serán conjuntos de hechos skolemizados:

$$B \wedge \neg E \models \neg \bot \models \neg H$$

donde $\neg\bot$ es el cjto. de literales ground que son consecuencia lógica de $B \wedge \neg E$.

De ahí tenemos que $\bot$ es la cobertura más específica para el ejemplo. Es una cláusula de comienzo. (también se usa en abducción)

Por tanto, H debe ser más general que $\bot$.

Consecuencia Lógica (entailment) Inversa

Ejemplo:

B = { animal(X) ← pet(X).
pet(X) ← dog(X). }

E = nice(X) ← dog(X).

¬E = { dog(sk). ← nice(sk). }

¬⊥ = { pet(sk), anim(sk), dog(sk). ← nice(sk) }.

⊥ = nice(X) ← pet(X), anim(X), dog(X).

Esta cláusula se puede utilizar como base de un grafo de refinamiento de cláusulas (puesto que H será más general). *Esto es lo que utiliza Progol (Muggleton 1995), aunque la búsqueda es top-down. La búsqueda se guía por el tamaño de los programas.*

Sistemas Bottom-Up

Cigol (Muggleton and Buntine 1988): interactivo, incremental y de aprendizaje de múltiples predicados.

- Está basado en la resolución inversa para cubrir nuevos ejemplos no implicados por la teoría.
- Utiliza el operador W para inventar predicados.
- Heurística: compresión.
- El usuario actúa como oráculo: se le hacen preguntas de si ciertas cláusulas son ciertas.

Marvin (Sammut & Banerji 1996) e Itou (Rouveirol 1992) también son bottom-up y basados en resolución inversa.

Sistemas Bottom-Up

Golem (Muggleton and Feng 1992): no-interactivo, batch (no-incremental) y de aprendizaje de un solo predicado.

- Basado en rlgg.
- Utiliza un límite de resolución de *h* pasos.
- Se limita a cláusulas *ij*-determinadas.
  - *i* representa la profundidad de variables.

    p(X,Y) :- q(Y,Z), r(Z, Z), s(Z, V), t(V)

    tiene profundidad 3. Si hay variables frescas no conectadas previamente la profundidad es infinita.
  - *j* representa el número máximo de la aridad de los predicados.
  - Determinada significa que utilizando B, el literal *k* del cuerpo están determinados si se sustituyen los *k*-1 anteriores.
- *i*, *j* y *h* son parámetros.

Métodos Top-Down

De más general (top) a más específico. Parten de predicados con todos los argumentos con variables y empiezan a especializar hasta excluir los ejemplos negativos.

- Operadores de especialización:
  - Operadores clásicos (inspirados en ML):
    - añadir condiciones.
    - transformación de variables en constantes.
  - Uso de reglas deductivas:
    - sustituciones.
    - implicación.

Métodos Top-Down

Operadores de Especialización.

- Como vimos no es útil el lgg (y menos un rlgg) de una sola cláusula (es ella misma). Existía para dos o más cláusulas.
- En los métodos top-down partimos de una sola cláusula, generalmente $p(X_1, X_2, ..., X_n)$ para cada predicado que se quiere aprender.

- POR TANTO NO SE PUEDEN UTILIZAR MÉTODOS BASADOS EN glb (lo que daría una lss, especialización menos específica).

Métodos Top-Down

Casos particulares de operadores deductivos:

- añadir condiciones: p:- q, r, s $\rightarrow$ p:- q, r, s, t
- unfolding: { p:- q, t t:- r, s } $\rightarrow$ p:- q, r, s
- sustituciones: p(X,Y) $\rightarrow$ p(a,f(Z)), especializan si no hay negación.

Se pueden ver como condiciones:

p(X,Y) :- X = a, Y = f(Z).

También se utilizan los llamados:

"operadores de refinamiento" hacia abajo

(downward refinement operators).

(a los de generalización ya vistos, también se les conoce a veces como operadores de refinamiento hacia arriba)

Operadores de Refinamiento

Un operador de refinamiento se define como una función sobre cláusulas en conjuntos de cláusulas:

Pueden crearse operadores de refinamiento para problemas concretos y usar diferentes para varios problemas.
Por ejemplo, un operador de refinamiento para predicados de aridad 1 sobre naturales sería:

$$\rho(C) = \begin{cases} \{ P(X) \} & \text{si } C = \bullet \\ \{ P(s^{n+1}(X)), (P(s^n(X)) \leftarrow P(X)), P(s^n(0)) \} & \text{si } C = P(s^n(X)) \\ 0 & \text{en otro caso.} \end{cases}$$

Operadores de Refinamiento

Propiedades Deseables de un Operador de Refinamiento:

Operador de Refinamiento Ideal:

Un operador es ideal si cumple las tres propiedades siguientes:

- Finito: para cualquier cláusula C, $\rho(C)$ es finito.
- Completo: cualquier especialización de una cláusula C debería ser alcanzable utilizando un número finito de aplicaciones del operador.
- Propia: no existe un C tal que $C \in \rho^k(C)$, entendiendo $\rho^k(C)$ como la aplicación repetida del operador a los miembros resultantes de la aplicación anterior. Es decir, el operador de refinamiento no se puede quedar enganchado en un bucle aplicándose a cláusulas equivalentes.

Operadores de Refinamiento

Operador de Refinamiento Óptimo:

- Son aquellos que para todo par de cláusulas C y D tal que C es estrictamente más general que D, sólo existe una ρ-cadena de C a D.

  Esto quiere decir que en vez de un grafo de refinamiento tendríamos un árbol.

Operadores de Refinamiento

Resultados:

- Sólo existen operadores de refinamiento ideales (i.e. finitos, completos y propios) para *átomos* utilizando subsunción.
- Para cláusulas y subsunción (y peor aún implicación) no existen operadores de refinamiento ideales.
- Existen, sin embargo para teorías lógicas ordenadas por implicación lógica, sí que existen operadores finitos y completos, pero no propios.
- No existen operadores óptimos para cláusulas.

Sistemas Top-Down

Progol (Muggleton 1995).
Es un sistema no interactivo, no incremental y que aprende múltiples predicados.
Se basa en cláusulas de base (bottom clauses) o de comienzo (starting clauses) generadas por inverse entailment.
Aunque el inverse entailment genera la más específica, ésta se generaliza lo máximo e inspira el descenso hacia esta cláusula más específica:

Muy expresivo:

- *B = programas normales,*
- *E = cláusulas de Horn,*
- *H = programas definidos con recursión.*

Sistemas Top-Down

Progol (Muggleton 1995). Ejemplo:

B = { animal(X) ← pet(X).
pet(X) ← dog(X). }
E = nice(X) ← dog(X).
¬E = { dog(sk). ← nice(sk). }
¬⊥ = { pet(sk), anim(sk), dog(sk). ← nice(sk) }.
⊥ = nice(X) ← pet(X), anim(X), dog(X).

Se generaliza ⊥ al máximo (p.ej. nice(X)).
La búsqueda se orienta hacia abajo introduciendo condiciones (p.ej. pet(X), anim(X), dog(X), etc. una a una).

El descenso se guía por el tamaño de los programas.
Progol asegura que se encuentra el programa más corto que cubre los ejemplos positivos y ningún negativo.

Sistemas Top-Down

Spectre (Boström and Almquist 1994, Boström 1995).
No-interactivo, no-incremental, de un solo predicado.

El operador de especialización es una combinación de unfolding y borrado de cláusulas.

Spectre II (Boström 1995b) permite ya aprender múltiples predicados.

Sistemas Top-Down

Claudien (De Raedt and Bruynooghe 1993).
Es un sistema *no supervisado*, no interactivo y no incremental.
El sistema comienza con la cláusula vacía • y utiliza un operador de refinamiento hacia abajo para encontrar las cláusulas más generales.

Es el único que permite inducir programas generales (full) y no sólo definidos o normales.

- *H = programas generales.*

Métodos Top-Down / Árboles de Decisión

La solución a un problema de inducción en ILP se puede representar como un conjunto de cláusulas *H*.

- Si las cláusulas sólo tienen un átomo en cabeza, tendremos que hay algunas cláusulas cuyas cabezas son compatibles (mismo predicado) y algunos cuerpos tendrán también cláusulas compatibles.

Ejemplo:

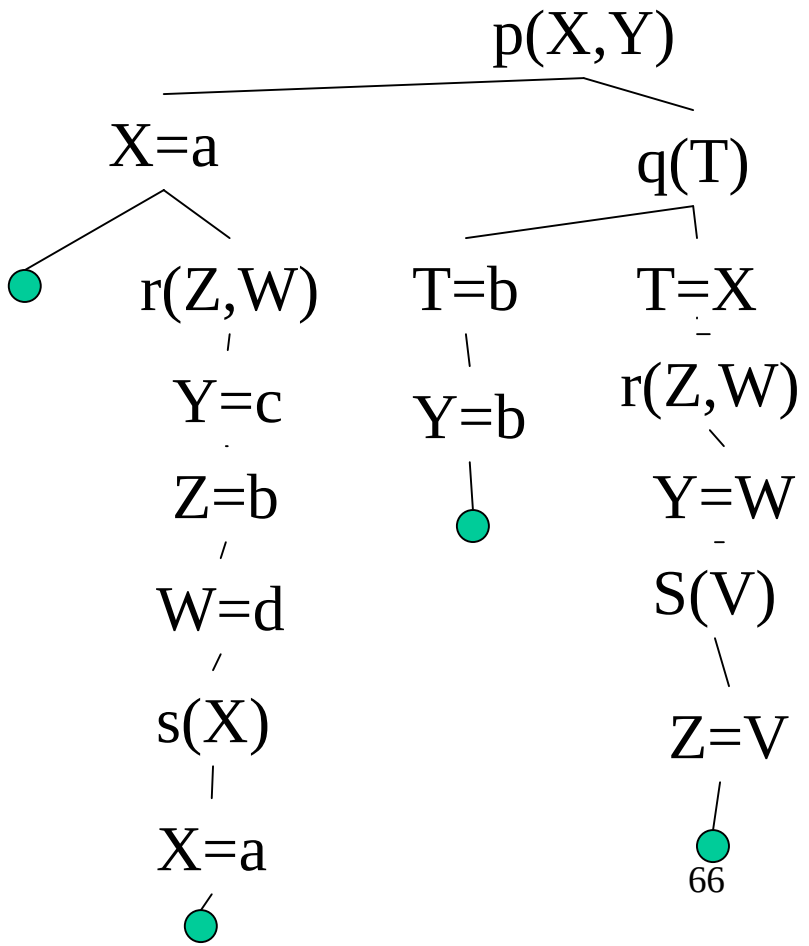
```
p(X,Y) :- q(X), r(Z,Y), s(Z).
p(a, W).
p(X,b): - q(b).
p(a,c):- r(b,d), s(a).
r(W,Z).
```

Métodos Top-Down / Árboles de Decisión

Por cada cabeza compatible podemos representar las cláusulas apropiadas con un árbol.

Ejemplo:

p(X,Y) :- q(X), r(Z,Y), s(Z).
p(a,W).
p(X,b): - q(b).
p(a,c):- r(b,d), s(a).
r(W,Z).



Pueden existir muchos árboles diferentes para el mismo conjunto de cláusulas con cabeza compatible.

Métodos Top-Down / Árboles de Decisión

Definición del Árbol de Decisión ILP:

- Cada nodo representa un predicado o una condición.
- La cabeza se interpreta de manera positiva y el resto de manera negativa (i.e. en cuerpo).
- Los caminos del árbol representan cláusulas.

Modo de Construir el Árbol

- Se crea un nodo raíz por cada predicado a aprender, donde todos los atributos corresponden a variables frescas y no hay cuerpo.
- Las particiones van añadiendo ramas y por tanto condiciones a la cláusula, condiciones que pueden ser incluidas de *B*.

Sistemas Top-Down / Árboles de Decisión

LINUS (Lavrac et al. 1991).

Supervisado, no-interactivo y no-incremental.

Transforma los problemas ILP en una representación de atributos. El resultado se resuelve con un sistema de aprendizaje de atributos, como árboles de decisión.
Aunque también se puede utilizar otro algoritmo de aprendizaje de reglas proposicionales.

H = programas normales sin funciones y no recursivos, y con algunas restricciones.

Sistemas Top-Down / Árboles de Decisión

LINUS: EJEMPLO. Aprender el concepto *daughter*

B= { parent(eve, sue). parent(ann, tom). parent(pat,ann). parent(tom,sue).
female(ann). female(sue). female(eve). }

E^+= { daughter(sue,eve). daughter(ann,pat). }

E^-= { daughter(tom,ann). daughter(eve,ann). }

LINUS transforma B y E a un problema de atributos (proposicional):

Clase	Variables		Atributos proposicionales						
	X	Y	fem(X)	fem(Y)	par(X,X)	par(X,Y)	par(Y,X)	par(Y,Y)	X=Y
+	sue	eve	true	true	false	false	true	false	false
+	ann	pat	true	false	false	false	true	false	false
–	tom	ann	false	true	false	false	true	false	false
–	eve	ann	true	true	false	false	false	false	false

Resultado del aprendizaje de atributos:

class = + if (female(X) = true) $\wedge$ (parent(Y,X) = true)

LINUS transforma de nuevo a cláusulas:

daughter(X,Y) :- female(X), parent(Y,X).

Es simplemente un ejemplo de Pick & Mix

Sistemas Top-Down / Árboles de Decisión

FOIL (Quinlan 1990, Quinlan & Cameron-Jones 1993, Cameron-Jones & Quinlan 1993, 1994):

No-interactivo, no-incremental, de simple predicado. Construye eficientemente (y de manera voraz) un árbol de decisión.

- Características especiales.
  - Inspirado en ID3 y C4.5.
  - El criterio de partición (split criterion) es el mismo que C4.5, la ganancia de información.
  - Una restricción (heurística) es que una nueva condición debe incluir al menos una variable ya aparecida en la cláusula.
  - También evita recursión infinita.

Sistemas Top-Down / Árboles de Decisión

FOIL:

H = programas normales sin funciones y recursividad limitada.

- El lenguaje de representación no es exactamente cláusulas de Horn.
  - Es más restringido, porque no permite funciones.
  - Es más extenso, porque permite literales negados en el cuerpo de las reglas (incluyendo igualdad y su negación).
- La búsqueda es de general a específico en el bucle interno (se añaden condiciones a las reglas, es decir se desciende por una rama) y de específico a general en el bucle externo (se añaden más reglas, es decir se crean más ramas). *Sería bidireccional.*

Difiere de C4.5 en que no se hacen particiones sino que sólo se va haciendo una sola rama entera!! (i.e. no se reaprovecha parte de ramas anteriores para otras ramas)

Sistemas Top-Down / Árboles de Decisión

FOIL: Algoritmo:

FOIL(Target_predicate, Predicates, Examples)
Pos := ejemplos para los cuales Target_predicate es True.
Neg := ejemplos para los cuales Target_predicate es False.
Learned_rules := ∅
while Pos <> ∅ **do** // Aprender una nueva regla
  NewRule :- el target_predicate con todo variables y sin condiciones.
  NewRuleNeg :- Neg;
  while NewRuleNeg <> ∅ **do** // Añadir condiciones para especializar la regla
    Candidate_Literals :- generar posibles condiciones (utilizando el cjto. Predicates)
    Best_Literal :- seleccionar el mejor literal utilizando el criterio de ganancia.
    Añadir este Best_Literal a las condiciones de NewRule;
    Quitar a NewRuleNeg los ej. neg. que ya no son cubiertos (incumplen Best_Literal)
  endwhile
  Learned_rules :- Learned_rules ∪ { NewRule}
  Pos :- Pos - {miembros de Pos cubiertos por la nueva regla).
endwhile;
Retornar Learned_rules;

Sistemas Top-Down / Árboles de Decisión

FOIL: Aprendiendo Reglas Recursivas.
¿Qué pasa si en Predicates ponemos también el Target_Predicate?

