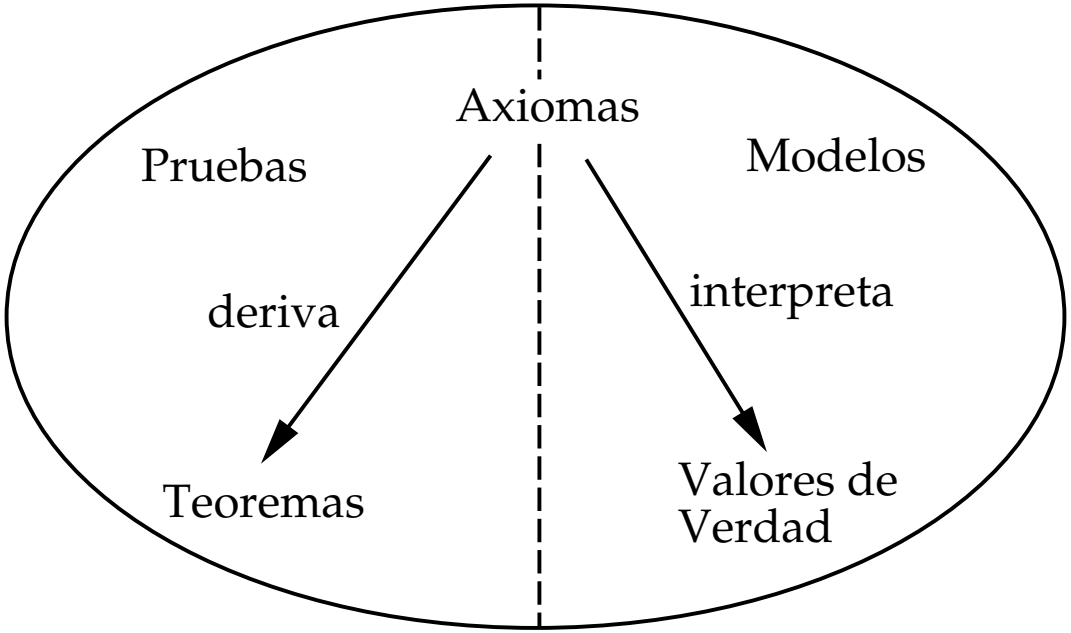
Progamación Lógica Inductiva

Eduardo Morales, Hugo Jair Escalante

INAOE

Contenido

Las dos caras de la Lógica



En lógica queremos que las cosas que son verdaderas coicidan con las que podemos probar o *que lo que nos implica la teoría es lo que podamos computar*

Lógica

- Características:
  - Sintáxis y semántica bien definidas
  - Reglas de inferencia
- Un *alfabeto* consiste de variables (aquí la primera letra en mayúscula), símbolos de predicados y de funciones (la primera letra en minúscula)
- *Términos* = Funciones (símbolo funcional + argumentos) y Variables
- *Un predicado* (símbolo + argumentos) es una fórmula atómica o simplemente un átomo

Lógica

<table>
<tr><td>válido</td><td colspan="2">inválido</td></tr>
<tr><td>siempre cierto</td><td>a veces T o F</td><td>siempre falso</td></tr>
<tr><td colspan="2">satisfacible</td><td>insatisfacible</td></tr>
</table>

Lógica

- Una fórmula G se dice que es una *consecuencia lógica* de un conjunto de fórmulas $F = \{F_1, \ldots, F_n\}, N \geq 1$, denotado por $F \models G$ si para cada interpretación w para la cual $w(F_1 \wedge F_2 \wedge \ldots F_n) = true$, entonces $w(G) = true$
- Satisfactibilidad, validez, equivalencia y consecuencia lógica son nociones semánticas
- Para derivar consecuencias lógicas también se pueden hacer por medio de operaciones exclusivamente sintáctivas

Cláusulas

- Utilizada en prueba de teoremas y programación lógica
- Una literal: un átomo o su negación
- Cláusula: fórmula cerrada de la forma:
  $\forall X_1 \ldots \forall X_s (L_1 \vee \ldots \vee L_m)$
  L_i = literal y X_i = todas las variables de las literales
- Equivalencias: $\forall x_1 \ldots \forall x_s (A_1 \vee \ldots A_n \vee \neg B_1 \ldots \vee \neg B_m) \equiv$
  $\forall x_1 \ldots \forall x_s (B_1 \wedge \ldots \wedge B_m \rightarrow A_1 \vee \ldots A_n)$
- Se escribe normalmente como:
  $A_1, \ldots, A_n \leftarrow B_1, \ldots B_m$

Cláusulas de Horn

Una cláusula de Horn: a lo más una literal positiva.

$$\begin{array}{l} A \leftarrow \\ \quad \leftarrow B_1, \ldots, B_n \\ A \leftarrow B_1, \ldots, B_n \end{array}$$

Una cláusula definitiva (*definite clause*) es una cláusula con una literal positiva ($A \leftarrow$ ó $A \leftarrow B_1, \ldots, B_n$)

Reglas de Inferencia

- Sólo hacen manipulación sintáctica (son formas procedurales)
- Lo interesante es ver cómo se relacionan las reglas semánticas con las sintácticas.
- Una regla de inferencia es *sólida* (*sound*) si $S \vdash F$ entonces $S \models F$
  Preserva la noción de verdad bajo las operaciones de derivación
- Una regla de inferencia es *completa* (*complete*) si $S \models F$ entonces $S \vdash F$

Resolución

- Sólo sirve para fórmulas en forma de cláusulas
- Idea: Prueba por refutación: Para probar: $P \vdash Q$, hacer $W = P \cup \{\neg Q\}$ y probar que W es insatisfactible ($\square$)
- Sean C_1 y C_2 dos cláusulas con literales L_1 y L_2 (donde L_1 y L_2 son *complementarias*)
- La resolución de C_1 y C_2 produce: $C = C_1' \cup C_2'$ donde: $C_1' = C_1 - \{L_1\}$ y $C_2' = C_2 - \{L_2\}$

Árbol de Derivación

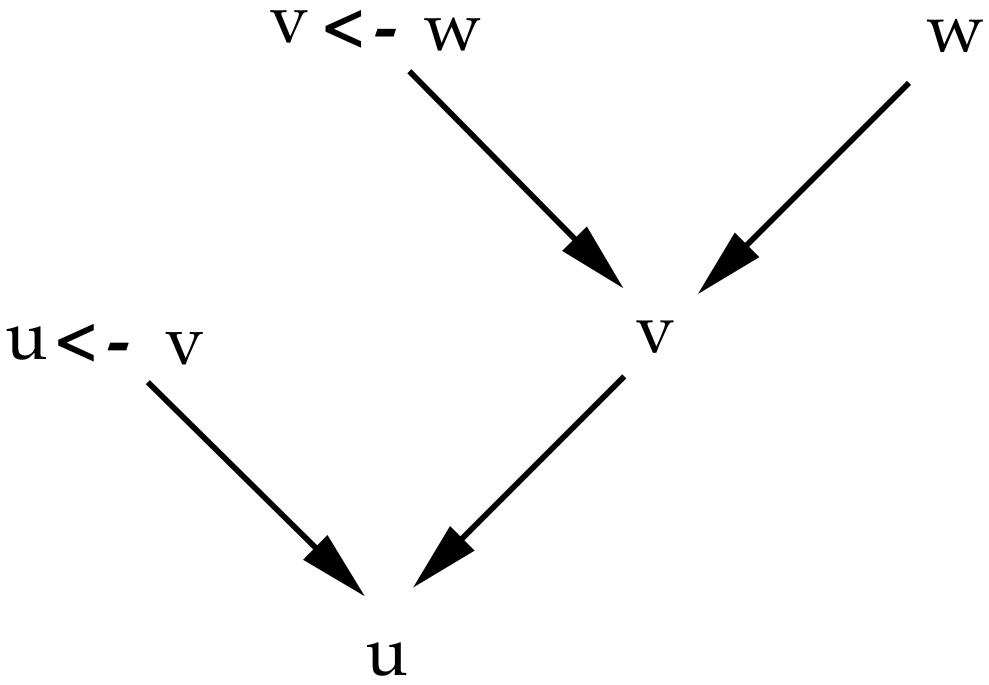


Figure: Un árbol de derivación proposicional

Unificación

- Para lógica de primer orden: *Sustitución* y *Unificación*
- Una *sustitución* $\Theta = \{X_1/t_1, \ldots, X_k/t_k\}$ es una función de variables a términos. La aplicación $W\Theta$ de una sustitución Θ a una *wff* W se obtiene al reemplazar todas las ocurrencias de cada variable X_j por el mismo término t_j
- Una sustitución σ es un *unificador* de un conjunto de expresiones $\{E_1, \ldots, E_m\}$ si $E_1\sigma = \ldots = E_m\sigma$
- Un unificador θ, es el *unificador más general* (*mgu*) de un conjunto de expresiones E, si para cada unificador σ de E, existe una sustitución λ tal que $\sigma = \theta\lambda$