Permite inducir reglas recursivas:
    Target_Predicate: ancestor
    Predicates: parent, ancestor
Programa inducido:
    ancestor(X,Y) :- parent(X,Y).
    ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).

El problema es que pueden incluir no terminación.
FOIL incluye algunas heurísticas para evitarlo (Cameron-Jones &
    Quinlan 1993):

Sistemas Top-Down / Árboles de Decisión

FOCL (Pazzani et al. 1991): mejora del FOIL: combina inducción - abducción (analítico) y funciona con teorías aproximadas. Es en realidad un revisor.

Ejemplo: Considérese un *T* aproximado:

cup :- stable, liftable, openvessel
stable :- bottomisflat
openvessel :- hasconcavity, concpointsup
liftable :- graspable, light
graspable :- hashandle

No se busca
$T \cup H \models E$
sino
$B \cup H \models E$

Y los siguientes ejemplos:

	cups				non-cups					
BottomIsFlat	y	y	y	y	y	y	y			y
ConcavityPointsOut	y	y	y	y	y		y	y		
Fragile	y	y			y	y		y		y
HandleOnTop					y		y			
HandleOnSide	y			y					y	
HasConcavity	y	y	y	y	y		y	y	y	y
HasHandle	y			y	y		y		y	
Light	y	y	y	y	y	y	y		y	

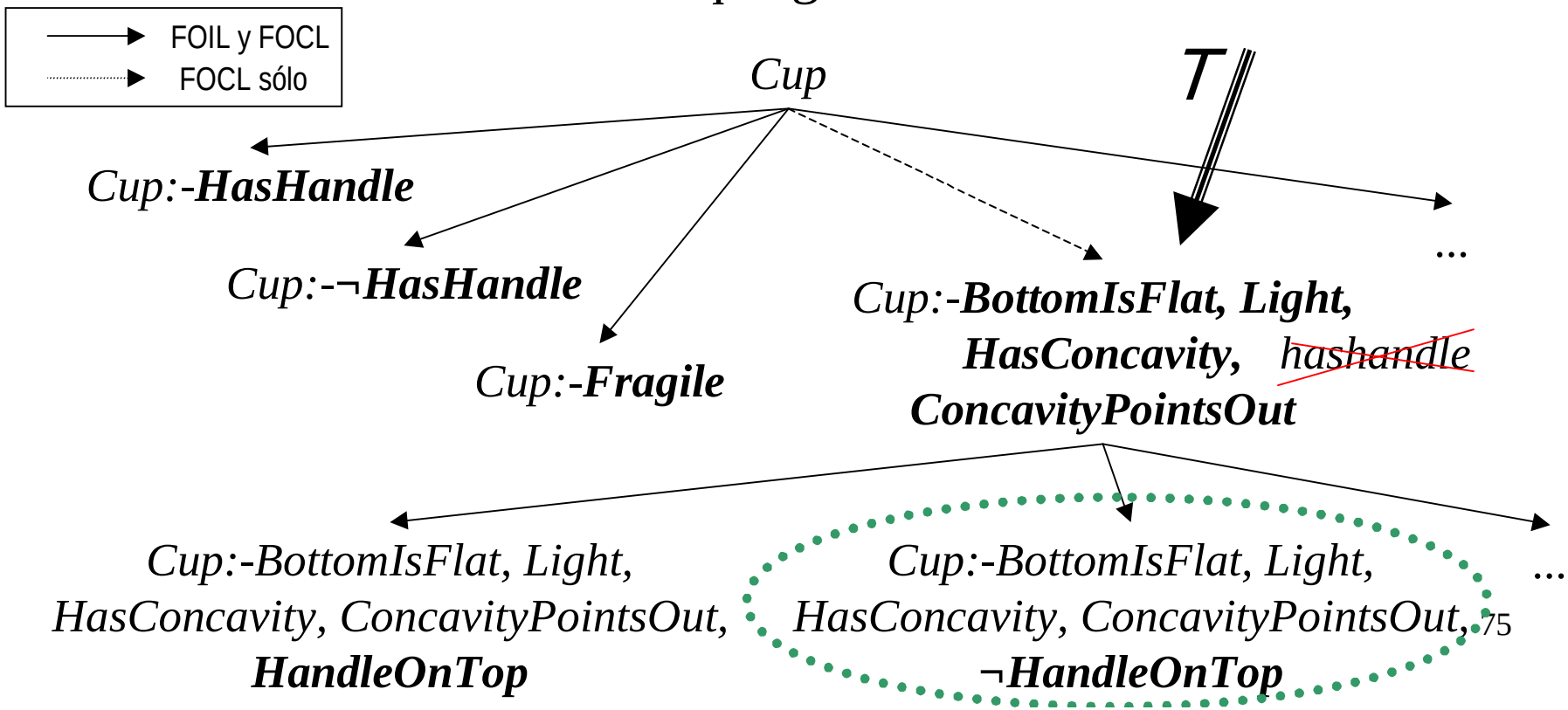
T es una especie de ayuda a la hipótesis.

Sistemas Top-Down / Árboles de Decisión

FOCL: Ejemplo:

FOCL podría explotar este B para guiar la búsqueda para una teoría exacta para los ejemplos:

Sería mucho más eficiente que generar la teoría desde cero:



Sistemas Top-Down / Árboles de Decisión

Tilde (Top-down Induction of first-order Logical Decision trEes).
(Blockeel and De Raedt 1998)

- Es un sistema no supervisado (los ejemplos son interpretaciones).

- Permite resolver tanto problemas de clasificación como de *clustering* (segmentación).

Métodos Bidireccionales ‘Servo’

No son ni bottom-up pero la búsqueda sí que está dirigida por generalización-especialización:
Como los reguladores servo, van descendiendo y cuando se pasan cambian de sentido con un recorrido menor y así hasta que alcanzan el objetivo.

Métodos de Inferencia de Modelos

Aunque su nombre es muy genérico, suelen denominar a un tipo especial de métodos bidireccionales servo e incrementales que funcionan de la siguiente manera:

Se empieza con una teoría inicial H:

Para cada nuevo ejemplo leído:

- *si H es demasiado fuerte, i.e., cubre ejemplos negativos. En este caso, la H contiene al menos una cláusula que es falsa. Se trata de buscar una de estas cláusulas y borrarla.*
- *Si H es demasiado débil, i.e., no cubre todos los ejemplos positivos. En este caso se recupera una cláusula borrada anteriormente y se especializa utilizando operadores de refinamiento.*

Evidentemente se debe partir de teorías generales.

Métodos de Inferencia de Modelos

EJEMPLO (Nienhuys-Cheng & de Wolf 1997):

Supongamos E de la siguiente manera y orden:

$p(0)$

$\neg p(s(0))$

$p(s(s(0)))$

$\neg p(s(s(s(0))))$

...

Como es una función sobre los números naturales con un solo argumento podemos utilizar el operador de refinamiento de antes:

$$\rho(C) = \begin{cases} \{ P(X) \} & \text{si } C = \bullet \\ \{ P(s^{n+1}(X)), (P(s^n(X)) \leftarrow P(X)), P(s^n(0)) \} & \text{si } C = P(s^n(X)) \\ 0 & \text{en otro caso.} \end{cases}$$

Métodos de Inferencia de Modelos

EJEMPLO (cont.):

1. H := {• }
2. Leer (P(0), T).
3. H no es ni demasiado fuerte ni demasiado débil.
4. Leer (P(s(0)), F).
5. H es demasiado fuerte, ya que el ejemplo falso P(s(0)) se puede deducir. La cláusula falsa • se borra y ahora H = ∅.
6. H es ahora demasiado débil, porque el ejemplo positivo P(0) no se puede deducir. Se añade ρ(•) = {P(x)} a H.
7. H es ahora demasiado fuerte. Se borra la cláusula falsa P(x). De nuevo H = ∅.
8. H es ahora demasiado débil. Añadir ρ(P(x)) = {P(s(x)), (P(x)←P(x)), P(0) } a H.
9. H es ahora demasiado fuerte. Borrar la cláusula P(s(x) y la tautología superflua P(x)←P(x). Por fin H = {P(0)} no es ni demasiado fuerte ni demasiado débil.
10. Leer (P(P(0)), T).
11. H es muy débil. Añadir ρ(P(s(x))) = {P(s^2(x)), (P(s(x))←P(x)), P(s(0)) } a H.
12. H es demasiado fuerte. Borrar P(s(x))←P(x) y P(s(0)) . De nuevo H = {P (s^2(x)), P(0)} no es ni demasiado fuerte ni demasiado débil.
13. Leer (P(s^3(0)), F)
14. H es demasiado fuerte. Borrar P(s^2(x)) de H.
15. H es ahora demasiado débil. Añadir ρ(P(s^2(x))) = {P(s^3(x)), (P(s^2(x))←P(x)), P(s^2(0)) }
16. H es demasiado fuerte. Borrar P(s^3(x)) de H.
17. Ahora H = {P(0), (P(s^2(x))←P(x)), P(s^2(0)) } es correcto para el resto de ejemplos y ya no cambiará.

Sistemas de Inferencia de Modelos

Sistema MIS (Shapiro 1981b, Shapiro 1983).
Es de los sistemas más antiguos.

Se empieza con una teoría inicial {• }*:*

- *Si H es demasiado fuerte, se descubre la cláusula que es falsa mediante un método denominado "BACKTRACKING". Una vez encontrada se borra.*
- *Si H es demasiado débil, i.e., no cubre todos los ejemplos positivos. En este caso se recupera una cláusula borrada anteriormente y se especializa utilizando operadores de refinamiento.*

Otros Métodos

- *Genetic Logic Programming* (GLP): recodifican los programas para poder ser usados con algoritmos genéticos (Augier and Venturini 1995, 1996) (Osborn, Chariff, Lamas and Dubossarsky).
- *Genetic Inductive Logic Programming* (GILP): basados en programación evolutiva. Los individuos son programas lógicos. Se definen operadores especiales de cross-over, mutación, etc. que consisten en especializaciones y generalizaciones así como intercambio de condiciones, foldings, unfoldings (Varsek 1993) (Ichise 1998).

En programación funcional, existen trabajos previos sobre el lenguaje ML, como la tesis de Olson (Olson 1995)

Métodos Guiados por Esquemas

Se trata de especifica cómo ha de ser el programa:

- En realidad por esquema se entiende casi cualquier restricción sobre el mismo, gran parte de lo que se llama usualmente *bias*.
- Por esquema se entiende más cualquier bias que afecte a la *forma* del programa.

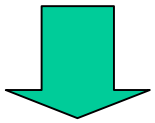
Método más simples:

- *Uso de tipos*.
- *Uso de modos. Se determina qué argumentos son de entrada y de salida. Necesario para problemas de clasificación en los que un argumento es la clase.*

Métodos Guiados por Esquemas

Método más usual:

- *Modelos de reglas* = cláusula en la que los símbolos de predicado se han reemplazado por variables (de predicado)



esquema de segundo orden

Métodos Guiados por Esquemas

EJEMPLO:

Esquema:

```
P(X,Y,Z):-Q(X1,X2,X), R(X1,Y), P(X2,Y,W), S(X1,W,Z).
```

B:

```
cons(X,Y,[X|Y]).
member(X,[X|Y]).
member(X,[_|Y]):-member(X,Y).
no_op(X,Y).
```

Dos cláusulas que se ajustan al esquema anterior:

```
append(X,Y,Z):-cons(X1,X2,X), no_op(X1,Y), append(X2,Y,W),
                cons(X1,W,Z).

intersection(X,Y,Z):-cons(X1,X2,X), member(X1,Y),
                      intersection (X2,Y,W), cons(X1,W,Z).
```

Métodos Guiados por Esquemas

Otros Tipos de Esquemas:

- Plantillas de cláusulas: los argumentos de los literales también pueden ser variables ⇒ definir predicados de diferentes aridades.

  subset(X,Y) :- cons(X1,X2,X), member(X1,Y), subset(X2,Y),
  no_op(X1,Y).

- Alternativas: *bias* generativos⇒ conjuntos de operadores que construyen las cláusulas incrementalmente, instanciando variables o añadiendo literales en el cuerpo.

Sistemas Basados por Esquemas

MOBAL (Kietz and Wrobel 1992).
Se trata de un entorno integrado de adquisición de conocimiento, que consiste de partes muy diversas. Una de estas partes es el sistema de aprendizaje RDT.

Utiliza el primer tipo de esquemas (modelos de reglas).
Va instanciando estos esquemas orientándose (con una búsqueda top-down) por la dependencia que detecta que pueda haber entre los predicados de *B*.

B = conjunto de literales (si B es normal lo transforma).
H = programas normales sin funciones.

Métodos y Sistemas Basados por Esquemas

Para una comparación completa y sus aplicaciones en la síntesis de programas (Flener & Yilmaz 1999).

Tabla en la transparencia siguiente...

	special-purpose recursion-only synthesisers												general-purpose				
	schema-biased								schema-less								
generality model [a]	ii	ii	**θ**	**θ**	**θ**	**θ**	**θ**	**θ**	ir	ir	**θ**	n/a	**θ**	**θ**	r**θ**	ie	n/a
interactive / passive [b]	p	i	p	p	p	p	p	i	p	p	p	p	i	i	p	i/p	i
data-driven / theory-guided [c]	dd	dd	dd	dd	dd	dd	dd	dd	tg	tg	dd	dd	tg	tg	dd	dd	dd
	CRUSTACEAN	CILP	FORCE2	SIERES	TIM	SYNAPSE	METAINDUCE	DIALOGS	SPECTRE II	MERLIN	SMART	SKILIT	MIS	CIGOL	GOLEM	PROGOL	FILP
evidence language [d]	gl	gl	gl	gl	ga	np	ga	np	gl	gl	gl	gl	gl	dp	gl	Hc	ga
# predicates in evidence	1	1	1	1	1	1	1	1	≥1	1	1	1	≥1	≥1	1	≥1	≥1
semantic manipulation?			×	×	×	×		×	×	×	×	×	×	×	×	×	×
from arbitrary evidence?	×	×	×	×	×			n/a			×	×			×		×
type information?								×			×	×	×			×	
mode information?				×	×	n/a		n/a			×	×	×		×	×	×
determinism information?						n/a		n/a					×			×	×
other bias? (except schemas)			×								×	×	×		×	×	×
background knowl. language [d]			gl	dp	dp	np		np			dp	dp	dp	dp	dp-	np	dp-
hypothesis language [d]	dp	dp	dp	dp	dp	np	dp	np	dp	dp	dp	dp	dp	dp	dp-	dp	dp-
# base clauses / predicate	1	1	1	?	1	≥1	1	≥1	≥1	≥1	1	≥1	≥1	≥1	≥1	≥1	≥1
# recursive clauses / predicate	1	1	1	1	1	≥1	1	≥1	≥1	≥1	1	≥1	≥1	≥1	≥1	≥1	≥1
# recursive calls / clause	1	1	1	≥1	1	≥1	1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1
necessary predicate invention?		×		×		×	×	×		×				×			
useful predicate invention?						×		×									
handling of sparseness?	×	×						×			×						

a. **θ** = **θ**-subsumption, r**θ** = relative **θ**-subsumption, ir = inverse resolution, ii = inverse implication, ie = inverse entailment
b. i = interactive, p = passive
c. dd = data-driven, tg = theory-guided
d. ga = ground atoms, gl = ground literals, dp = definite programs, dp- = some restricted form of dp, np = normal programs, Hc = Horn

ILP y Evidencia Positiva

Problemas de ILP y evidencia positiva.

La hipótesis más general es válida: $p(X_1, X_2 \ , ..., X_n)$

- Afecta notablemente al paradigma ILP porque la evidencia es un conjunto y no una secuencia.
- Afecta frecuentemente. Los problemas de clasificación con más de dos clases deben reformularse en ILP con sólo evidencia positiva. Si no se usan modos tenemos un problema de evidencia positiva.
- Especialmente problemático para los sistemas top-down y bidireccionales.

ILP y Evidencia Positiva

Problemas de ILP y evidencia positiva.

Primera Solución: MDL aplicado a ILP (Conklin & Witten 1994)

$$h_{MDL} = \arg\min_{k \in H}(K(h) + K(D \mid h))$$

Resuelve algunos problemas, porque teorías más específicas (no demasiado más largas) hacen mucho más pequeño K(D|h).

Pero otros no, por ejemplo:

E: odd(33). odd(7). odd(29). odd(15). odd(123). odd(45).

H_1: odd(X).

H_2: odd(s(0)).
odd(s(s(X))) :- odd(X).

H_2 no consigue minimizar K(D|h) respecto H_1.

ILP y Evidencia Positiva

Problemas de ILP y evidencia positiva.

Segunda Solución: (uso de la generalidad de una hipótesis)

- combinado con MDL: (Conklin & Witten 1994)(Muggleton 1999)
- combinado con refuerzo: (Hernández 2000b)

EJEMPLO

E: odd(33). odd(7). odd(29). odd(15). odd(123). odd(45).

H_1: odd(X).

H_2: odd(s(0)).
odd(s(s(X))) :- odd(X).

- H1 es mucho más general que H2.

¿Cómo se computa la generalidad de una hipótesis?

- Generando un número suficiente de términos aleatoriamente (distribución universal) y viendo el porcentaje de TRUE's sobre el total.

$G(H_1) = 1$ $G(H_2) = 0.5$

ILP. Aplicaciones

Áreas de Aplicación:

- Adquisición de **conocimiento** para sistemas expertos.
- Descubrimiento de **conocimiento** en bases de datos.
- Descubrimiento de **conocimiento** científico.
- Lenguaje natural.
- Síntesis de programas y verificación.
- Agentes/control.

ILP. Aplicaciones

Áreas de Aplicación:

- Adquisición de **conocimiento** para sistemas expertos.
  - Existe un gran desarrollo en I.A. para el uso de la programación lógica como representación de conocimiento (razonamientos no monótonos, revisión de teorías, modularidad de teorías, etc.).
  - La adquisición del conocimiento era manual. Utilizando ILP, esta adquisición puede realizarse de manera (semi-) automática. Un ejemplo es el sistema MOBAL.

ILP. Aplicaciones

Áreas de Aplicación:

- Descubrimiento de **conocimiento** en bases de datos.
  - ILP permite tratar directamente con el esquema *relacional* de una base de datos relacional.
  - Permite aprovechar las restricciones de integridad (CP, CAj, R.I. Generales, etc.), poniéndolas directamente en el conocimiento previo.
  - Además, el conocimiento obtenido es inteligible y se puede añadir directamente como una R.I.
  - Hay extensiones inductivas a lenguajes de consulta.

ILP. Aplicaciones

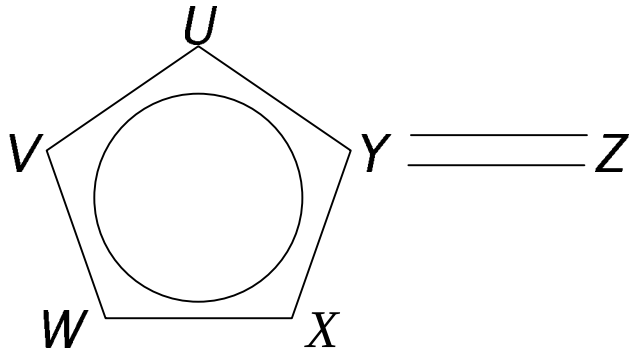
Áreas de Aplicación:

- Descubrimiento de **conocimiento** científico.
  - Aplicaciones más relevantes (diseño de mallas de elementos finitos, predicción de actividad estructural en el diseño de medicamentos, predicción de estructura secundaria de proteínas, predicción de mutagénesis de nuevos compuestos químicos.
  - Suelen ser problemas intrínsecamente relacionales, con relativamente no muchos ejemplos...
  - Algunos resultados han aparecido en revistas científicas del área de la aplicación, como son “Protein Engineering” (King et al. 1992) (Muggleton et al. 1992).

ILP. Aplicaciones

Áreas de Aplicación:

- Descubrimiento de **conocimiento** científico.
- Ejemplo: MUTAGÉNESIS:
  - Teoría descubierta por Progol:
  - "A molecule is mutagenic if there exists a double bond connected to a 5-aromatic ring"



Progol	85.7%
BackProp	64.3%
Regression	66.7%
Default	69.0%

Ver:

(Muggleton 1999b) Muggleton, S. "Scientific Knowledge Discovery using Inductive Logic Programming",

ILP. Aplicaciones

Áreas de Aplicación:

- Lenguaje natural. **LLL** (Logic, language & learning).
  - La programación lógica se ha utilizado extensivamente para el lenguaje natural
  - Existencia de conocimiento previo validado en representación lógica (gramáticas).
  - Grandes cuerpos de datos.
  - Necesidad de aproximaciones estocásticas.

Ver:

(Muggleton 1999a) Muggleton, S. "Inductive Logic Programming: issues, results and the LLL challenge"

ILP. Aplicaciones

Áreas de Aplicación:

- Síntesis de programas y verificación.
  - Una de las áreas más prometedoras inicialmente.
  - Probablemente una de las áreas más desalentadoras y olvidadas durante unos años (principios de los 90).
  - Reciente resurgimiento con menos pretensiones (aplicaciones software específicas, semi-automatización, pequeños agentes...).
  - Un review sobre el estado actual del arte en síntesis de programas con ILP (Flener & Yilmaz 1999).

ILP. Aplicaciones

Áreas de Aplicación:

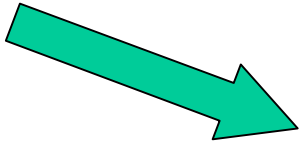
- Agentes/control.
  - La dinámica o contexto se puede utilizar de conocimiento previo.
  - Existencia de simuladores para obtener datos de entrenamiento.
  - Necesidad de sistemas ILP más interactivos y de representaciones de acciones y agentes.

ILP. Problemas Abiertos

Current Research Issues (según Muggleton http://www.cs.york.ac.uk/mlg/index.html)

Background Knowledge.

- Relevance. When large numbers of background predicates are available, determining which predicate is relevant.
- Revision. How clauses should be removed, or unfolded with respect to deep-structured theories.
- Invention. Further research is required into predicate invention.



Si no, no podemos hablar de expresividad universal

ILP. Problemas Abiertos

Current Research Issues (cont.)

- Complex Theories.
  - Multi-clause. Most present ILP systems are concerned with generating a single clause. Research is required into improved performance of multiple clause generation.
  - Deep Theories. Current ILP systems perform poorly in the presence of relevant long chains of literals, connected by shared variables.
  - Recursion. Recursive hypotheses are poorly handled by most ILP system. This is particularly important in the natural language domain.
  - Structure. Structural concepts result in complex clauses when encoded as literals. These present problems to the search strategies used in current ILP systems.

ILP. Problemas Abiertos

Current Research Issues (cont.)

- Built-in semantics.
  - Numbers. ILP systems have severe restrictions on the form of numeric constraints that can be used.
  - Probabilities. ILP systems lack the ability te express probabilistic constraints. This effects the performance of ILP systems in the database discovery domain.
  - Constraints. ILP system require the ability to learn and make use of general constraints, rather than requiring large numbers of ground negative examples.
  - Built-in predicates. Some predicates are best defined procedurally. ILP systems may experience improved efficiency of learning using built-in predicates.

ILP. Problemas Abiertos

Current Research Issues (cont.)

- Sampling Issues.
  - Large Data Sets. Incremental learning systems may be more effective than batch systems, where difficulties are experienced with learning from all examples at once.
  - Small Data Sets. Statistical tests for significance break down when learning from small data sets. ILP systems need to demonstrate high predictive accuracy with small training sets.
  - Reliability. Extending ILP systems to indicate reliability estimates when exact generalisations are not possible.

ILP. Problemas Abiertos

Current Research Issues (cont., tb. de Muggleton)

- Better Representations.
  - Probabilistic logic programs (for hypotheses and for qualifying the relevance of different parts of the background knowledge).
  - Higher-order (for more structured theories, for learning schemas, for organising the background knowledge).
  - Representation of agency, of situations, of actions...

ILP. Conclusiones

De todas las aproximaciones al *machine learning* vistas ésta es **la más próxima al lenguaje natural** al representar el conocimiento como reglas y hechos sobre ciertos predicados, lo que facilita su interpretación.

REPRESENTACIÓN UNIVERSAL: esto es una ventaja y un inconveniente (como hemos visto), debido a la complejidad.

Si se quiere limitar la expresividad, la lógica permite muchísimos más subconjuntos explorados ya o por explorar que limitarse a elegir entre proposicional y 1er orden completo.

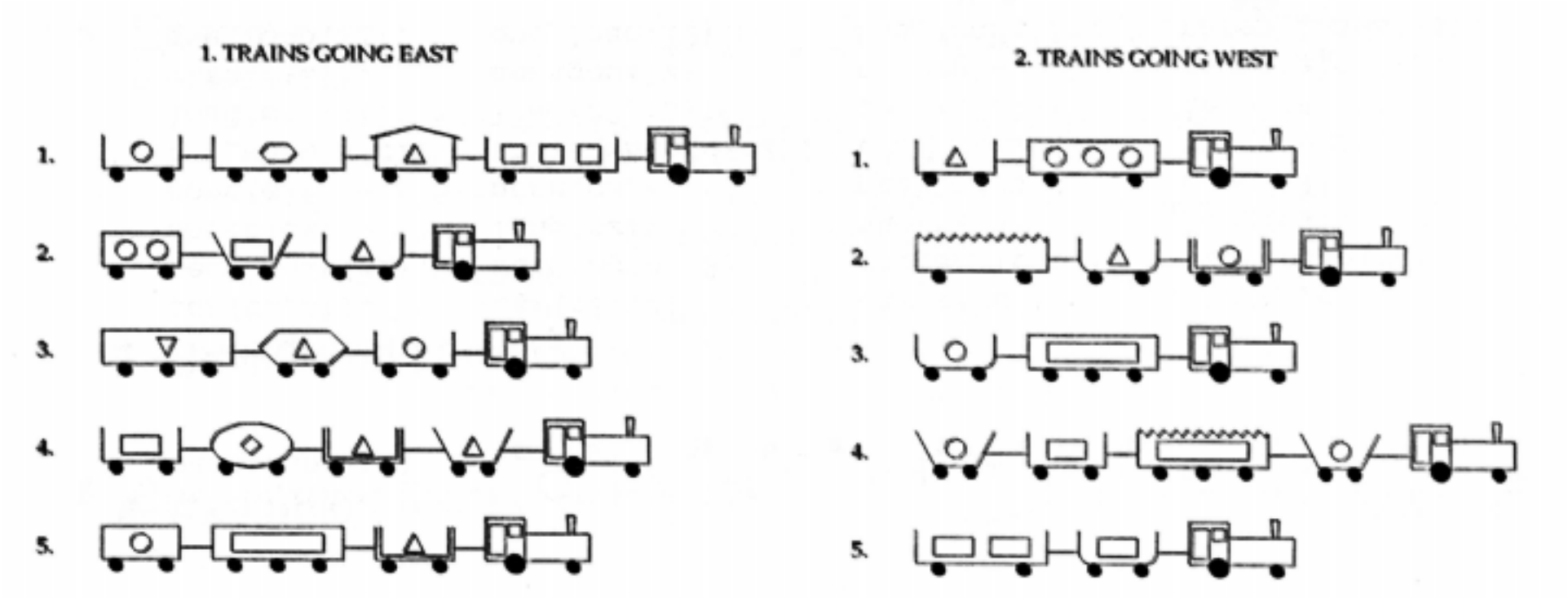
ILP. Limitaciones de Representación

ILP proporciona una REPRESENTACIÓN UNIVERSAL, pero, aún así, presenta inconvenientes para algunos problemas.

- Muchos sistemas aplanan ejemplos complejos por problemas al tratar términos con funciones.
- La mayoría de sistemas añaden modos y tipos para describir el funcionamiento del predicado a aprender.

ILP. Limitaciones de Representación

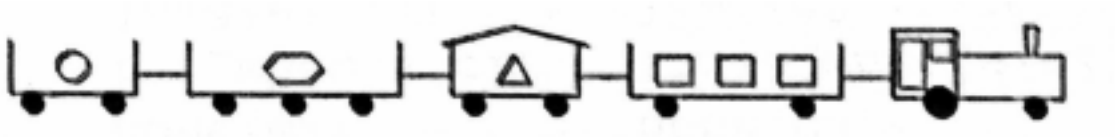
EJEMPLO: EAST-WEST TRAINS



ILP. Limitaciones de Representación

EJEMPLO: EAST-WEST TRAINS. (flattened)

Codificación del Ejemplo 1:



E: eastbound(t1)

B:

car(t1,c1).	rectangle(c1).	short(c1).	none(c1).
two_wheels(c1).	load(c1,l1).	circle(l1).	one_load(l1).
car(t1,c2).	rectangle(c2).	long(c2).	none(c2).
three_wheels(c2).	load(c2,l2).	hexagon(l2).	one_load(l2).
car(t1,c3).	rectangle(c3).	short(c3).	peaked(c3).
two_wheels(c3).	load(c3,l3).	triangle(l3).	one_load(l3).
car(t1,c4).	rectangle(c4).	long(c4).	none(c4).
two_wheels(c4).	load(c4,l4).	rectangle(l4).	three_loads(l4).