Resolución

- Para hacer resolución en lógica de primer orden se buscan *unificaciones* (*mgu*) entre literales complementarias
- Sean C_1 y C_2 dos cláusulas con literales L_1 y L_2 respectivamente. Si L_1 y $\neg L_2$ tienen un *mgu* σ, el *resolvente* de C_1 y C_2 es la cláusula: $(C_1\sigma - \{L_1\sigma\}) \cup (C_2\sigma - \{L_2\sigma\})$ (ver figura 2).
- El algoritmo de unificación no es determinístico (se pueden seleccionar las cláusulas de varias formas)

Árbol de Deriación

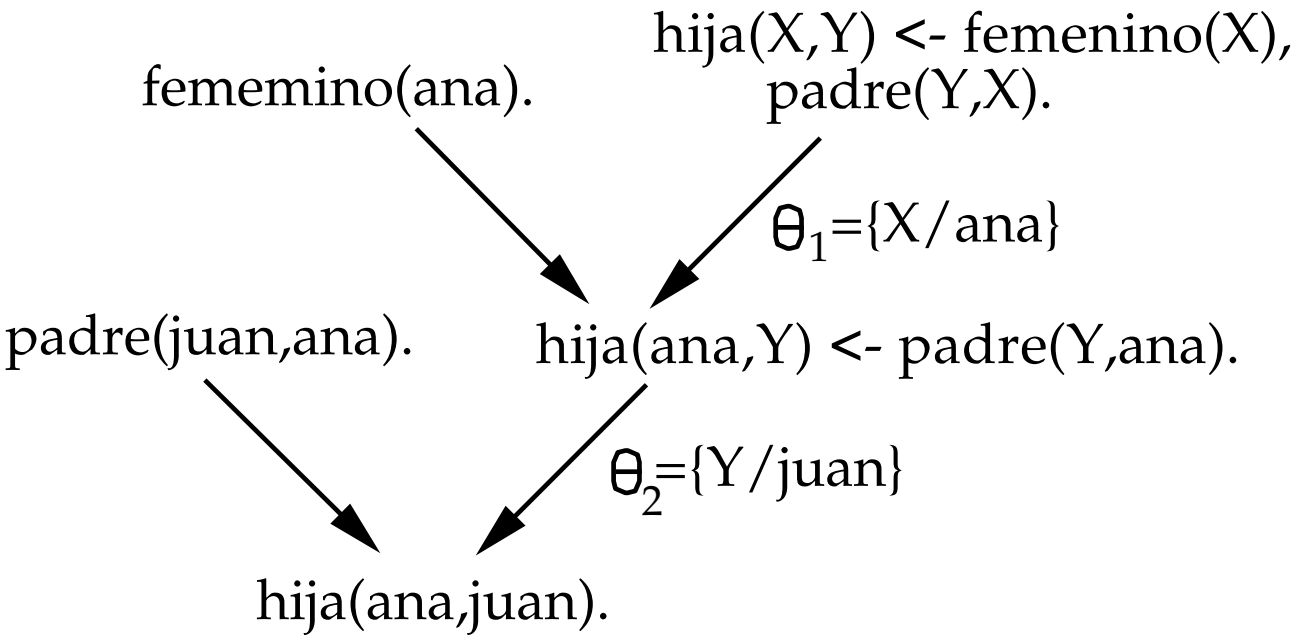


Figure: Un árbol de derivación lineal de primer orden

Estrategias de Resolución

- **Resolución lineal**: (i) El último resolvente se toma como cláusula padre y (ii) La otra cláusula padre se toma de otro resolvente o del conjunto original
- **Input resolution**: En cada paso de resolución, exceptuando el primero, se toma del último resolvente (cláusulas metas) y del conjunto original (cláusulas de entrada). Es *completa* para cláusulas de Horm.
- **Resolución SLD**: **S**eleccionar una literal, usando una estrategia **L**ineal, restringido a cláusulas **D**efinitivas.

Resolución en Prolog

- Aunque resolución SLD es *sound* y *refutation complete* para cláusulas de Horn, en la práctica (por razones de eficiencia) se hacen simplificaciones:
  - Eliminar el "occur check" de unificación
  - Usar un orden específico
- Esto es lo que usa básicamente PROLOG

Introducción ILP

- El lenguaje de representación de algunos de los algoritmos más exitosos de aprendizaje computacional (e.g., árboles de decisión o reglas de clasificación) es escencialmente proposicional
- Cada prueba que se hace sobre un atributo se puede ver como una proposición
- Por lo mismo, no podemos relacionar propiedades de dos o más objetos

ILP

- La Programación Lógica Inductiva o ILP (*Inductive Logic Programming*) combina resultados experimentales y métodos inductivos de ML con el poder de representación y formalismo de la lógica de primer orden
- Supongamos que queremos aprender (y por lo tanto representar) los movimientos de una torre en ajedrez
- Podemos representar los movimientos con cuatro atributos, *col1, ren1, col2* y *ren2* (columna y renglón antes y después del movimiento)

Ejemplo ILP

Un algoritmo proposicional produce algo como:
If col1 = 1 and col2 = 1 Then mov_torre = true
If col1 = 2 and col2 = 2 Then mov_torre = true
. . .
If col1 = 8 and col2 = 8 Then mov_torre = true
If ren1 = 1 and ren2 = 1 Then mov_torre = true
If ren1 = 2 and ren2 = 2 Then mov_torre = true
. . .
If ren1 = 8 and ren2 = 8 Then mov_torre = true

Ejemplo ILP

En una representación relacional, si suponemos que tenemos un predicado $mov(X, Y, Z, W)$ cuyos argumentos representan la columna y renglón antes y después del movimiento, se puede producir algo como:

$$mov(X, Y, X, Z) : -Y \neq Z.$$
$$mov(X, Y, Z, Y) : -X \neq Z.$$

Ventajas ILP

- Representación más compacta
- Relaciona propiedades de más de un objeto a la vez
- Puede incluir conocimiento del dominio dentro del proceso de aprendizaje

Críticas de sistemas de ML

Las críticas actuales a los sistemas de aprendizaje son:

- Generalización *pobre*
- Poca o nula interpretabilidad
- Necesidad de una gran cantidad de datos

ILP

Los sistemas de ILP pueden:

- Usar conocimiento del dominio
- Aprender teorías relacionales complejas
- Inducir modelos entendibles
- Generalizar con pocos ejemplos
- Dar soporte a *tranfer* y a *lifelong learning*

Ejemplo ILP

Aprender el concepto de $hija(X, Y)$ (es verdadero si X es hija de Y)

Ejemplos positivos ($\oplus$) y negativos ($\ominus$):

$hija(fernanda, eduardo).\oplus$ $hija(eugenia, ernesto).\ominus$

$hija(camila, rodrigo).\oplus$ $hija(valentina, roberto).\ominus$

...

Conocimiento del Dominio:

$femenino(fernanda).$	$padre(eduardo, fernanda).$
$femenino(camila).$	$padre(rodrigo, camila).$
$femenino(eugenia).$	$padre(roberto, eugenia).$
$femenino(valentina).$	$padre(ernesto, valentina).$
...	...

Resultado: $hija(X, Y) :- femenino(X), padre(Y, X).$

Nuevos Predicados

- Algunos sistemas de ILP pueden introducir nuevos predicados automáticamente durante el aprendizaje, simplificando la representación de los conceptos aprendidos
- Por ejemplo, introducir *progenitor* refiriendose a *padre* o *madre*, para simplificar una representación de un concepto

Limitaciones de otros sistemas

- *Representación Restringida*: Inadecuados en áreas que requieren expresar conocimiento relacional (v.g., razonamiento temporal y/o espacial, planificación, lenguaje natural, razonamiento cualitativo, etc.).
- *Conocimiento del Dominio*: Incapaces de incorporar conocimiento del dominio
- *Vocabulario Fijo*: No pueden inventar nuevo vocabulario con conocimiento insuficiente del dominio.[1]

[1] Aunque existen sistemas proposicionales de *feature construction* que generan nuevos atributos como combinaciones de atributos existentes y uno de los argumentos para usar Deep Learning es que permiten definir los atributos adecuados.