H: eastbound(T) :- car(T,C), short(C), not none(C).

ILP. Limitaciones de Representación

EJEMPLO: EAST-WEST TRAINS. (ground clauses)

Codificación del Ejemplo 1: 

E: eastbound(t1) :-
car(t1,c1), rectangle(c1), short(c1), none(c1), two_wheels(c1),
load(c1,l1), circle(l1), one_load(l1),
car(t1,c2), rectangle(c2), long(c2) , none(c2), three_wheels(c2),
load(c2,l2), hexagon(l2), one_load(l2),
car(t1,c3), rectangle(c3), short(c3), peaked(c3), two_wheels(c3),
load(c3,l3), triangle(l3), one_load(l3),
car(t1,c4), rectangle(c4), long(c4), none(c4), two_wheels(c4),
load(c4,l4), rectangle(l4), three_loads(l4).

B: ∅

H: eastbound(T) :- car(T,C), short(C), not none(C).

ILP. Limitaciones de Representación

EJEMPLO: EAST-WEST TRAINS. (datalog)

Codificación del Ejemplo 1: 

E:
train_table(t1, true).
car_table(c1, t1, rectangle, short, none, 2).
car_table(c2, t1, rectangle, long, none, 3).
car_table(c3, t1, rectangle, short, peaked, 2).
car_table(c4, t1, rectangle, long, none, 2).
load_table(l1, c1, circle, 1).
load_table(l2, c2, hexagon, 1).
load_table(l3, c3, triangle, 1).
load_table(l4, c4, rectangle, 3).

B: ∅

H: train_table(T, true) :- car_table(C, T, _, short, W, _), W <> none.

ILP. Limitaciones de Representación

EJEMPLO: EAST-WEST TRAINS. (terms)

Codificación del Ejemplo 1: 

E: eastbound([c(rectangle, short, none, 2, l(circle,1)),
c(rectangle, long, none, 3, l(hexagon, 1)),
c(rectangle, short, peaked, 2, l(triangle, 1)),
c(rectangle, long, none, 2, l(rectangle, 3))]).

B: ∅

H: eastbound(T) :- member(C,T), arg(2, C, short), not arg(3,C,none).

ILP. Limitaciones de Representación

EJEMPLO: APPEND

Append con programación lógica:

```
append([],X,X).
append([X|Y], Z, [X|W]) :- append(Y,Z,W).
```

La mayoría de sistemas ILP requieren modos para aprender la función:

```
append(-,-,+).
```

Append con programación funcional:

```
app([X|Y],Z) = [X|app(Y,Z)]
app([],X) = X
```

No hacen falta modos...

ILP. Limitaciones de Representación

EJEMPLO: MEMBER

Member con programación lógica:

```
member(X, [X|Z]).
member(X, [Y|Z]) :- member(X, Z).
```

Si no se incluye información de tipos, el proceso de inducción podría llegar a considerar términos como los siguientes:

```
member([X|Y], W).
member(X, [Z|W]).
member([X|Y], [Z|W]).
...
```

Con tipos, estos términos no se generan.

IFLP

¿Qué es FLP?

Programa Lógico Funcional: Sistema de Reescritura de Términos (SRT) de la forma:

$$l = r \Leftarrow e_1, \ldots, e_n \text{ with } n \geq 0$$

Subsume la programación lógica y la programación funcional.

Narrowing:

- *Método correcto y completo de ε-unificación.*
- *Más poder expresivo en comparación con los lenguajes funcionales*
- *Mejor comportamiento operacional respecto a los lenguajes lógicos.*

IFLP

¿Qué es IFLP?
IFLP: Inductive Functional Logic Programming

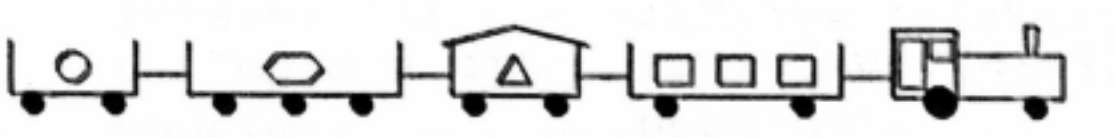
Ventajas de la inducción sobre FLP:

- *El conocimiento previo puede ser más rico: esquemas, biases...*
- *Más poder expresivo → Teorías más compactas.*
- *La relación entre deducción & inducción puede ser considerada en detalle*
- *Más adecuado para problemas de clasificación.*
- *Mejor tratamiento de evidencia positiva (confluencia)*
- *No necesidad de modos y tipos (si se utiliza FLP tipado).*

IFLP. Ventajas de Representación

EJEMPLO 1: EAST-WEST TRAINS. (terms)

Codificación del Ejemplo 1:



E: eastbound([c(rectangle, short, none, 2, l(circle,1)),
c(rectangle, long, none, 3, l(hexagon, 1)),
c(rectangle, short, peaked, 2, l(triangle, 1)),
c(rectangle, long, none, 2, l(rectangle, 3))]) = true.

B: ∅

H: eastbound(T) = true :- member(c(_, short, Z, _, _),T)=true, Z ≠ none.

Estructura de términos compleja

IFLP. Ventajas de Representación

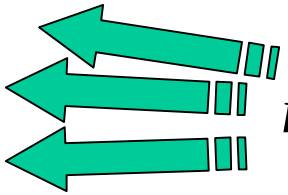
EJEMPLO 2: Clientes con diferente grado de gratificación (30%, 20% or 10%)

E:
gratification(: (: ([],name (john), has_children)) = 30
gratification(: (: (: ([],married), teacher) has_cellular)) = 30
gratification(: (: (: ([],sex(male)), teacher), name (jimmy))) = 30
gratification(: (: ([],name (john), tall)) = 20
gratification(: (: ([],married), police)) = 10
gratification(: (: ([],married), politician)) = 10
gratification(: (: ([],sex(male)), boxer)) = 20
...

B: ∅

H:
gratification(p(X0,X1)) = gratification(X0)
gratification(p(X0,has_children)) = 30
gratification(p(X0,sex(female))) = 30
gratification(p(X0,teacher) = 30
gratification(p(X0,politician) = 10
gratification(p(X0,boxer) = 20
...

recursividad



más de dos clases

IFLP. Operadores

Operadores IFLP:

(Hernandez & Ramírez 1998)

Inverso de unification → *generalisación*

Inverso de narrowing → "*inverse narrowing*"

Consistent Restricted Generalisation (CRG)

La ecuación $e = \{ l_1 = r_1 \}$ de *e' es un CRG respecto* E^+ *y* E^- *y la teoría* $T = B \cup P$ *sii:*

(1) e es una RG de e',

(2) no existe una cadena de narrowing $(s \rightsquigarrow^*_{T \cup e} t)$ *tal que:*

$s=t \in E^-$, *(consistencia respecto.* E^-*)*

(3) para cada ecuación $l = r \in E^+$*, no se puede computar una forma normal de l diferente de t respecto T (consistencia respecto* E^+*).*

IFLP. Operadores

Operadores IFLP:

Reverse Narrowing ($\leftarrow$)

t 'reversely narrows' in t' ($t \leftarrow_\theta t'$) iff

- $u \in O(t)$,
- $l = r$ is a new variant of a rule from P ,
- $\theta = mgu(t_{|u}, r)$, and
- $t' = \theta(t[l]_u)$.

Reverse Narrowing + CRG = Inverse Narrowing.

Generación de Reglas por Inverse Narrowing

r=s *(receiver rule)* t=u *(sender rule)*

1. $s \leftarrow_\theta s'$ w.r.t. t=u
2. CRG(r=s')

IFLP. Operadores

Operadores IFLP. Ejemplo:

Evidence E:

(E_1^+)	0 + 0 = 0	(E_1^-)	s0 + 0 = 0
(E_2^+)	s0 + s0 = ss0	(E_2^-)	0 + 0 = s0
(E_3^+)	0 + s0 = s0	(E_3^-)	s0 + s0 = s0
(E_4^+)	ss0 + s0 = sss0	(E_4^-)	s0 + 0 = ss0
		(E_5^-)	ss0 + s0 = 0

La ecuación e_a = { X + s0 = sX } es un CRG of E^+_2, E^+_3, E^+_4

La ecuación e_b = { X' + 0 = X' } es un CRG of E^+_1

De la ecuación e_a ={X + s0 = <u>sX</u>} seleccionamos t= sX

y una variante fresca de e_b ={X' + 0 = <u>X'</u>}.

Dos ocurrencias: $u_1 = 1$ *da* t'_1= s(X + 0)

$u_2 = \varepsilon$ *da* t'_2= sX + 0

dando dos ecuaciones: $e_{a,1}$ ={X + s0 = <u>s(X+0)</u>}

$e_{a,2}$ ={X + s0 = <u>sX+0</u>}

Es obvio que ambos se 'narrowean' en e_a *utilizando* e_b.

Lo mismo se cumple después de CRG: e'$_{a,1}$={X + sY = <u>s(X+Y)</u>}

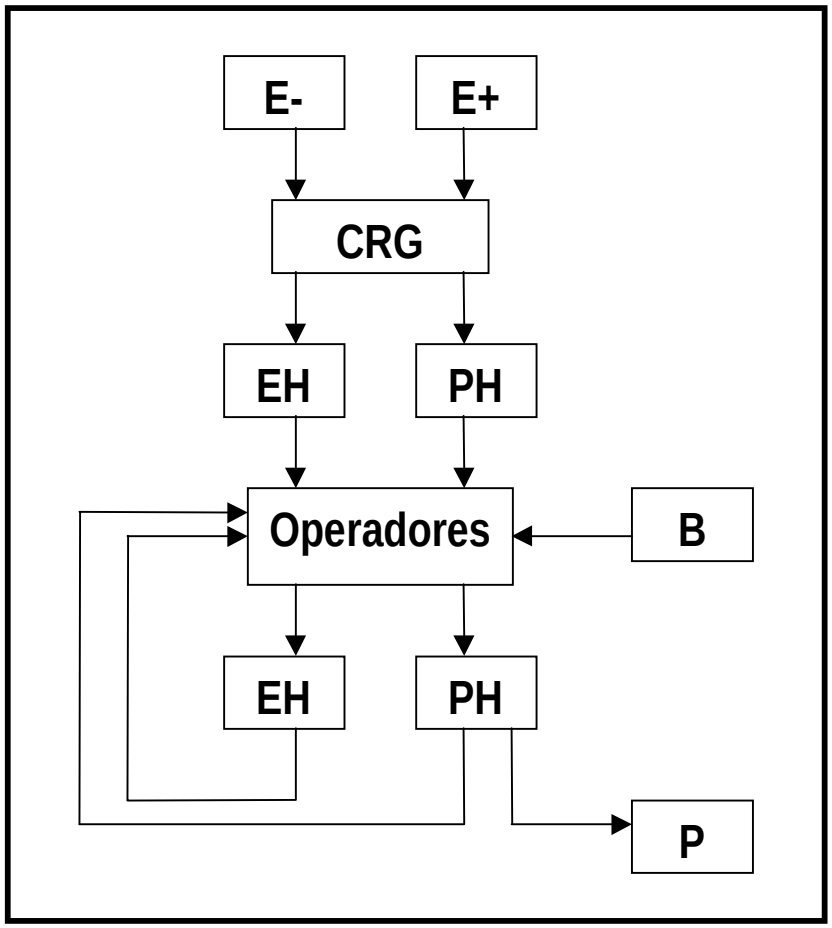
IFLP. El sistema FLIP.

El sistema FLIP (Functional Logic Inductive Programmer):

(Ferri, Hernandez & Ramírez 2000a)

El sistema FLIP es una implementación *pragmática* del esquema IFLP:

- ✓ Input: **E-, E+, B**
- ✓ Output: **P**



IFLP. El sistema FLIP.

Algunos Resultados del Sistema FLIP v.0.6

Program	Steps
sum (s(X),Y) = s (sum(X,Y)) sum (0,Y) = 0	2
app(: (X , Y) , Z) = : (app (X ,Z), Y) app([] , X) = X	2
prod(s(X),Y) = sum(prod(X,Y),Y) prod(0,X) = 0	2
fact(s(X)) = prod(fact (X) , s(X)) fact(0) = 0	2
rev(: (X,Y) = app (rev (X), : ([],Y)) rev([]) = []	3
sort(: (X,Y)) = inssort(Y, sort (X)) sort([]) = []	2

IFLP. El sistema FLIP.

FLIP v.0.7. Versión Incremental

(Ferri, Hernandez & Ramírez 2000b)

Para cada nuevo ejemplo *e* que se presenta:

If *e is a positive example:* check for every program $P_i \in PH$:

- HIT: ($P_i \models e$).
- NOVELTY: ($P_i \models \neq e \wedge \forall e' \in OE^+_n : P_i \cup \{e\} \models e'$)*:* Revision of PH and EH in order to cover the new evidence.
- ANOMALY: ($P_i \models \neg e$): Remove all non confluent and inconsistent P_i from *PH* and prune *EH*.

the CRG's of e are generated in EH and PH is extended with all the new unary programs.

If *e a negative example: check the consistency for every program* $P_i \in$ *PH and act as in either the HIT or as in the ANOMALY cases.*

IFLP. El sistema FLIP.

FLIP v.0.7. Resumen:

- Incremental
- No-interactivo
- Genético.
- Guiado por cobertura y compresión.

Mayores limitaciones actuales:

- Funciones con muchos atributos.
- Terminación (actualmente tratada con límite de pasos).

IFLP. Otros Sistemas

Algunos intentos de aprendizaje en Escher:

ESCHER: un lenguaje lógico-funcional basado en residuación.
Adaptación de métodos de aprendizaje de árboles:

- (Bowers, Giraud-Carrier & Lloyd 2000)
- (Kennedy 2000)

IFLP. Limitaciones de Representación

¿Qué pasa con el orden superior?

Aunque la mayoría de los lenguajes lógico-funcionales permiten orden superior, no ha habido demasiados trabajos sobre inducción en orden superior:

(Bensunsan, Giraud-Carrier, Kennedy 2000)

¿Qué pasa si datos complejos contienen una estructura de grafo?

- Los términos representan árboles...

Árboles de Decisión

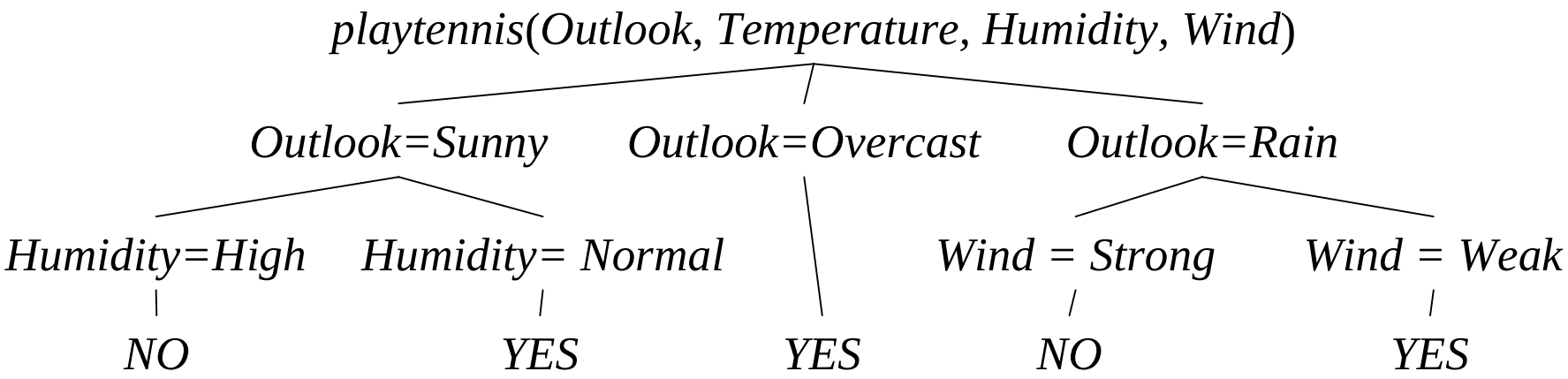
- Árboles de Decisión No Recursivos con nº de Clases Finito
  - Cualquier función computable f: $S_1 \times S_2 \times ... \times S_n \rightarrow T$, donde T es finito, se puede representar como una unión finita de m funciones parciales f_j: $S_1 \times S_2 \times ... \times S_n \rightarrow t_j$, donde $t_j \in T$ y $m \geq n$.
  - Por la propia definición de función, ninguna de estas funciones f_i puede ser no confluente con las otras.

- Definición de Árbol de Decisión con nº de Clases Finito :
  - El nodo raíz representa el origen abierto de la función $f(X_1, X_2, ..., X_n)$ donde X_i son variables.
  - Cada nodo no raíz y no hoja representa una condición.
  - Cada nodo hoja representa un elemento de T.
  - Cada rama representa una f_j.

Una misma función puede tener muchos árboles diferentes pero equivalentes.

Árboles de Decisión

- Árboles de Decisión No Recursivos con nº de Clases Finito
- EJEMPLO:



playtennis(sunny, Y, high, W) = NO
playtennis(sunny, Y, normal, W) = YES
playtennis(overcast, Y, Z, W) = YES
playtennis(rain, Y, Z, strong) = NO
playtennis(rain, Y, Z, weak) = YES

Árboles de Decisión

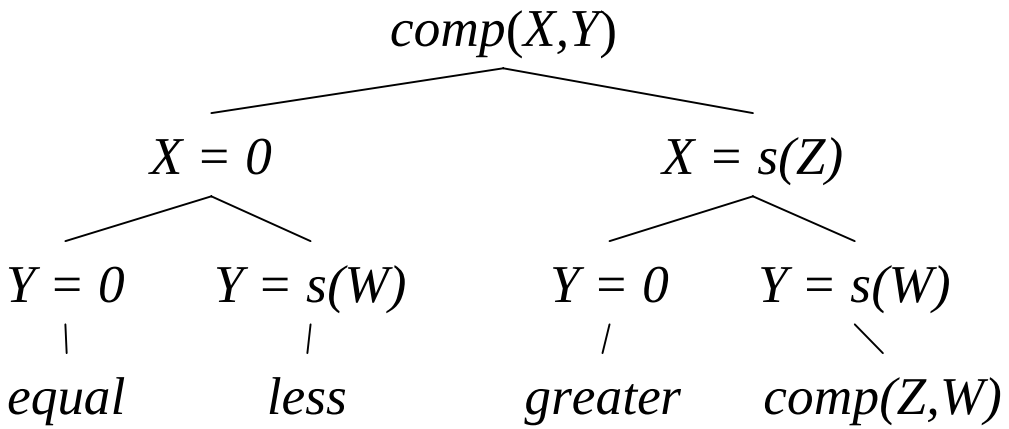
- Árboles de Decisión Recursivos con nº de Clases Finito
  - Cualquier función computable f: $S_1 \times S_2 \times ... \times S_n \rightarrow T$, donde T es finito, se puede representar como una unión finita de m funciones parciales f_j: $S_1 \times S_2 \times ... \times S_n \rightarrow t_j$, donde $t_j \in T$ y $m \geq n$.
  - Por la propia definición de función, ninguna de estas funciones f_i puede ser no confluente con las otras.

- Def. de Árbol de Decisión Recursivo con nº de Clases Finito:
  - El nodo raíz representa el origen abierto de la función $f(X_1, X_2, ..., X_n)$ donde X_i son variables.
  - Cada nodo no raíz y no hoja representa una condición.
  - Cada nodo hoja representa un elemento de T o una llamada recursiva a la propia función.

Una misma función puede tener muchos árboles diferentes pero equivalentes.

Árboles de Decisión

- Árboles de Decisión Recursivos con nº de Clases Finito
- EJEMPLO:



comp(0,0) = equal
comp(s(X),0) = greater
comp(0,s(X)) = less
comp(s(X),s(Y)) = comp(X,Y)

NO vale para todas las funciones recursivas.

Árboles de Decisión

- Árboles de Decisión Recursivos con nº de Clases Infinito

(o clase continua, también denominados árboles de regresión)

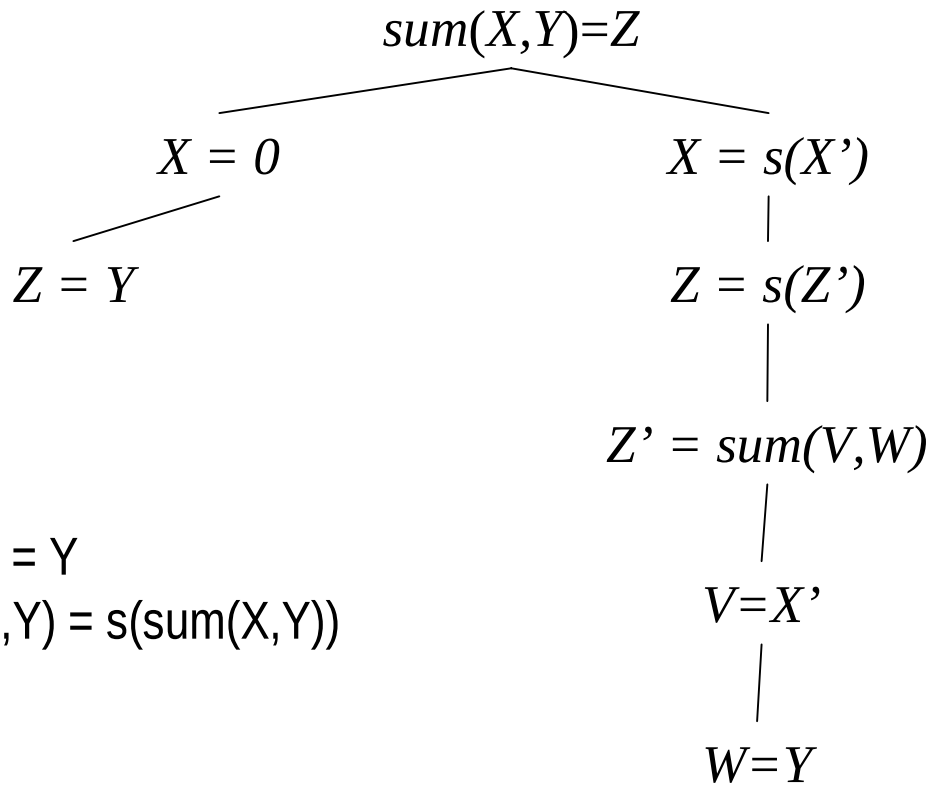
  - Cualquier función computable f: $S_1 \times S_2 \times ... \times S_n \rightarrow T$, se puede representar como una unión finita de m funciones parciales f_j: $S_1 \times S_2 \times ... \times S_n \rightarrow T$.
  - Por la propia definición de función, ninguna de estas funciones f_i puede ser no confluente con las otras.

- Def. de Árbol de Decisión Recursivo con nº de Clases Infinito:
  - El nodo raíz representa el la correspondencia universal $f(X_1, X_2, ..., X_n) = Y$, donde X_i y Y son variables.
  - Cada nodo no raíz representa una condición.

Una misma función puede tener muchos árboles diferentes pero equivalentes.

Árboles de Decisión

- Árboles de Decisión Recursivos con nº de Clases Infinito
- EJEMPLO:



sum(0,Y) = Y
sum(s(X),Y) = s(sum(X,Y))

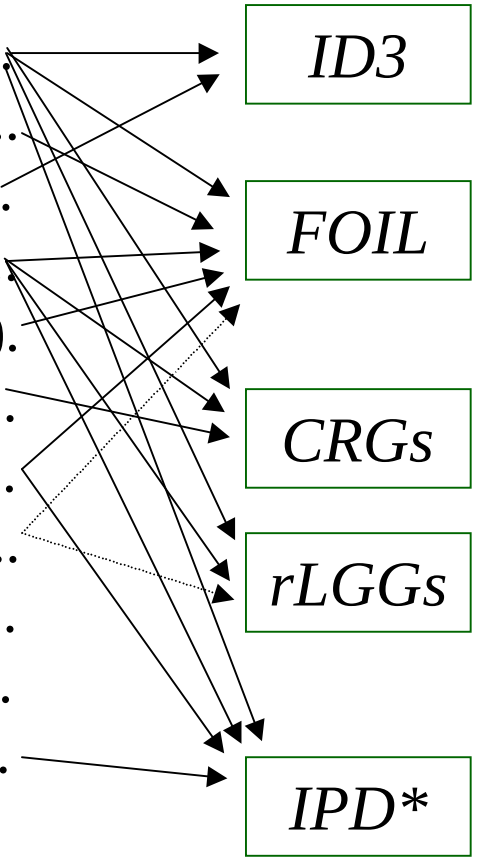
Árboles de Decisión

- Sistemas de atributos o proposicionales (ID3, C4.5, CART):
  - no permiten recursividad.
  - no permiten estructura en los atributos.
  - condiciones muy limitadas.
- Sistemas basados en ILP (FOIL, FFOIL, FOIDL, IPD):
  - Sólo pueden aprender funciones sobre clases finitas (además booleanas).
  - Cuando las clases son con más de 2 valores o infinitas, se añade como un argumento y se utilizan modos.
  - Los sistemas basados en árboles de decisión (p.ej. FOIL) no funcionan bien para estos problemas.
  - Han aparecido soluciones como FFOIL (Quinlan 1996) o FOIDL (Mooney and Califf 1995).
  - Se utilizan modos y el atributo de salida (+) se asume como la salida de una función.

Árboles de Decisión

TIPOS DE CONDICIONES:

- Uso de igualdades con constantes.....
- Uso de desigualdades (< >) con constantes.....
- Uso de comparaciones (<, >) con constantes.....
- Comparación de variables en positivo (X=Y).....
- Uso de negación (o operador < > con variables).
- Introducción de funciones (X = f(Y,Z,...))
- Introducción de funciones recursivas.....
- Pick & Mix libre.....
- Pick & Mix con uso de enlaces (CAj p.ej.).....
- Invención de funciones/predicados.....
- Generadas por algoritmos de aprendizaje.....



TIPOS DE RETORNO:

- Constante
- Con variable

* (Numao and Wakatsuki 1996)

Árboles de Decisión

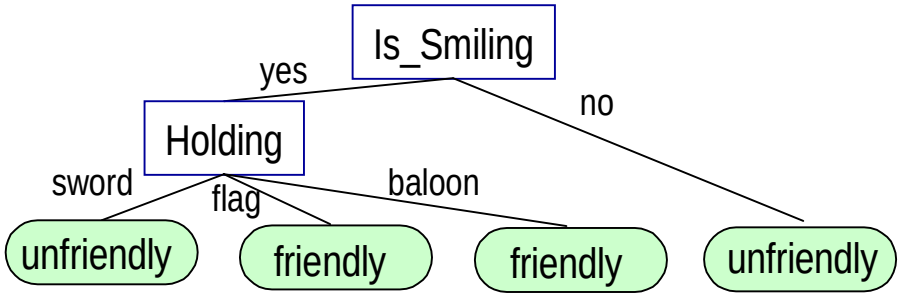
Uso de igualdad entre variables-atributos. Simplifica el árbol

EJEMPLO.
Aprender el concepto *friendly*.

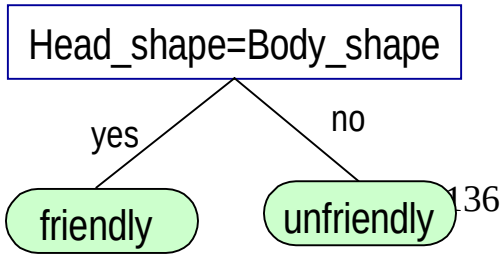
Evidencia $\Rightarrow$

Clase	Atributos y Valores				
	Is_Smiling	Holding	Has_tie	Head_shape	Body_shape
friendly	yes	balloon	yes	square	square
friendly	yes	flag	yes	octagon	octagon
unfriendly	yes	sword	yes	round	octagon
unfriendly	yes	sword	no	square	octagon
unfriendly	no	sword	no	octagon	round
unfriendly	no	flag	no	round	octagon

H (sin igualdad)
friendly if Is_Smiling=yes ^ Holding=balloon
friendly if Is_Smiling=yes ^ Holding=flag
unfriendly if Is_Smiling=no
unfriendly if Is_Smiling=yes ^ Holding=sword



H (con igualdad)
friendly if Head_shape = Body_shape
unfriendly if Head_shape ≠ Body_shape



Árboles de Decisión

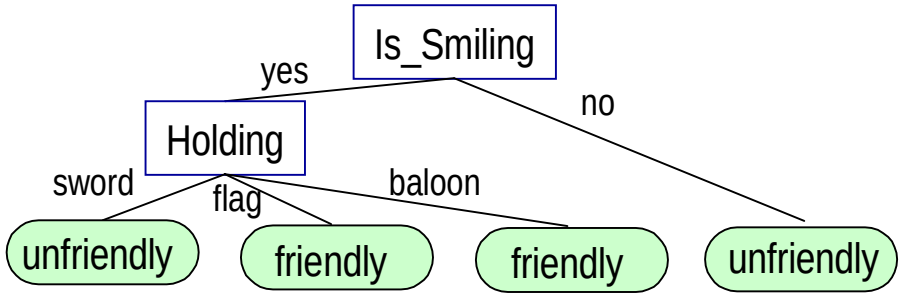
Uso de negación (o operador < >). Simplifica el árbol

EJEMPLO.
Aprender el concepto *friendly*.