Ejemplo

Otro ejemplo (ver figura 3):

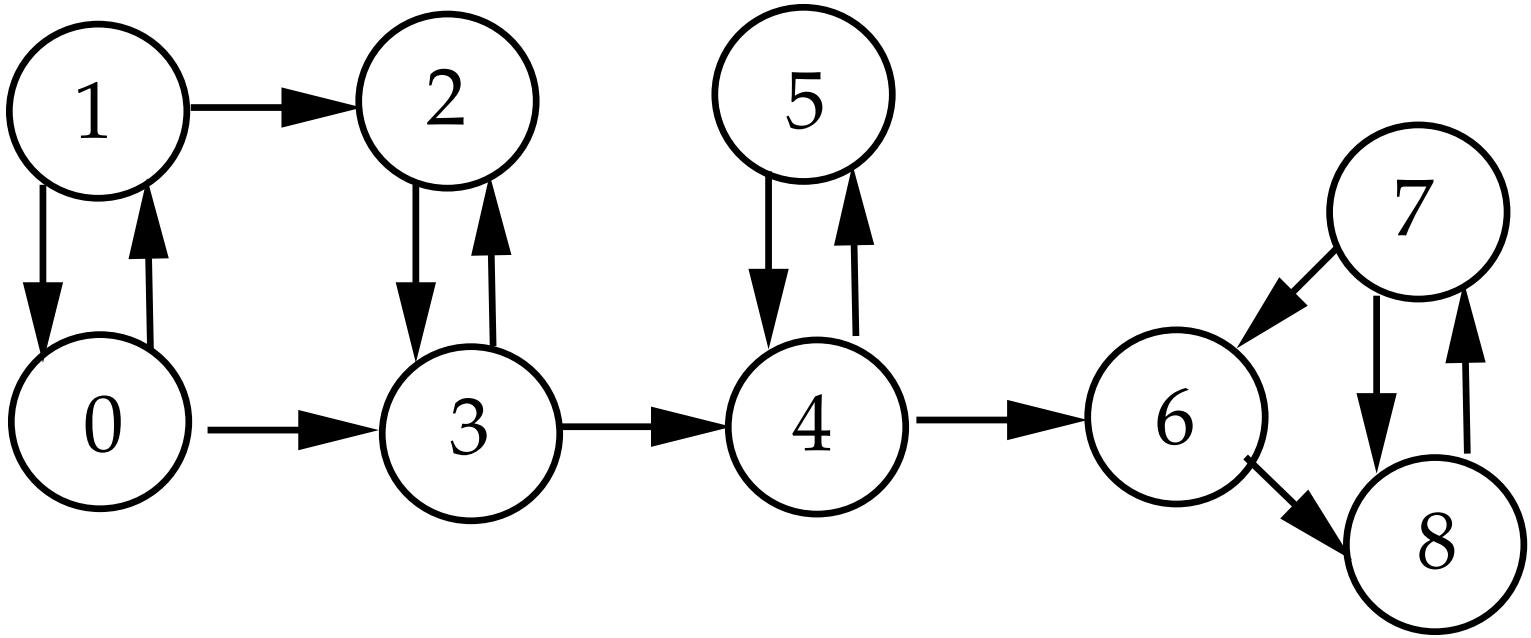


Figure: Grafo conectado.

Ejemplo

Ejemplos (en ILP): conectados(0,1). $\oplus$, conectados(8,1). $\ominus$,
...
Conocimiento del dominio:
liga(0,1). liga(0,3). liga(1,0). liga(1,2). liga(2,3).
liga(3,2). liga(3,4). liga(4,5). liga(4,6). liga(5,4).
liga(6,8). liga(7,6). liga(7,8). liga(8,7).

```
conectados(X,Y) :- liga(X,Y).
conectados(X,Y) :- liga(X,Z), conectados(Z,Y).
```

Aprendizaje en ILP

Aprender una hipótesis que cubra los ejemplos positivos y no cubra los negativos (se verifica cobertura con algún algoritmo de inferencia basado en resolución)

- Un programa lógico P se dice *completo* (con respecto a $\mathcal{E}^+$) sii para todos los ejemplos $e \in \mathcal{E}^+$, $P \vdash e$
- Un programa lógico P se dice *consistente* (con respecto a $\mathcal{E}^-$) sii para ningún ejemplo $e \in \mathcal{E}^-$, $P \vdash e$

Apredizaje en ILP

Dados:

- Un conjunto de ejemplos positivos $\mathcal{E}^+$
- Un conjunto de ejemplos negativos $\mathcal{E}^-$
- Un programa lógico consistente, $\mathcal{T}$, tal que $\mathcal{T} \nvdash e^+$ para al menos un $e^+ \in \mathcal{E}^+$

Encontrar un programa lógico $\mathcal{H}$ tal que $\mathcal{H}$ y $\mathcal{T}$ sea completo y consistente: $\mathcal{T} \cup \mathcal{H} \vdash \mathcal{E}^+$ y $\mathcal{T} \cup \mathcal{H} \nvdash \mathcal{E}^-$.

$\mathcal{T}$ normalmente se refiere a conocimiento del dominio o conocimiento *a priori*.

Aprendizaje en ILP

Desde un punto de vista semántico la definición de ILP es:

- Satisfactibilidad previa: $T \wedge E^- \not\models \square$
- Satisfactibilidad posterior (correcto o consistente): $T \wedge H \wedge E^- \not\models \square$
- Necesidad previa: $T \not\models E^+$
- Suficiencia posterior (completo): $T \wedge H \models E^+$

Interpretaciones y Modelos de Herbrand

- Interpretación: Asignar significado dentro de un dominio (valores de verdad) a cualquier expresión
- Por ejemplo, si tenemos: *gusta(juan,ana).*, tenemos que asociar *juan* y *ana* a elementos del dominio, y tenemos que asociar la relación *gusta/2* en el dominio
- Por ejemplo, *juan* con "persona-juan", *ana* con "persona-ana" y *gusta/2* con "persona-juan, persona-juan", "persona-ana, persona-ana" y "persona-juan, persona-ana" (por ejemplo)

Interpretaciones y Modelos de Herbrand

- Bajo esta interpretación, la relación: *gusta(juan,ana)* es verdadera.
- Si asignamos *juan* a “persona-ana” y *ana* a “persona-juan” y mantenemos la misma interpretación la relación *gusta(juan,ana)* es falsa
- Una interpretación que nos da un valor de verdad para una sentencia lógica se dice que la satisface y a la interpretación se le llama un *modelo* de la sentencia

Interpretaciones y Modelos de Herbrand

- En ILP, normalmente se aprenden cláusulas definitivas, las cuales tienen un modelo de Herbrand mínimo único $\mathcal{M}^+(\mathcal{T})$ y todas las fórmulas lógicas son o verdaderas o falsas.
- Por ejemplo, si tenemos:
  $gusta(juan, X) \leftarrow gusta(X, vino).$
  $gusta(ana, vino).$
- Tomando, sólo las constantes: *juan*, *ana* y *vino*, todas las instanciaciones aterrizadas (*ground*) del programa lógico son:
  $gusta(juan, juan) \leftarrow gusta(juan, vino).$
  $gusta(juan, ana) \leftarrow gusta(ana, vino).$
  $gusta(juan, vino) \leftarrow gusta(vino, vino).$
  $gusta(ana, vino).$

Interpretaciones y Modelos de Herbrand

- Podemos asignar valores de verdad a todos estos elementos y obtener modelos para ciertas interpretaciones
- Los modelos se pueden organizar en un *lattice*.
- En el ejemplo, el modelo mínimo de Herbrand es asignar el valor de verdad a: *gusta(ana,vino).* (a fuerzas) y a *gusta(juan,ana)* $\leftarrow$ *gusta(ana,vino).* (derivado del primero)
- Las consecuencias lógicas aterrizadas de un programa lógico son su modelo mínimo de Herbrand ($\mathcal{M}$) y también es lo que podemos derivar con resolución SLD en programas con cláusulas definitivas

ILP desde el punto de vista de Modelos

- Satisfactibilidad previa: $\forall e \in E^-$ son falsos en $\mathcal{M}^+(\mathcal{T})$
- Satisfactibilidad posterior (correcto o consistente): $\forall e \in E^-$ son falsos en $\mathcal{M}^+(\mathcal{T} \wedge \mathcal{H})$
- Necesidad previa: algunos $e \in E^+$ son falsos en $\mathcal{M}^+(\mathcal{T})$
- Suficiencia posterior (completo): $\forall e \in E^+$ son verdaderos en $\mathcal{M}^+(\mathcal{T} \wedge \mathcal{H})$

Un caso especial, el que más se usa en ILP, es cuando todos los ejemplos son hechos (sin variables)

Búsqueda de Hipótesis

- El proceso de inducción puede verse como un proceso de búsqueda
- En ILP este espacio puede ser demasiado grande
- Para realizar una búsqueda eficiente de hipótesis, normalmente es necesario estructurar el espacio de hipótesis, lo cual se puede hacer con un modelo de generalización

Subsumption

- Esta estructuración se puede hacer utilizando $\Theta-$ *subsumption*
- Una cláusula C, θ–subsume (o es una generalización de) una cláusula D si existe una substitución θ tal que $C\theta \subseteq D$. Usualmente se escribe como $C \preceq D$.
- Ejemplo: Sea $C = hija(X, Y) \leftarrow padre(Y, X)$. Con la substitución vacía, C subsume a $hija(X, Y) \leftarrow femenino(X), padre(Y, X)$.
- Con la substitución $\Theta = \{Y/X\}$, C subsume a $hija(X, X) \leftarrow femenino(X), padre(X, X)$
- Con la substitución $\Theta = \{X/ana, Y/luis\}$, C subsume a $hija(ana, luis) \leftarrow femenino(ana), padre(luis, ana), padre(luis, pepe)$.

$\Theta-$Subsumption

- Una cláusula C es más general que C' si $C\Theta$-subsume a C' y no al revés; También se dice que C' es una especialización (o refinamiento) de C
- Si $C\Theta$-subsume a C', entonces C' es una consecuencia lógica de C, $C \models C'$, pero al revés no se cumple
- $\Theta-$*subsumption* es decidible entre cláusulas y fácil de calcular (aunque es NP), mientras que implicación no es decidible
- Crea un *lattice* y permite buscar por hipótesis: (i) De específico a general, (ii) de general a específico, ó (iii) en ambos sentidos

Ejemplo

Ejemplos	**Conocimiento del Dominio**
ordena([2,1],[1,2]).	junta([ ],L,L).
ordena([0,3,1],[0,1,3]).	junta([H|L1],L2,[H|L3]) $\leftarrow$
ordena([4,2,6],[2,4,6]).	junta(L1,L2,L3).
ordena([1],[1]).	
ordena([ ],[ ]).	divide(El,[H|T],Men,[H|May]) $\leftarrow$
...	El $<$ H, divide(T,Men,May).
	divide(El,[H|T],[H|Men],May) $\leftarrow$
	El $>=$ H, divide(T,Men,May).
	divide(_,[ ],[ ],[ ]).

Ejemplo: Espacio de Soluciones

Muy general:

ordena(X,Y).

Solución:

ordena([ ],[ ]).

ordena([H|T],LOrd) ←

divide(H,T,Men,May),
ordena(Men,MenOrd),
ordena(May,MayOrd),
junta(MenOrd,[H|MayOrd],LOrd).

Muy específico:

ordena([1],[1]) ←

junta([ ],[1],[1]), junta([1],[2,3],[1,2,3]), ...,
divide(1,[ ],[ ],[ ]), divide(2,[1,5],[1],[5]), ...
ordena([ ],[ ]), ordena([4],[4]), ...

Generalización menos general

- Una forma de ir búscando hipótesis es generalizando cláusulas gradualmente
- La *generalización menos general* (*lgg*) de dos cláusulas C y C' es la generalización más específica de las cláusulas C y C' dentro del *lattice* generado por $\Theta-$*subsumption*
- C es la generalización menos general (*lgg*) de D bajo $\theta-$*subsumption* si $C \preceq D$ y para cualquier otra E tal que $E \preceq D$, entonces $E \preceq C$

Algoritmo de *lgg* entre dos términos

Sean L_1 y L_2 dos términos o literales compatibles

1. Sea $P_1 = L_1$ y $P_2 = L_2$
2. Encuentra dos términos, t_1 y t_2, en el mismo lugar en P_1 y P_2, tal que $t_1 \neq t_2$ y, o los dos tienen un nombre de función diferente o por lo menos uno de ellos es una variable
3. Si no existe ese par, entonces acaba: $P_1 = P_2 = lgg(L_1, L_2)$
4. Si existe, escoge una variable X distinta de cualquier variable que ocurra en P_1 o P_2, y en donde t_1 y t_2 aparezcan en el mismo lugar en P_1 y P_2, reemplazalos con X
5. Ve a 2

LGG vs MGU

- Con respecto a átomos, *lgg* es el dual de *mgu*. Dados dos términos f_1 y f_2 y el orden impuesto por $\preceq$, entonces el *lgg* de f_1 y f_2 es su límite inferior más grande (glb) y el *mgu* es el límite superior más bajo (lub)
- Si tenemos dos literales el *lgg* es la literal más específica que subsume a las dos literales, mientras que el *mgu* es la literal más general subsumida por las dos

$$\overbrace{\underbrace{L_1 = foo(a, f(a), g(X, b), Z)}_{mgu(L_1, L_2) = foo(a, f(a), g(c, b), Z).} \qquad \underbrace{L_2 = foo(Y, f(Y), g(c, b), Z)}_{}}^{lgg(L_1, L_2) = foo(V, f(V), g(W, b), Z).}$$

LGG

El *lgg* de dos cláusulas C_1 y C_2 está definido por: $\{l : l_1 \in C_1 \text{ y } l_2 \in C_2 \text{ y } l = lgg(l_1, l_2)\}$. Por ejemplo, si:

$$\begin{aligned} C1 &= hija(fernanda, eduardo) \leftarrow padre(eduardo, fernanda), \\ &\quad femenino(fernanda), pequeña(fernanda). \\ C2 &= hija(camila, rodrigo) \leftarrow padre(rodrigo, camila), \\ &\quad femenino(camila), grande(camila). \end{aligned}$$

$lgg(C1, C2) = hija(X, Y) \leftarrow padre(Y, X), femenino(X)$.
La longitud del *lgg* de las cláusulas C_1 y C_2 es a lo más $| C_1 | \times | C_2 |$

RLGG

- En general nos interesa encontrar generalizaciones de un conjunto de ejemplos en relación a cierta teoría o conocimiento del dominio
- Una cláusula C es más general que una D con respecto a una teoría T si $T \wedge C \vdash D$
- Una cláusula C es un *lgg* de una cláusula D con respecto a una teoría T, si $T \vdash C\Theta \rightarrow D$ para alguna substitución Θ
- Decimos que C es la generalización menos general de D relativa a T (*rlgg*)

RLGG

- En general, puede no existir un *rlgg*, pero sí existe para teorías aterrizadas (sin variables)
- En particular, si T es un conjunto finito de literales aterrizadas, el *lgg* de C_1 y C_2 con respecto a T, es: $lgg(T \rightarrow C_1, T \rightarrow C_2)$
- *Rlgg* sin embargo, puede tener algunas conclusiones no intuititvas, por ejemplo, es fácil de verificar que: $P \leftarrow Q$ es más general que $R \leftarrow S, Q$ relativa a $R \leftarrow P, S$

Subsumción Generalizada

- Caso especial de *rlgg*, restringido a cláusulas definitivas
- La idea es que C es más general que D con respecto a T, si cada vez que D se puede usar (junto con T) para explicar algún ejemplo, C también se pueda usar
- Una cláusula $C \equiv C_{cabeza} \leftarrow C_{cuerpo}$, subsume a otra cláusula $D \equiv D_{cabeza} \leftarrow D_{cuerpo}$ con respecto a T ($C \preceq_T D$) si existe una substitución mínima σ tal que $C_{cabeza}\sigma = D_{cabeza}$ y para cualquier substitución aterrizada (*ground*) θ con constantes nuevas para D, se cumple que: $T \cup D_{cuerpo}\theta \models \exists\,(C_{cuerpo}\sigma\theta)$

RLGG

Sean:

1. C y D dos cláusulas con variables disjuntas, y T un programa lógico,
2. θ_1 una substitución (*ground*) para las variables en C_{cabeza},
3. θ_2 una substitución para el resto de las variables en C, y similarmente, ϕ_1 y ϕ_2 para D