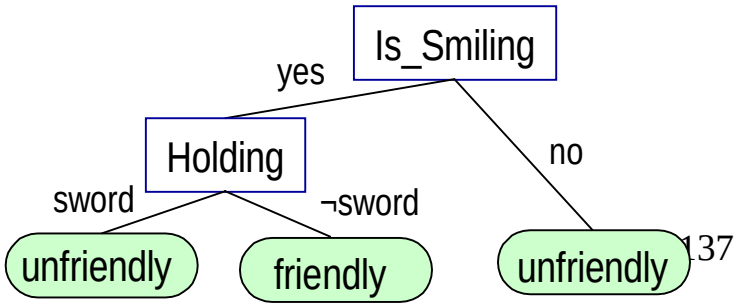
Evidencia $\Rightarrow$

Clase	Atributos y Valores				
	Is_Smiling	Holding	Has_tie	Head_shape	Body_shape
friendly	yes	balloon	yes	square	square
friendly	yes	flag	yes	octagon	octagon
unfriendly	yes	sword	yes	round	octagon
unfriendly	yes	sword	no	square	octagon
unfriendly	no	sword	no	octagon	round
unfriendly	no	flag	no	round	octagon

H (sin negación)
friendly if Is_Smiling=yes ^ Holding=balloon
friendly if Is_Smiling=yes ^ Holding=flag
unfriendly if Is_Smiling=no
unfriendly if Is_Smiling=yes ^ Holding=sword



H (con negación)
friendly if Is_Smiling=yes ^ Holding<>sword
unfriendly if Is_Smiling=no
unfriendly if Is_Smiling=yes ^ Holding=sword



Árboles de Decisión

Uso de otras condiciones *necesarias para muchos problemas*:

- Si los ejemplos tienen estructura ⇒ Introducción de funciones (X = f(Y,Z,...))
- Si la función es recursiva ⇒ Introducción de funciones recursivas.
- Si la función es relacional o necesita funciones auxiliares presentes en B ⇒ Pick & Mix
- Si la función necesita funciones auxiliares no presentes en B ⇒ Invención de funciones/predicados
- Si hay particiones lineales no unitarias o no lineales sobre atributos continuos ⇒ Generadas por algoritmos de aprendizaje.
  - P.ej. el IPD permite crear particiones sugeridas por un perceptron: 2X + Y > 3.

Árboles de Decisión

Construcción:

- SPLITTING ALGORITHM (ID3, C4.5, CART, ASSISTANT, IPD)
  - Llamado también DIVIDE-AND-CONQUER
  - Asume las ramas son disjuntas (no solapan).

- COVERING ALGORITHM (AQ, CN2, FOIL)
  - Llamado también SEPARATE-AND-CONQUER

Árboles de Decisión

SPLITTING ALGORITHM (ID3, C4.5, CART, ASSISTANT)

con sólo dos clases

```
Algorithm CART(Target_function, Pos, Neg)
    node = f(X1, X2, ..., Xn)
    split_and_conquer(∅, &node, Pos, Neg);
    return node;
endalgorithm

Function split_and_conquer(condition, node, Pos, Neg);
    if Neg <> ∅ then      // Añadir condiciones para especializar la regla
       Candidate_Split := generar posibles condiciones
       Best_Split := seleccionar la mejor condición según el criterio de ganancia.
       Link Node with every node of Best_Split (as children)
       for each node ∈ Best_Split do
          cond:= condition of node;
          Pos:= Pos which are true under cond.    Neg:= Neg which are true under cond.
          split_and_conquer(cond, &node, Pos, Neg);
       endfor;
    endif
    return node;
endfunction
```

Árboles de Decisión

COVERING ALGORITHM (AQ, CN2, FOIL, FFOIL)

con sólo dos clases

FOIL(Target_predicate, Predicates, Pos, Neg)
Learned_rules := ∅
while Pos <> ∅ **do** // Aprender una nueva regla
    NewRule :- el target_predicate con todo variables y sin condiciones.
    NewRuleNeg :- Neg;
    while NewRuleNeg <> ∅ **do** // Añadir condiciones para especializar la regla
        Candidate_Literals :- generar posibles condiciones (utilizando el cjto. Predicates)
        Best_Literal :- seleccionar el mejor literal utilizando el criterio de ganancia.
        Añadir este Best_Literal a las condiciones de NewRule;
        Quitar a NewRuleNeg los ej. neg. que ya no son cubiertos (incumplen Best_Literal)
    endwhile
    Learned_rules :- Learned_rules ∪ { NewRule}
    Pos :- Pos - {miembros de Pos cubiertos por la nueva regla).
endwhile;
Retornar Learned_rules;

Árboles de Decisión

SPLITTING vs COVERING

Ventajas del SPLITTING:

- Aprovecha caminos ya abiertos (más eficiente)
- Poda más sencilla.
- Representación generalmente más corta y eficiente. Se pueden utilizar DECISION LISTS (uso del cut) en vez de DECISION TREES.

Ventajas del COVERING:

- Permite hacer coberturas no totales (interesante cuando un atributo tiene muchos valores y sólo algunos son significativos).
- Es menos voraz y las últimas ramas se ajustan más a los ejemplos. Esto también es una desventaja, porque las últimas ramas suelen hacer overfitting.

Árboles de Decisión

Funciones de Ganancia:

ID3, C4.5 (y ≅FOIL) (Quinlan 1993):

C: nº de clases,

$p(E,j)$: proporción de los casos de E de la clase j

T: test (split) considerado.

E_i: evidencia que cumple cada uno de los k resultados del test.

$$\text{Info}(E) = -\sum_{j=1}^{C} p(E,j) \times \log_2(p(E,j))$$

$$Gain(E,T) = Info(E) - \sum_{i=1}^{k} \frac{|E_i|}{|E|} \times Info(E_i)$$

$$Split(E,T) = -\sum_{i=1}^{k} \frac{|E_i|}{|E|} \times \log_2\left(\frac{|E_i|}{|E|}\right)$$

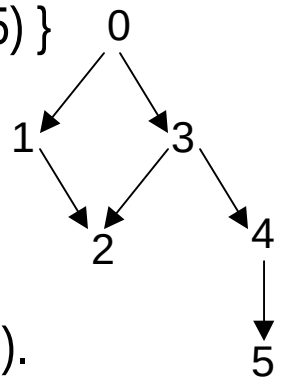
$$GainRatio(E,T) = \frac{Gain(E,T)}{Split(E,T)}$$

De las particiones (splits) con al menos GAIN medio, se selecciona el que tenga el GAINRATIO mayor.

Árboles de Decisión

Ejemplo: Linked con sólo 6 nodos y con GainRatio ID3:

B = { linked(0,1), linked(0,3), linked(1,2), linked(3,2), linked(3,4), linked(4,5) }



E+= { reach(0,1). reach(0,2). reach(0,3). reach(0,4). reach(0,5).
reach(1,2). reach(3,2). reach(3,4). reach(3,5). reach(4,5).}

E- (el resto de combinaciones de los 6 nodos, total 26 ejemplos negativos).

	True	False	Total	
No. Examples	10	26	36	
P(E,Ci)	0,2778	0,7222	1	INFO: 0,8524

TESTS on can_reach(X,Y)										
		C1	C2	TOTAL	P(E,Econd)		P(Econd,C1)	P(Econd,C2)	Info	P(E,Econd) *
t1	X= 0	5	1	6	0,16666667		0,83333333	0,16666667	0,65002242	0,10833707
t2	X<>0	5	25	30	0,83333333		0,16666667	0,83333333	0,65002242	0,54168535
			Total	36	1					
	…									
t13	Y= 0	0	6	6	0,16666667		0	1	0	0
t14	Y<>0	10	20	30	0,83333333		0,33333333	0,66666667	0,91829583	0,76524653
			Total	36	1					
	…									
t25	X=Y	0	6	6	0,16666667		0	1	0	0
t26	X<>Y	10	20	30	0,83333333		0,33333333	0,66666667	0,91829583	0,76524653
			Total	36	1					
t27	linked_to(X,Z)	10	14	24	0,66666667		0,41666667	0,58333333	0,97986876	0,65324584
t28	¬linked_to(X,Z)	0	12	12	0,33333333		0	1	0	0
			Total	36	1					
t29	linked_to(Z,X)	5	20	25	0,69444444		0,2	0,8	0,72192809	0,50133895
t30	¬linked_to(Z,X)	5	6	11	0,30555556		0,45454545	0,54545455	0,99403021	0,30373145
			Total	36	1					
t31	linked_to(Y,Z)	4	12	16	0,44444444		0,25	0,75	0,81127812	0,36056806
t32	¬linked_to(Y,Z)	6	14	20	0,55555556		0,3	0,7	0,8812909	0,48960606
			Total	36	1					
t33	linked_to(Z,Y)	10	20	30	0,83333333		0,33333333	0,66666667	0,91829583	0,76524653
t34	¬linked_to(Z,Y)	0	6	6	0,16666667		0	1	0	0
			Total	36	1					

SPLITS			Gain	Split	Gain Ratio
S1={t1,t2}			0,2024	0,65002242	0,31134735
…					
S8={t13,t14}			0,0872	0,65002242	0,13408561
…					
S15={t25,t26}			0,0872	0,65002242	0,13408561
S16={t27,t28}			0,1992	0,91829583	0,21687928
S17={t29,t30}			0,0473	0,88797632	0,05330634
S18={t31,t32}			0,0022	0,99107606	0,00225116
S19={t33,t34}			0,0872	0,65002242	0,13408561

Árboles de Decisión

Funciones de Ganancia:

Ganancia Mejorada para Valores continuos:

C4.5 (Quinlan 1996)

$$AdjGain(E,T) = Gain(E,T) \times \frac{\log_2(N-1)}{|E|} \text{ for continuous splits}$$

$$AdjGain(E,T) = Gain(E,T) \text{ for non-continuous splits}$$

$$AdjGainRatio(E,T) = \frac{AdjGain(E,T)}{Split(E,T)}$$

Árboles de Decisión

Funciones de Ganancia:

CART (Breiman et al. 1984):

$$\text{Gini}(E) = \left(\frac{|P|}{|E|}\right)\left(\frac{|N|}{|E|}\right) = \left(\frac{|P||N|}{|E|^2}\right)$$

$$Gain(E, cond) = Gini(E) - [(r(Gini(E_{neg}) + (1-r)Gini(E_{pos}))]$$

$$with \;\; r = \frac{|E_{neg}|}{|E|}$$

Otras: Ortogonalidad de GID3 (Fayyad 1994)

Árboles de Decisión

Incorporación de Recursividad:

Chequeo extensional. FOIL o FFOIL:

- Cuando se intenta introducir una llamada recursiva en el árbol, su valor de verdad se extrae de los ejemplos positivos.
- Esto causa problemas, p.ej.
  - member(X,[Y|Z]) :- member(X,Z)
  - no cubriría el ejemplo member(a, [c,b,a]) con respecto a los otros si member(a, [b,a]) no estuviera en los ejemplos.

Chequeo intensional. FOIDL:

- Cuando se intenta introducir una llamada recursiva en el árbol, su valor de verdad se extrae de los ejemplos positivos y de llamadas a la propia función.
  - En este caso bastaría con tener member(a, [a])

También hay sistemas que permiten utilizar las ramas cerradas como verdaderas y otros no. En este caso sería suficiente unos cuantos member(x, [x]) que se hubieran generalizado.

Árboles de Decisión

Incorporación de Recursividad. Chequeo de Terminación

Aproximación a terminación. (FOIL o FFOIL):

- Excluye algunas soluciones interesantes.

Límite de pasos en la comprobación de cláusulas recursivas. (FOIDL):

- Ralentiza los algoritmos. Se suele hacer que el límite de pasos dependa del tamaño del ejemplo a probar.

Árboles de Decisión

Evitar Mínimos Locales:

Los métodos de construcción de árboles de decisión suelen ser muy voraces y se suelen quedar estancados en límites locales.

Una manera de evitarlo es introducir condiciones (predicados) que introduzcan nuevas variables.

- FOIL lo hace cuando ningún split tiene ganancia positiva (con la esperanza de que esta jugada hará que más abajo se consigan mayores ganancias).
- Generalmente se introducen literales determinados por una variable X, como por ejemplo un ins(Z,Y,X), sobre esa variable.

Árboles de Decisión

PODA DE ÁRBOLES:

Una vez realizado el árbol, muchos sistemas pospodan (postpruning) el árbol, con el objetivo de obtener árboles más cortos (con menos condiciones) y, por tanto, más generales.

- Poda de árboles por redundancia: (pueden existir condiciones superfluas, debido a la manera como se construye el árbol)
- Poda de árboles por ruido o principio MDL: (se poda porque se supone que algunos nodos han sido introducidos porque puede haber ruido).

Árboles de Decisión

PODA DE ÁRBOLES:

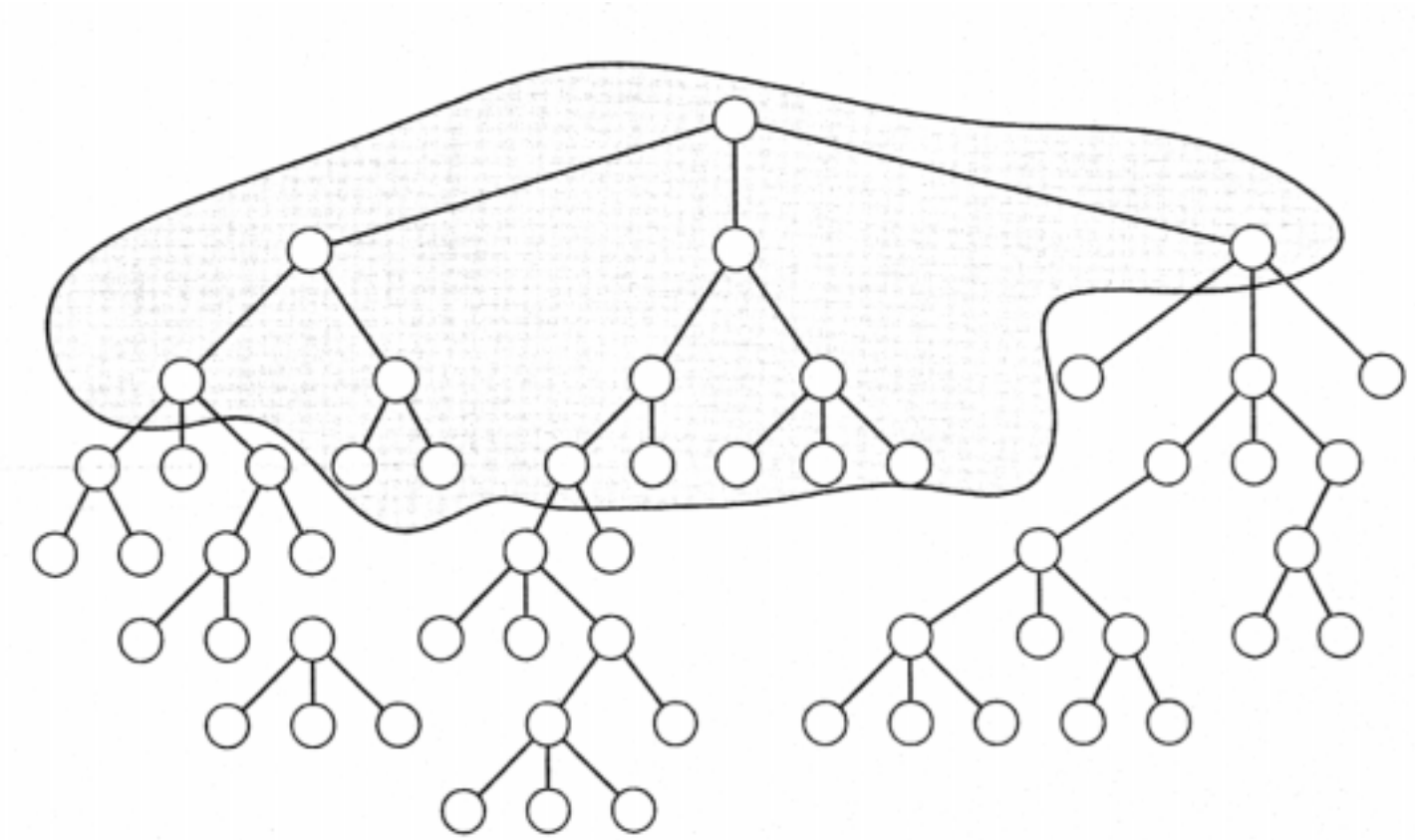


Figure 11.10 The challenge of pruning is to find the best subtree.