Si $lgg_T(C, D)$ existe, es equivalente al $lgg(C', D')$, donde:

$$C' \equiv C\,\theta_1 \cup \{\neg A_1, \ldots, \neg A_n\} \text{ y } D' \equiv D\,\phi_1 \cup \{\neg B_1, \ldots, \neg B_m\}$$

y para $1 \leq i \leq n$, $T \wedge C_{cuerpo}\theta_1\theta_2 \models A_i$, y A_i es un átomo aterrizado construido con símbolos que ocurren en T, C, θ_1, θ_2, y D, y similarmente para cada B_j

RLGG

Esto se puede utilizar dentro de un sistema de aprendizaje de la siguiente forma:

- Toma una cláusula ejemplo (C_1) y sea $\theta_{1,1}$ una substitución instanciando las variables en la cabeza de C_1 a nuevas constantes y $\theta_{1,2}$ instanciando las variables que quedan a nuevas constantes
- Construye una nueva cláusula *saturada* (*NC*) definida como: $NC \equiv C_1\theta_{1,1} \cup \{\neg A_{1,1}, \neg A_{1,2}, \ldots\}$ donde $T \wedge C_{1\,cuerpo}\theta_{1,1}\theta_{1,2} \models A_{1,i}$, y $A_{1,i}$ es una átomo instanciado
- Construye para cada ejemplo, su cláusula saturada, y calcula el *lgg* entre ellas

Ejemplo

Queremos aprender *faldero* y tenemos las siguientes dos cláusulas ejemplo:
$C = faldero(fido) \leftarrow consen(fido), peque(fido), perro(fido).$
$D = faldero(morris) \leftarrow consen(morris), gato(morris).$
Entonces:

$$lgg(C, D) = faldero(X) \leftarrow consen(X).$$

lo cual podría ofender a varias personas

Ejemplo

Si tenemos como conocimiento del dominio:
$mascota(X) \leftarrow perro(X).$
$mascota(X) \leftarrow gato(X).$
$peque(X) \leftarrow gato(X).$

Añadimos al cuerpo de C y D lo que podamos deducir del cuerpo de cada cláusula con el conocimiento del dominio:
$C' = faldero(fido) \leftarrow$
$consen(fido), peque(fido), perro(fido), mascota(fido).$
$D' = faldero(morris) \leftarrow$
$consen(morris), gato(morris), mascota(morris), peque(morris).$

Entonces: $rlgg_T(C, D) = lgg(C', D') =$
$faldero(X) \leftarrow consen(X), peque(X), mascota(X).$

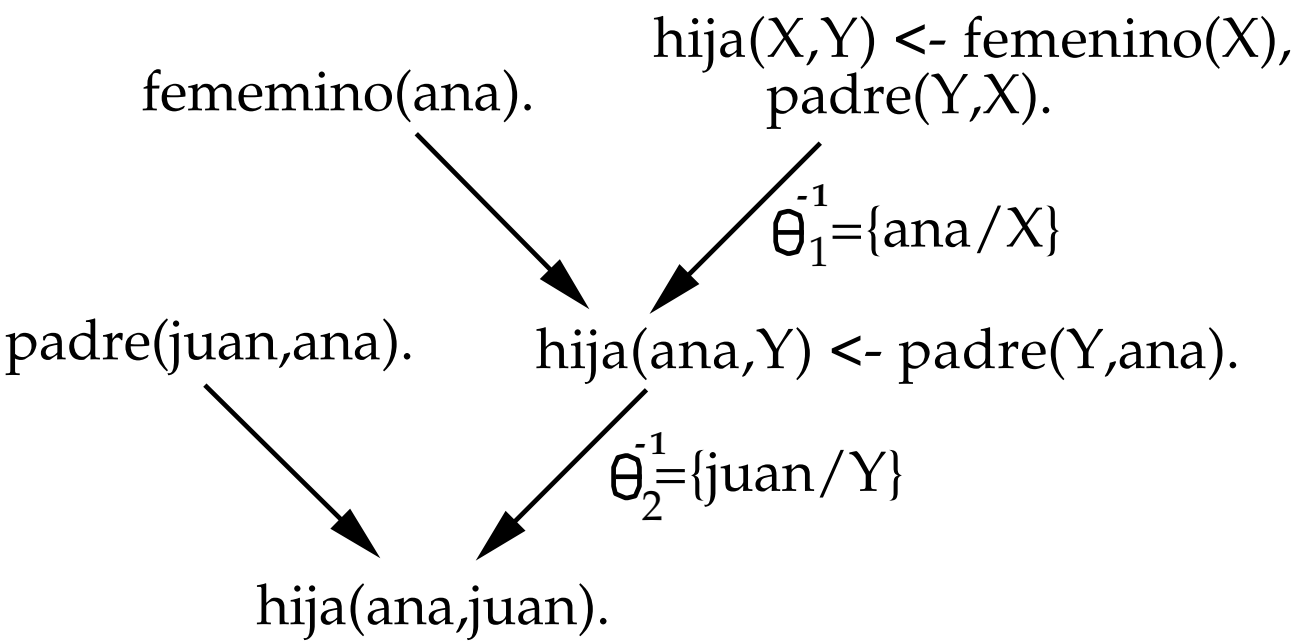
Inversión de Resolución

- Idea: Invertir *resolución*
- Necesitamos definir una substitución inversa Θ^{-1} que mapea términos a variables
- Ejemplo, si $C = hija(X, Y) \leftarrow femenino(X), padre(Y, X)$, la substitución: $\Theta = \{X/ana, Y/juan\}$ nos da: $C' = C\Theta = hija(ana, juan) \leftarrow femenino(ana), padre(juan, ana)$ y la substitución inversa: $\Theta^{-1} = \{ana/X, juan/Y\}$ nos da: $C'\Theta^{-1} = hija(X, Y) \leftarrow femenino(X), padre(Y, X)$.
- De forma similar, si conocemos $hija(ana, juan)$ y $padre(juan, ana)$ podríamos aplicar un paso inverso de resolución para obtener $hija(ana, Y) \leftarrow padre(Y, ana)$, con una substitución inversa de $\Theta_2^{-1} = \{juan/Y\}$

Inversión de Resolución

- Si además sabemos que $femenino(ana)$, podríamos aplicar otro proceso inverso de resolución para obtener $hija(X, Y) \leftarrow femenino(X), padre(Y, X)$ con $\Theta_1^{-1} = \{ana/X\}$
- En general, se tienen que especificar los lugares, dentro de la cláusula, en donde se hace la substitución
- Ejemplo: $c = quiere(X, hija(Y))$. con $\Theta = \{X/ana, Y/ana\}$ nos da: $c\Theta = quiere(ana, hija(ana))$. Para recuperar c necesitamos tener: $\Theta^{-1} = \{(ana, \{1\})/X, (ana, \{2, 1\})/Y\}$ para poder recuperar la c original

Árbol de derivación inversa



Resolución Inversa

El tratar de invertir el proceso presenta algunos problemas:

- En general no existe una solución única
- Tenemos que decidir si vamos a cambiar términos a variables y cómo

Dado un árbol de derivación de dos cláusulas C_1 y C_2 para obtener C, el operador de *absortion*, construye C_2, dados C y C_1
De la ecuación del resultante de aplicar resolución podemos despejar C_2:

$$C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{L_2\}$$

donde θ_1 y θ_2 son substituciones involucrando únicamente las variables de las cláusulas C_1 y C_2 respectivamente

Resolución Inversa

- Tenemos que decidir qué términos y subtérminos se deben de remplazar por la misma variable y cuáles por variables diferentes
- *Cigol* supone que C_1 es una cláusula unitaria (i.e., $C_1 = L_1$), con lo que obtenemos:

$$C_2 = (C \cup \{\neg L_1\}\theta_1)\theta_2^{-1}$$

- Si C_1 es una cláusula aterrizada (ejemplo positivo) entonces θ_1 es vacía
- Para θ_2^{-1} tenemos que decidir cómo cambiar las ocurrencias de términos en variables

Absorption

- Con cláusulas de Horn en general, el cuerpo de C_1 es absorbido en el cuerpo de C (después de la aplicación de una unificación adecuada) y reemplazada con su cabeza
- Ejemplo: $C = ave(tweety) \leftarrow plumas(tweety), alas(tweety), pico(tweety).$
  $C_1 = vuela(X) \leftarrow plumas(X), alas(X).$
  El cuerpo de C_1 es absorbido en el cuerpo de C después de la substitución $\theta = \{X/tweety\}$ dando una posible solución:
  $C_2 = ave(tweety) \leftarrow vuela(tweety), pico(tweety).$