Árboles de Decisión

Poda de árboles por redundancia:

Ejemplo:

Clase	Atributos y Valores				
	Is_Smiling	Holding	Has_tie	Head_shape	Body_shape
friendly	yes	balloon	yes	square	square
friendly	yes	flag	yes	octagon	octagon
unfriendly	yes	sword	yes	round	octagon
unfriendly	yes	sword	no	square	octagon
unfriendly	no	sword	no	octagon	round
unfriendly	no	flag	no	round	octagon

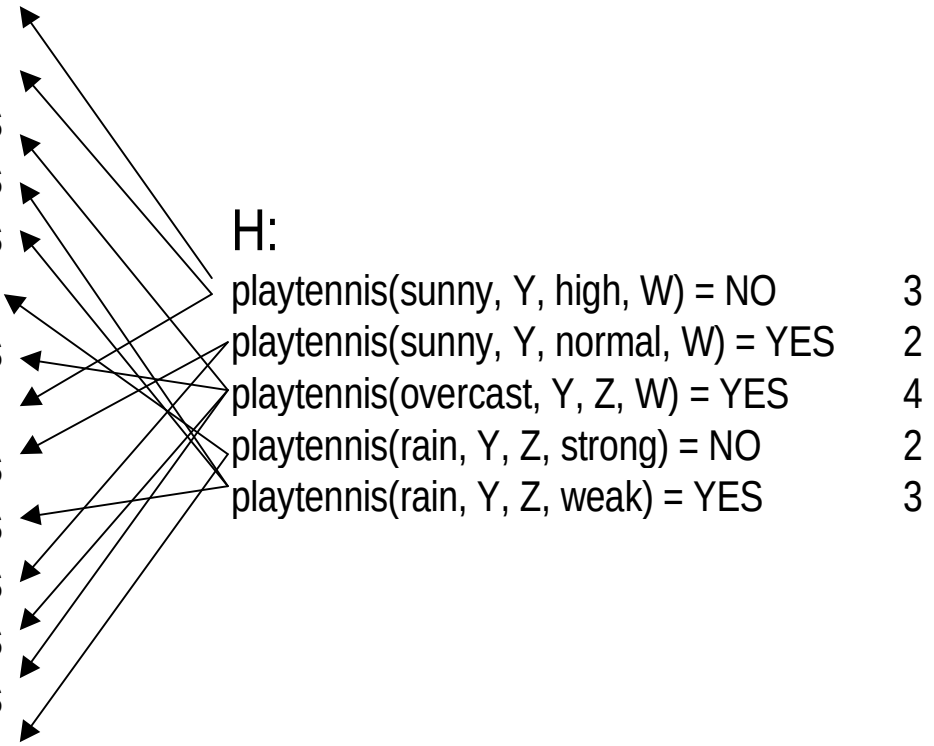
friendly if Is_Smiling=yes ^ Holding<>sword
unfriendly if Is_Smiling=no
unfriendly if Is_Smiling=yes ^ Holding=sword

unfriendly if Is_Smiling=no
unfriendly if Holding=sword

Árboles de Decisión

Poda de árboles por ruido o principio MDL:

Ex.	Sky	Temp	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overc.	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overc.	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overc.	Mild	High	Strong	Yes
13	Overc.	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No



Árboles de Decisión

Poda de árboles por ruido o principio MDL:

Ex.	Sky	Temp	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overc.	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overc.	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	**Sunny**	**Cool**	**Normal**	**Weak**	**No**
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overc.	Mild	High	Strong	Yes
13	Overc.	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

ruido

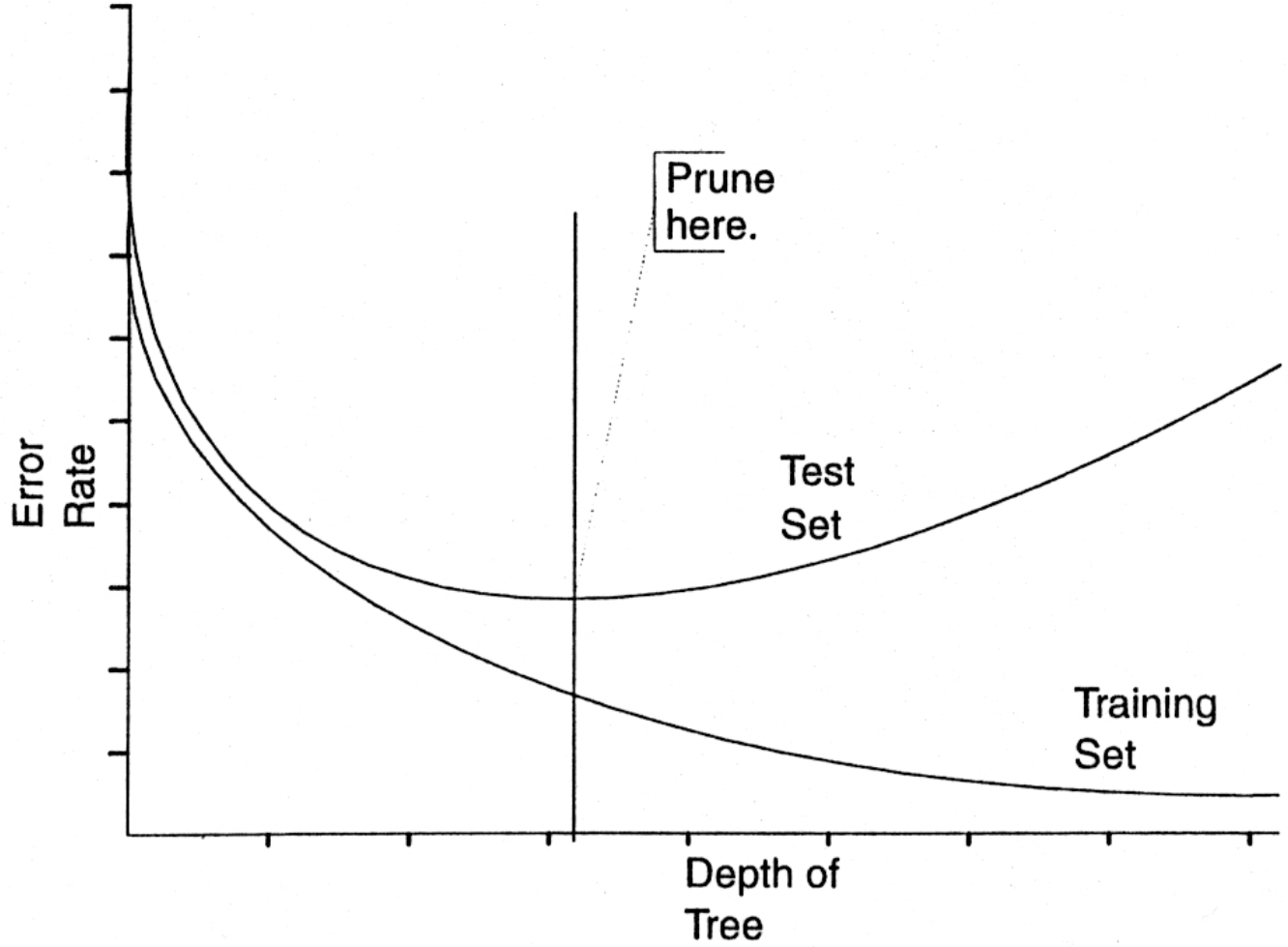
H:

	nº de ejs.
playt(sunny, Y, high, W) = NO	3
playt(sunny, mild, normal, W) = YES	1
playt(overcast, Y, Z, W) = YES	4
playt(rain, Y, Z, strong) = NO	2
playt(rain, Y, Z, weak) = YES	3
playt(sunny, cool, normal, W) = NO	1

Es preferible podar esas dos reglas con poca cobertura (1).

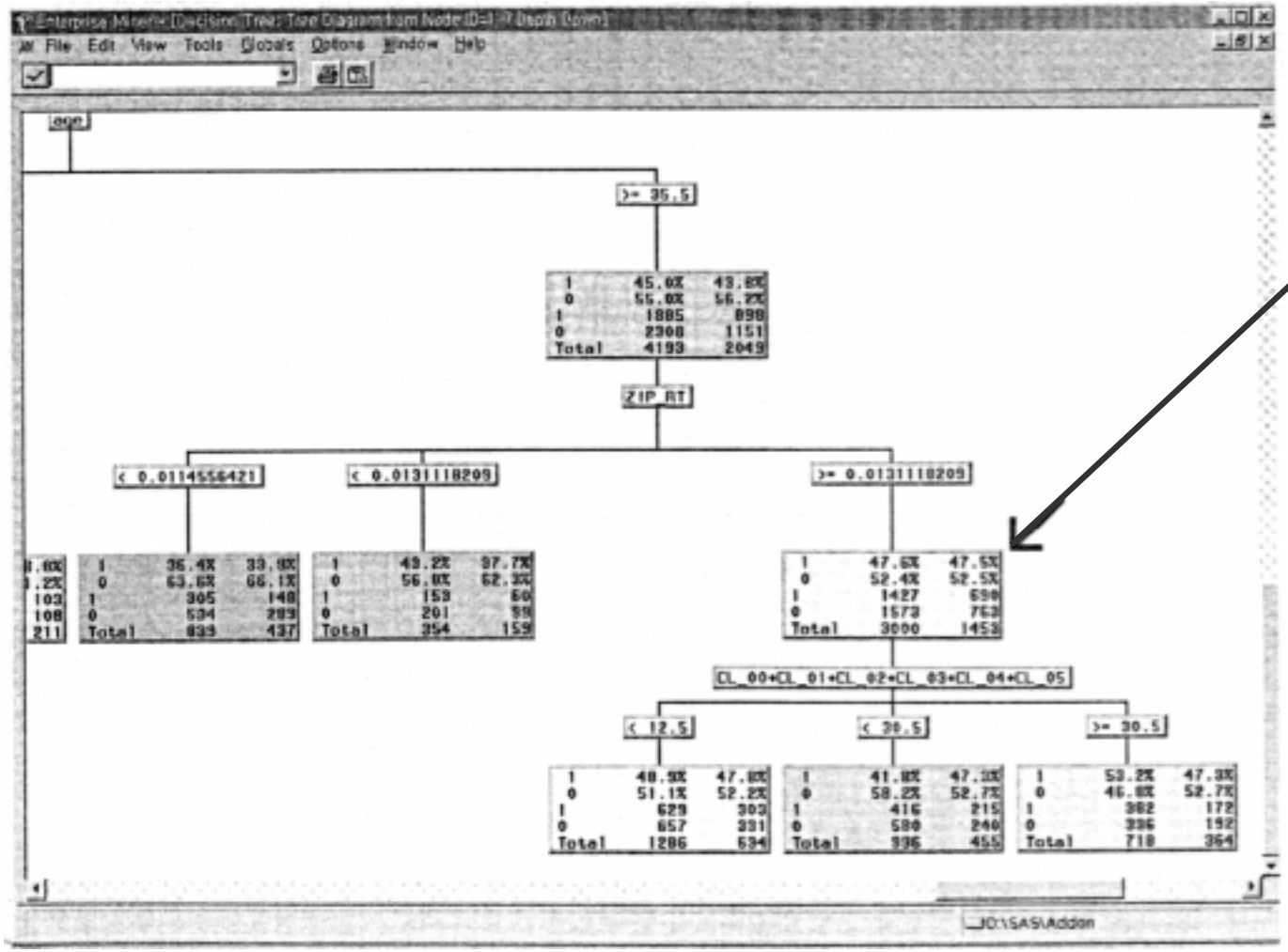
Árboles de Decisión

Poda de árboles utilizando validación cruzada:



Árboles de Decisión

Poda de árboles utilizando validación cruzada:



Para ver qué hojas podar se examina la diferencia de cada subárbol en el comportamiento con el training y test set:

Árboles de Decisión

Reestructuración de árboles:

Debido al carácter voraz de los algoritmos de construcción de árboles de decisión es posible que algunas condiciones sean evaluadas peor por los criterios y se pospongan hasta más abajo.

Los métodos de reestructuración de árboles permiten reestructurar el árbol con el objetivo de simplificar la representación y/o conseguir mayor predictividad.

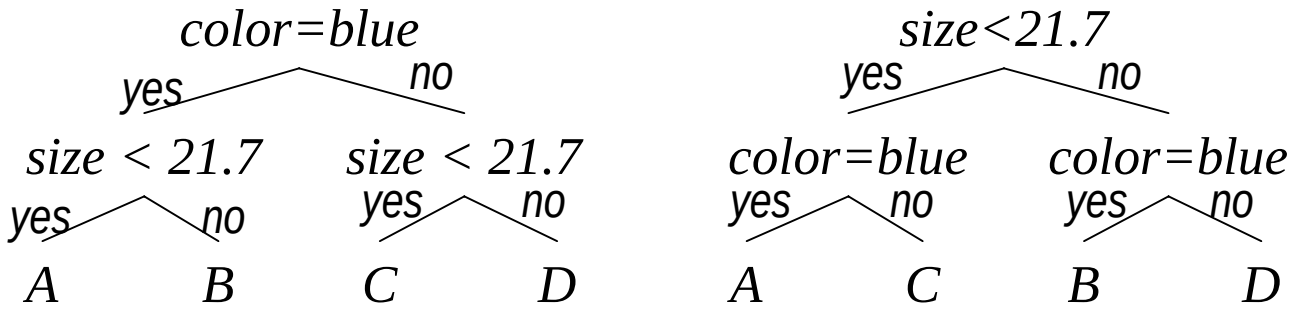
Es necesario cuando los árboles se construyen de manera incremental $\Rightarrow$ REVISIÓN DE ÁRBOLES.

Árboles de Decisión

Reestructuración de árboles:

Basados en operadores (Utgoff, Berkman, Clouse 1997):

- Operador de Transposición:



- Transposición Recursiva.
- Poda Virtual.