Absorption

- *Absorption* es destructiva, las literales remplazadas se pierden y no pueden usarse para futuras generalizaciones (es problema cuando los cuerpos de las cláusulas se traslapan parcialmente), por lo que las generalizaciones dependen del orden de estas
- Ejemplo:
  $C_1 = P \leftarrow Q, R.$
  $C_2 = S \leftarrow R, T.$
  $C_3 = V \leftarrow Q, R, T, W.$
  C_1 y C_2 comparten una literal y ambas se pueden usar para hacer *absorption* con respecto a C_3. *Absorption* de C_3 con C_1 nos da: $C_4 = V \leftarrow P, T, W$, pero ahora no se puede hacer *absorption* de C_3 con C_2

Saturación

- Este problema se puede resolver si usamos *saturación*
- Saturación hace todas las posibles deducciones en el cuerpo de una cláusula de entrada usando el conocimiento del dominio, y se generaliza al eliminar las literales redundantes
- Ejemplo:
  C_1 y C_3: $C_4 = V \leftarrow [Q, R], P, T, W.$
  C_4 y C_2: $C_5 = V \leftarrow [Q, R, T], W, P, S.$

Operador "W"

- Otro operador que invierte el proceso de resolución es el operador "W" que se obtiene al combinar dos operadores "V"
- Supongamos que C_1 y C_2 resuelven en una literal común L dentro de la cláusula A para producir B_1 y B_2
- El operador W construye A, C_1 y C_2 dados B_1 y B_2
- Cuando L es negativo se llama *intraconstruction* y cuando es positivo *interconstruction*

Operador "W"

- Como la literal L en A es eliminada en la resolución y no se encuentra en B_1 o B_2, se tiene que inventar un nuevo predicado.

$$B_1 = (A - \{l_1\})\theta_{A,1} \cup (C_1 - \{l_1\})\theta_{C,1}$$

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

- Suponiendo otra vez que C_1 y C_2 son cláusulas unitarias:

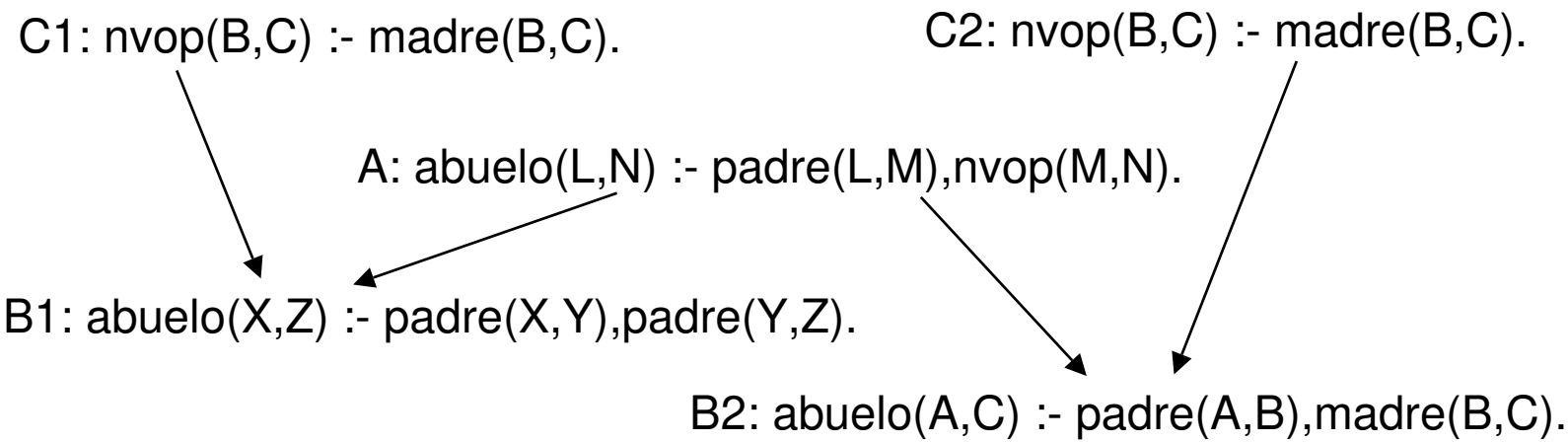
$$A = B_1\theta_{A,1}^{-1} \cup \{l\} = B_2\theta_{A,2}^{-1} \cup \{l\} = B \cup \{l\}$$

donde B es una generalización común de las cláusulas B_1 y B_2.

Ejemplo

- Por ejemplo, supongamos que:
  $B_1 = abuelo(X, Z)$:- $padre(X, Y), padre(Y, Z)$.
  $B_2 = abuelo(A, C)$:- $padre(A, B), madre(B, C)$.
- Podemos aplicar el operador "W" para obtener las siguientes cláusulas:
  $A = abuelo(L, N)$:- $padre(L, M), nvop(M, N)$.
  $C_1 = nvop(Y, Z)$:- $padre(Y, Z)$.
  $C_2 = nvop(B, C)$:- $madre(B, C)$.

Ejemplo del operador "W"



Un esquema común de generalización

- Relaciona resolución inversa y *rlgg*
- Idea, para cada ejemplo que se tenga:
  1. Realizar resolución inversa lineal con substitución inversa más específica (vacía) y
  2. Evaluar el *lgg* de las cláusulas resultantes
- Esto es, las generalizaciones más específicas con respecto a conocimiento del dominio (*rlgg*) son las generalizaciones más específicas (*lgg*) de los árboles inversos de derivación más específicos

Resolución inversa y RLGG

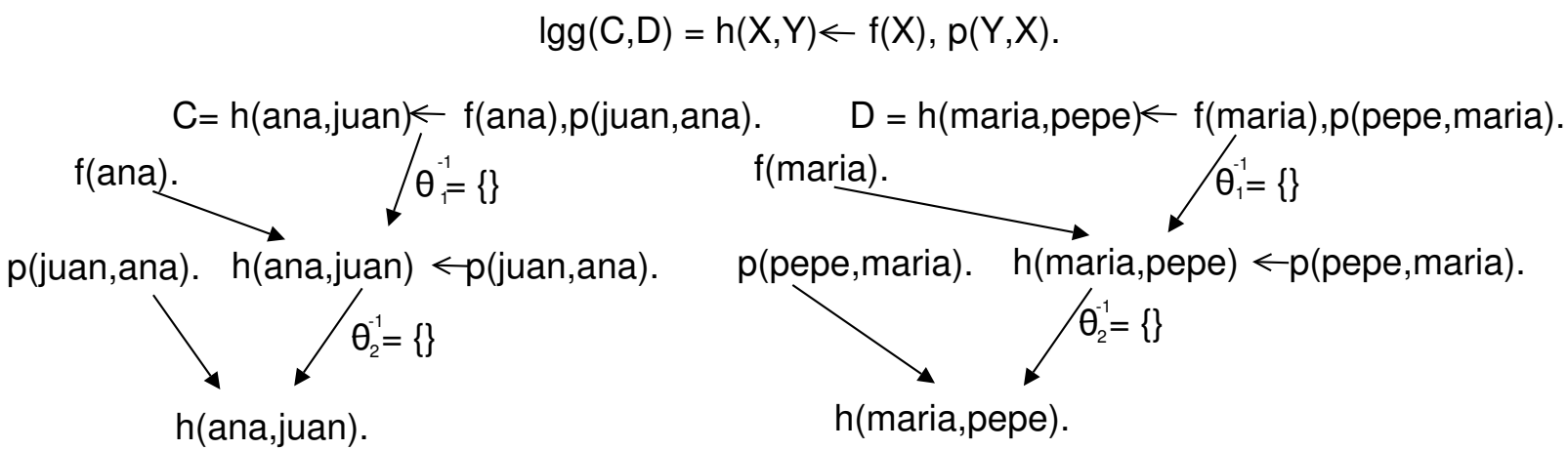


Figure: Esquema común de generalización, donde *h* se refiere a *hija*, *f* a *femenino* y *p* a *padre*.