Resumen

TÉCNICAS DE APRENDIZAJE AUTOMÁTICO VISTAS:

- Interpolación y Predicción Secuencial (regresión lineal, no lineal, local).
- Supervisado
  - Similaridad (k-NN Nearest Neighbor).
  - Fence & Fill (k-means, Perceptron Learning, ANN, RBF, Decision Trees, Bayes Classifiers, Center Splitting)
  - Pseudo-relational
  - Relational: ILP, IFLP, SCIL.
- No Supervisado

- *Representación como árboles de decisión*

¿Por qué Aprendizaje Automático?

Según MITCHELL:

- Gran progreso reciente.
- Gran aumento de datos en línea.
- Gran progreso en poder computacional en mini y PC.
- Interés por la industria (esp. gestión empresarial y control).

- Tres nichos importantes:
  - Data and Web Mining.
  - Aplicaciones Software.
  - Programas que se personalizan al usuario.

Es lo que vamos a ver en los T2 - T4

La Ciencia y la Industria se basan en conocimiento ⇒ Haremos énfasis en técnicas que produzcan conocimiento.

Recursos Web sobre ML

URLs:

- MLNET: http://www.mlnet.org/
- Datasets: http://www.mlnet.org/cgi-bin/mlnetois.pl/?File=datasets.html
- ML software: http://mlwww.diee.unica.it/ML/gdl/mlsoftware.html
- ILPNET2: http://www-ai.ijs.si/~ilpnet2/
- ILP systems: http://www-ai.ijs.si/~ilpnet2/systems/
- UCI ML datasets repository: http://www.ics.uci.edu/~mlearn/MLRepository.html

Listas de correo: machine-learning@egroups.com

To UNSUBSCRIBE, send an empty message to
machine-learning-unsubscribe@egroups.com
To SUBSCRIBE, send an empty message to
machine-learning-subscribe@egroups.com
Archives: http://www.egroups.com/group/machine-learning

Referencias del Tema 1b

(Bensunsan et al. 2000) Bensusan, H.; Giraud-Carrier, C. and Kennedy, C. “A higher-order approach to meta-learning” In Proceedings of the ECML'2000 workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination, pages 109--117. ECML'2000, June 2000.

(Blockeel & De Raedt 1997) Blockeel H.; De Raedt. L “Lookahead and Discretization in ILP”. In S. Dzeroski and N. Lavrac, editors, Proceedings of the 7th International Workshop on Inductive Logic Programming, volume 1297 of Lecture Notes in Artificial Intelligence, pages 77--84. Springer-Verlag, 1997.

(Blockeel and De Raedt 1998) Blockeel, H. and De Raedt, L. “Top-down Induction of First-order Logical Decision Trees”Artificial Intelligence(101)1-2, pp. 285-297, 1998.

(Boström 1995) Boström, H. “Covering vs. divide-and-conquer for top-down induction of logic programs” in Proc. of the 14th Intl. Conf. on Artificial Intelligence (IJCAI-95), pp. 1194-1200, Morgan Kaufmann, San Mateo, CA, 1995.

(Boström and Idestam-Almquist 1994) Boström, H.; Idestam-Almquist, P. “Specialization of logic programs by pruning SLD-trees”, in Wrobel, S. (ed.) Proc. of the 4th Intl. Workshop on Inductive Logic Programming (ILP94), Vol. 237 of GMD-Studien. Bad Honnef/Bonn 1994. Gesellschaft für Mathematik und Dateverarbeitung.

(Bowers et al. 2000) Bowers, A. F.; Giraud-Carrier, C.; and Lloyd. J. W. “Classification of Individuals with Complex Structure” In Proceedings of the Seventeenth International Conference on Machine Learning (ICML'2000), pages 81--88. Morgan Kaufmann, June 2000.

(Breiman et al 1984) Breiman, L.; Friedman, J.H.; Olshen, R.A.; Stone, C.J. “Classification and Regression Trees” Wadsworth and Brooks/Cole, Monterey, 1984.

Referencias del Tema 1b

(Brunk & Pazzani 1991) Brunk, C.A. and Pazzani, M.J. “An Investigation of Noise-Tolerant Relational Concept Learning Algorithms”. In L. Birnbaum and G. Collins, editors, Proceedings of the 8th International Workshop on Machine Learning, pages 389--393. Morgan Kaufmann, 1991.

(Cameron-Jones & Quinlan 1993) Cameron-Jones R.M.; Quinlan, J.R. “Avoiding Pitfalls When Learning Recursive Theories”. In R. Bajcsy, editor, Proceedings of the 13th International Joint Conference on Artificial Intelligence, pages 1050--1057. Morgan Kaufmann, 1993.

(Cameron-Jones & Quinlan 1994) Cameron-Jones R.M.; Quinlan, J.R.“Efficient Top-down Induction of Logic Programs”. SIGART Bulletin, 5(1):33--42, 1994.

(Cheeseman & Stutz 1996) Chesseman, P.; Stutz, J. “Bayesian Classification (AutoClass). Theory and Results”, chap. 6 of Fayyad, U.M.; Piatetsky-Shapiro, G.; Smyth, P. *Advances in Knowledge Discovery and Data Mining*, AAAI Press/The MIT Press, 1996.

(Conklin and Witten 1994) Conklin, D.; Witten, I. H. “Complexity-Based Induction” *Machine Learning*, 16, 203-225, 1994.

(De Raedt & Bruynooghe 1988) De Raedt, L. and Bruynooghe, M. “On interactive concept-learning and assimilation” in D. Sleeman, editor, Proceedings of the 3rd European Working Session on Learning. Pitman, 1988.

De Raedt & Bruynooghe 1993. L. De Raedt and M. Bruynooghe. A Theory of Clausal Discovery. In R. Bajcsy, editor, Proceedings of the 13th International Joint Conference on Artificial Intelligence, pages 1058--1063. Morgan Kaufmann, 1993. Also in S. Muggleton, editor, Proceedings of the 3rd International Workshop on Inductive Logic Programming, pages 25--40. J. Stefan Institute, 1993.

(Dzeroski 1991) Dzeroski. S. *Handling noise in inductive logic programming*. Master's thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, 1991.

Referencias del Tema 1b

(Ferri, Hernandez & Ramírez 2000a) Ferri Ramírez, C.; Hernández Orallo, J. Ramírez Quintana, M.J. "FLIP: User's Manual", Dpto. de Sistemas Informáticos y Computación, Valencia, TR: (II-DSIC-24/00), 2000

(Ferri, Hernandez & Ramírez 2000b Ferri Ramírez, C.; Hernández Orallo, J. Ramírez Quintana, M.J. "Learning functional logic classification concepts from databases" in María Alpuente (Ed.) "Functional and Logic Programming", 9th International Workshop on Functional and Logic Programming (WFLP'2000) pp. 296-308., 2000.

(Flener & Yilmaz 1999) Flener, P.; Yilmaz, S. "Indcutive Synthesi of Recursive Logic Programs: Achievements and Prospectes" *Journal of Logic Programming*, 1999.

(Friedman 1991) Friedman, J. "Multivariate adaptive regression splines (with discussion)" Annals of Statistics, 19(1), 1-141.

(Helft 1987) Helft. N. "Induction as nonmonotonic inference". In Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning, pages 149--156. Morgan Kaufmann, 1989.

(Hernández 2000b) Hernández Orallo, J. "Computational Measures of Information Gain and Reinforcement in Inference Process" *AI Communications*, Vol.13, nº 1, 49-50, IOS Press, Amsterdam, 2000.

(Hernández & Ramírez 1998) Hernández Orallo, J.; Ramírez Quintana, M.J. "*Inverse Narrowing for the Induction of Functional Logic Programs*" in Freire-Nistal, J.L.; Falaschi, M.; Vilares-Ferro, M. (eds) Proceedings of the 1998 Joint Conference of Declarative Programming, APPIA-GULP-PRODE'98, pp. 379-392, 1998

(Hernández & Ramírez 1999) Hernández Orallo, J. Ramírez-Quintana, M.J. "A Strong Complete Schema for Inductive Functional Logic Programming" in Dzeroski, S.; Flach, P. (eds) "Inductive Logic Programming" Lecture Notes in Artificial Intelligence (LNAI) series, 1634, p.116-127, Springer 1999.

Referencias del Tema 1b

(Ichise 1998) Ichise, R.: 1998, `Synthesizing Inductive Logic Programming and Genetic Programming'.In: H. Prade (ed.): Proceedings of the 13th European Conference on AI, ECAI'98. pp. 465-468.

(Idestam-Almquist 1992) Idestam-Almquist, P. “Learning missing clauses by inverse resolution” in Proc. of the Intl. Conf. on Generation Computer Systems. Ohmsha, Tokyo, 1992.

(Idestam-Almquist 1993a) Idestam-Almquist, P. *Generalization of Clauses*, PhD. Thesis, Stockholm University 1993.

(Idestam-Almquist 1993b) Idestam-Almquist, P. “Generalization under implication by using or-introduction” in Brazdil, P.B. (ed.) Proc. of the 6th European Conference on Machine Learning (ECML-93) Vol. 667 of Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, 1993.

(Idestam-Almquist 1993c) Idestam-Almquist, P. “Generalization under implication: Expansion of clauses for indirect roots” in Proc. of the 4th Scandinavian Conference on Artificial Intelligence, IOS Press, Amsterdam, 1993.

(Idestam-Almquist 1995) Idestam-Almquist, P. “Generalization of clauses under implication” *Jouranl of ARtificial Intelligence Reserach*, 3: 467-489, 1995.

(Karalic 1995) Karalic A. *First order regression*. PhD thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia, 1995.

(Kietz and Wrobel 1992) Kietaz, J.U.; Wrobel, S. “Controlling the complexity of learning in logic through syntactic and task-oriented models” in Muggleton S. (ed) *Inductive logic programming, Vol. 32* of APIC Series, Academic Press, London, pag 335-359.

(Kohavi et al. 1994) Kohavi, R.; John, G.; Long, R.; Manley, D.; and Pfleger, K. “MLC++: A machine learning library in C++” in Porc. Tools with Artificial Intellgience, 740-743, IEEE Computer Society Press, 19994.

Referencias del Tema 1b

(Lavrac et al. 1991) Lavrac, N.; Dzeroski, S. and Grobelnik. M. “Learning Nonrecursive Definitions of Relations with LINUS”. In Y. Kodratoff, editor, Proceedings of the 5th European Working Session on Learning, volume 482 of Lecture Notes in Artificial Intelligence, pages 265--281. Springer-Verlag, 1991.

(Lee 1967) Lee, R.C.T. *A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms*, PhD thesis, University of California, Berkeley, 1967.

(Kennedy 2000) C J Kennedy. *Strongly Typed Evolutionary Programming.* PhD thesis, Department of Computer Science, University of Bristol, January 2000.

(Mooney and Califf 1995) Mooney, R.J.; Califf, M.E. “Induction of first-order decision lists: Results on learning the past tense of English verbs” *Journal of Artificial Intelligence Research*, 3, 1-24, 1995.

(Morik et al. 1993) Morik, K.; Wrobel, S.; Kietz, J-U. and Emde. W. *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications*. Academic Press, 1993.

(Mozetic 1987) Mozetic I.; Lavrac, N. “Incremental learning from examples in a logic-based formalism”. In P. Brazdil, editor, Proceedings of the Workshop on Machine Learning, Meta-Reasoning and Logics, pages 109-127, 1988.

(Muggleton 1987) Muggleton, S. “Duce, an oracle based approach to constructive induction”. In Proceedings of the 10th International Joint Conference on Artificial Intelligence, pages 287--292. Morgan Kaufmann, 1987.

(Muggleton and Buntine 1988) Muggleton, S. and Buntine, W. “Machine invention of first order predicates by inverting resolution”. In Proceedings of the 5th International Workshop on Machine Learning, pages 339--351. Morgan Kaufmann, 1988.

(Muggleton and Feng 1992) Muggleton S.; Feng. C. “Efficient Induction in Logic Programs”. In S. Muggleton, editor, Inductive Logic Programming, pages 281--298. Academic Press, 1992.

Referencias del Tema 1b

(Muggleton 1995) Muggleton. S. “Inverse entailment and Progol”. *New Generation Computing*, 13:245-286, 1995.

(Muggleton 1999) S. Muggleton. “Learning from positive data”. *Machine Learning*, accepted.

(Muggleton 1999a) Muggleton, S. “Inductive Logic Programming: issues, results and the LLL challenge”, *Artificial Intelligence*, 114(1):283-296, 1999.

(Muggleton 1999b) Muggleton, S. “Scientific Knowledge Discovery using Inductive Logic Programming”, *Communications of the ACM*, 42(11):42-46, 1999.

(Numao and Wakatsuki 1996) Numao, M.; Wakatsuki, Y. “ILP on Decision Trees” Japanese Society for Artificial Intelligence, SIG-FAI-9601, pp. 124-128, 1996.

(Plotkin 1971a) Plotkin, G.D. *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University, 1971.

(Plotkin 1971b) Plotkin, G.D. A further note on inductive generalization. In *Machine Intelligence*, volume 6, pages 101--124. Edinburgh University Press, 1971.

(Quinlan 1990) Quinlan, J.R. “Learning logical definitions from Relations”. *Machine Learning*, 5:239--266, 1990.

(Quinlan & Cameron-Jones 1993) Quinlan, J.R.; Cameron-Jones, R.M. “FOIL: A Midterm Report”. In P. Brazdil, editor, Proceedings of the 6th European Conference on Machine Learning, volume 667 of Lecture Notes in Artificial Intelligence, pages 3--20. Springer-Verlag, 1993.

(Quinlan 1996) Quinlan, J.R. “Learning first-order definitions from relations” *Machine Learning*, 5(3), 239-266, 1996.

(Rouveirol 1992) C. Rouveirol, C. “Extensions of Inversion of Resolution Applied to Theory Completion”. In S. Muggleton, editor, Inductive Logic Programming, pages 63--92. Academic Press, 1992.

Referencias del Tema 1b

(Sammut & Banerji 1986) Sammut, C. and Banerji, R. “Learning concepts by asking questions”. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, Machine Learning: An Artificial Intelligence Approach, Volume 2, pages 167--191. Morgan Kaufmann, 1986.

(Shapiro 1981) Shapiro. E.Y., An algorithm that infers theories from facts. In Proceedings of the 7th International Joint Conference on Artificial Intelligence, pages 446--452. Morgan Kaufmann, 1981.

(Shapiro 1983) Shapiro. E.Y. Algorithmic Program Debugging. The MIT Press, 1983.

(Utgoff, Berkman, Clouse 1997) Utgoof, P.E.; Berkman, N.C.; Clouse, J.A. “Decision Tree Induction Based on Efficient Tree Restructuring” Machine Learning, October 1997.

(Varsek 1993) Varsek, A.: 1993, `Genetic Inductive Logic Programming'. Ph.D. thesis, University of Ljubljana, Slovenia.

(Olson 1995) Olson, R.: 1995, `Inductive functional programming using incremental program transformation'. *Artificial Intelligence*, 74(1), 55--81.

Referencias del Tema 1b

faltan:

(Fayyad 1994) GID3 “BUSCAR”
Kodratoff y ...
Summers... LISP?
Vere 1975 LISP?
Vere 1977