Inversión de Implicación

- Bajo una interpretación semántica podemos pensar en invertir implicación
- Sean C y D cláusulas. Decimos que C implica D, o $C \rightarrow D$, sii todo modelo de C también es modelo de D, i.e., $C \models D$.
- Decimos que C es una generalización (bajo implicación) de D.
- El problema de invertir implicación es que implicación es indecidible y computacionalmente es muy costoso, a menos, que se impongan ciertas restricciones

Inversión de Implicación

- Lo que quiere encontrar es una H tal que: $T \wedge H \models E$
- Por el teorema de deducción: $T \wedge \neg E \models \neg H$, donde $\neg E$ y $\neg H$ representan conjunciones de literales aterrizadas (Skolemizadas)
- Si $\neg \bot$ representa todas las literales aterrizadas (potencialmente infinitas) ciertas en todos los modelos de: $T \wedge \neg E$, $\neg H$ debe de ser un subconjunto de $\neg \bot$
- Por lo que: $T \wedge \neg E \models \neg \bot \models \neg H$ y para toda H, $H \models \bot$
- Con esto se puede buscar a H en cláusulas que subsumen a $\bot$

Inversión de Implicación

- En la práctica para construir $\perp$ se utiliza resolución SLD[2] de profundidad limitada (h)
- Al resultado se le conoce como modelos *h-easy*
- Una forma de encontrar H es construyendo gradualmente hipótesis que sean subconjuntos de $\perp$ siguiendo una estrategia de general a específico (e.g., *Progol*)

[2] **S**eleccionar una literal, usando una estrategia **L**ineal, restringido a cláusulas **D**efinitivas.

Búsqueda de Específico a General

- Los algoritmos que buscan de específico a general pueden tener problemas en presencia de ruido (e.g., AQ)
- Esto originó el proponer algoritmos de general a específico que siguen una estrategia de *covering* (e.g., CN2 o PART)
- Esta misma estrategia se propuso en ILP

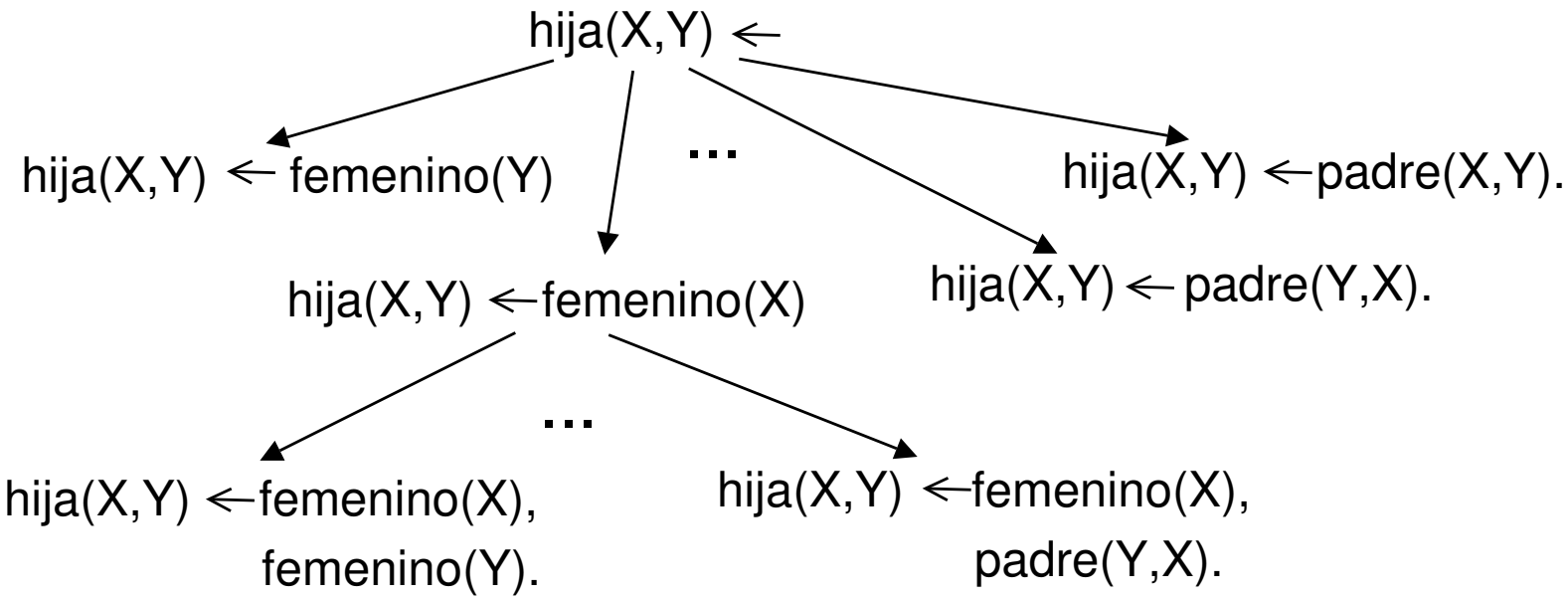
Sistemas de General a Específico

- Siguen un esquema basado en *covering*
- Idea: Ir añadiendo literales (condiciones a reglas) siguiendo un proceso de búsqueda (generalmente *hill-climbing*), usando una medida heurística
- Cuando se cumple el criterio de paro se eliminan los ejemplos positivos cubiertos y se empieza a generar una nueva cláusula
- Sin ruido: Completas y consistentes
- Con ruido: Semi-completas y semi-consistentes

Algoritmo de General a Específico

Inicializa $\mathcal{E}_{actual} := \mathcal{E}$
Inicializa $H := \emptyset$
repeat
  Inicializa $C := T \leftarrow$
  repeat
    Encuentra el mejor refinamiento (C_{mejor}) de C
    Sea $C := C_{mejor}$
  until criterio de paro (necesidad)
  Añade C a H, $H := H \cup \{C\}$
  Elimina ejemplos positivos cubiertos por C de $\mathcal{E}_{actual}$
until criterio de paro (suficiencia)
Regresa H

Proceso de Construcción de Hipótesis



Operadores de Refinamiento

- Se tiene que especificar el criterio a utilizar para seleccionar una nueva literal y el criterio de paro
- Para añadir una nueva literal (especializar una cláusula) se puede hacer con operadores de refinamiento (*refinement operators*)
- Q es un refinamiento de T si T implica Q y $\mathit{tamaño}(T) < \mathit{tamaño}(Q)$, donde *tamaño* es una función que hace un mapeo de cláusulas a números naturales

Operadores de Refinamiento

- Un operador de refinamiento se dice *completo* sobre un conjunto de cláusulas, si podemos obtener todas las cláusulas por medio de refinamientos sucesivos a partir de la cláusula vacía
- Un operador de refinamiento induce un orden parcial sobre el lenguaje de hipótesis
- Dado un operador de refinamiento completo para su lenguaje de hipótesis, se recorre su grafo de refinamiento hasta encontrar la hipótesis deseada
- La diferencia entre muchos de los sistemas de ILP radica en qué operador de refinamiento utilizan y cómo recorren su grafo de refinamiento

Ganancia de Información y FOIL

- Se puede ver como una extensión “natural” de ID3
- Utiliza una medida basada en ganancia en información
- Empieza con cláusulas unitarias instanciadas representando ejemplos positivos, negativos y la teoría del dominio y aprende incrementalmente cláusulas hasta cubrir todos los ejemplos positivos y ningúno de los negativos

Foil

- Una *tupla* es una secuencia finita de constantes
- Una tupla satisface una cláusula si existe un mapeo de las variables de la cabeza de la cláusula hacia la tupla y una extensión de todos las variables del cuerpo a constantes satisfaciendo el cuerpo
- Foil empieza con tuplas positivas y negativas y con una cláusula muy general, la cual es gradualmente especializada añadiéndole literales al cuerpo

Foil

- Si la literal añadida usa sólo variables existentes, el nuevo conjunto de tuplas positivas y negativas es un subconjunto de aquellas tuplas que satisfacen el predicado adicional
- Si se introduce una nueva variable, las tuplas se tienen que extender para incluir los valores de esa variable
- La asignación de valor a las tuplas (positiva o negativa) se toma de la asignación original

Ejemplo

ligados(X, Y).

$\oplus$: (0,1), (0,2), (0,3), (0,4), ..., (7,8)
$\ominus$: (0,0), (0,7), (1,0), (1,1), ..., (8,8)

ligados(X, Y) :- *liga*(X, Y).
Elimina 10 tuplas de $\oplus$

Ejemplo

ligados(X, Y) :- *liga*(X, Z).
Introduce una nueva variable y las nuevas tuplas serían:

$\oplus$: (0,2,1), (0,2,3), (0,4,1), ..., (4,8,6)
$\ominus$: (0,0,1), (0,0,3), (0,7,1), ..., (7,7,8)

ligados(X, Y) :- *liga*(X, Z), *ligados*(Z, Y).
cubre las otras tuplas positivas.

Refinamientos en Foil

Cada literal del cuerpo de la cláusula puede tomar una de las siguientes 4 formas (refinamientos):
(i) $X_j = X_k$
(ii) $X_j \neq X_k$
(iii) $P(V_1, V_2, \ldots, V_n)$
(iv) $\neg P(V_1, V_2, \ldots, V_n)$
donde X_i's son variables existentes, V_i's son variables existente o nuevas, y P es alguna relación.

Las nuevas literales en Foil deben de contener por lo menos una variable existente.

Heurística de Foil

Foil usa una métrica de ganacia de información para añadir nuevas literales:

$$Gain(literal) = T^{++} * [\log_2 (\frac{P_1}{P_1 + N_1}) - \log_2 (\frac{P_0}{P_0 + N_0})]$$

donde:

- P_0 y N_0 es el número de tuplas negativas y positivas antes de añadir la nueva literal
- P_1 y N_1 son el número de tuplas positivas y negativas después de añadir la literal a la cláusula
- T^{++} es el número de tuplas positivas antes de añadir la literal que satisfacen la nueva literal

Algoritmo Foil

Dado un conjunto de tuplas positivas y negativas y tuplas de conocimiento del dominio

repeat

sea cláusula actual = la cabeza del predicado más general con cuerpo vacío

repeat

- calcula la ganancia de información de todas las posibles literales que pueden añadirse a la cláusula
- selecciona la literal con más ganancia de información
- añade la literal al cuerpo de la cláusula actual
- elimina las tuplas de ejemplos positivos satisfechos por la nueva cláusula

until no se cubren tuplas negativas

until todas las tuplas positivas estén cubiertas

Foil

- La heurística de ganancia de información no garantiza encontrar una solución cuando existen varias posibles literales con ganancia aproximadamente igual
- Puede hacer decisiones locales óptimas pero globalmente malas
- Foil puede cambiar de hipótesis si se le dan los mismos ejemplos positivos dos veces
- Foil no tiene símbolos funcionales y se ve afectado por el número de argumentos en el predicado meta
- Extensiones (Foil2) utiliza un tipo de *back–up* primitivo

Medidas de calidad para las hipótesis

- Precisión: $A(c) = p(\oplus|c)$. Normalmente $p(\oplus|c) = \frac{n^{\oplus}(c)}{n(c)}$, donde $n^{\oplus}(c) =$ ejemplos positivos cubiertos por la cláusula c y $n(c) =$ todos los ejemplos cubiertos por c
- Basada en información: $I(c) = -log_2 p(\oplus|c)$.
- Ganancia en precisión al añadir una literal a $c \rightarrow c'$: $AG(c', c) = A(c') - A(c) = p(\oplus|c') - p(\oplus|c)$.
- Ganancia de información: $IG(c', c) = I(c) - I(c') = log_2 p(\oplus|c') - log_2 p(\oplus|c)$.
- Ganancia en precisión/información pesada: $\frac{n^{\oplus}(c')}{n^{\oplus}(c)}$
- Ganancia de información mejorada: $IG(c) = \frac{\frac{n^{\oplus}(c)+(|N^{\ominus}|-n^{\ominus}(c)}{|N|} * (I(\top)-I(c))}{|c|}$ donde $|N^{\ominus}| =$ núm. ejem. negativos, $n^{\ominus}(c) =$ núm ejem. negativos cubiertos por hipótesis c, $|N| =$ núm. total de ejemplos, y $|c| =$ núm. literales en el cuerpo de c.

Etimadores de Probabilidades

- Frecuencia relativa: $p(\oplus|c) = \frac{n^{\oplus}(c)}{n(c)}$. Adecuada cuando se cubren muchos ejemplos
- Estimador Laplaciano: $p(\oplus|c) = \frac{n^{\oplus}(c)+1}{n(c)+2}$. Aplica cuando se tienen dos clases y supone que se tiene una distribución uniforme de las dos clases
- Estimador-m: $p(\oplus|c) = \frac{n^{\oplus}(c)+m\times p_a(\oplus)}{n(c)+m}$ donde $p_a(\oplus) = \frac{n^{\oplus}}{n}$, es la probabilidad *a priori* de la clase y m expresa nuestra confianza en la evidencia (ejemplos de entrenamiento). Si $m = 0$ regresamos a frecuencia relativa y si $m = 2$ y $p_a(\oplus) = \frac{1}{2}$ regresamos al estimador Laplaciano.

Miopía

- Los sistemas que usan estrategias de general a específico tienden a tener dificultades con cláusulas que involucran muchas literales y usan una estrategia de búsqueda tipo *hill climbing*
- Para aliviar el problema de la miopía algunos proponen utilizar: (i) *macro-operadores*, (ii) *beam-search*, (iii) *fixed-depth look-ahead* y (iv) *templates*

Restricciones y Técnicas Adicionales

En general todos los sistemas ILP introducen ciertas restricciones para generar sus hipótesis:

- Se le dice al sistema qué argumentos están determinados (argumentos de salida) en un predicado si el resto de los argumentos son conocidos (argumentos de entrada). Se pueden formar gráfos que ligan entradas y salidas guiando la construcción de las hipótesis. Se puede restringir aún más utilizando tipos.
- Considera sólo cláusulas en que todas las variables aparescan por lo menos 2 veces en la cláusula, o introduce una literal con al menos una variable existente (v.g., Foil).

Restricciones y Técnicas Adicionales

- Construye hipótesis sólo de una clase de cláusulas definidas con esquemas o modelos de reglas o gramáticas (v.g., Mobal).
- Construye hipótesis siguiendo un operador de refinamiento particular (v.g., MIS).
- Utiliza predicados adicionales para determinar qué predicados del conocimiento del dominio son relevantes para la hipótesis actual (v.g., Tracy).

Símbolos Funcionales

- Los programas lógicos pueden tener términos complejos usando símbolos funcionales
- Muchos sistemas usan una representación aplanada o *flattened* para eliminar los símbolos funcionales
- Cada término $f(X_1, \ldots, X_n)$ en cada cláusula C de un programa, se cambia por una nueva cláusula con variables X y se añade al cuerpo de C un nuevo predicado $P_f(X_1, \ldots, X_n, X)$ representando la función f.

Proposicionalización

- Idea: transformar un problema relacional en una representación de atributo-valor y usar algoritmos más convencionales como son C4.5 y CN2
- Razones: (i) utilizar algoritmos más eficientes, (ii) contar con una mayor cantidad de opciones de algoritmos, y (iii) utilizar técnicas más maduras
- Se construyen atributos a partir del conocimiento del dominio y de las propiedades estructurales de los individuos.
- Este proceso puede ser *completo* (no pierde información pero puede ser infinito) ó *parcial* (pierde información, es el más utilizado)

Ejemplo

La tabla ilustra cómo se podría expresar con *Linus* el problema de aprender una definición para la relación *hija*, donde $f(X)$ significa *femenino* y $p(X, Y)$ significa *progenitor*.

Vars.		Atributos proposicionales							Cl
X	Y	f(X)	f(Y)	p(X,X)	p(X,Y)	p(Y,X)	p(Y,Y)	X=Y	
f	e	T	F	F	T	F	F	F	$\oplus$
c	g	T	T	F	T	F	F	F	$\oplus$
e	t	F	F	F	F	T	F	F	$\ominus$
v	r	T	F	F	F	F	F	F	$\ominus$

Ejemplo

El resultado sería:

IF f(X) = T AND p(Y,X) = T
THEN Clase = $\oplus$.

lo que se puede transformar de regreso a una representación relacional como sigue:

$hija(X, Y) \leftarrow femenino(X), progenitor(Y, X).$

El problema está en cómo generar un conjunto adecuado de atributos que sean manejables y la incapacidad de inducir definiciones recursivas

ILP y Meta-Interpretes

- El aprender programas recursivos ha sido tradicionalmente un problema en ILP, ya que los primeros sistemas seguían una estrategia *bottom-up*
- Con cada ejemplo se trata de construir la cláusula más específica que cubre al ejemplo y luego trata de generalizar para cubrir otros ejemplos
- La recursividad resurgió con la introducción de *meta-interpretive learning* (MIL) y Metagol
- La idea es usar meta-reglas o templates para restringir la forma de los programas a inducir y con esto, el espacio de hipótesis

Metagol y MIL

- Por ejemplo, las meta-reglas:
  $P(A, B) \leftarrow Q(A, C), R(C, B)$ y
  $P(A, B) \leftarrow Q(A, C), P(C, B)$
- Nos permite inducir cláusulas como:
  $tio(A, B) : -papa(A, C), hermano(C, B)$ y
  $conectado(A, B) : -liga(A, C), conectado(C, B)$

ILP, MIL y Progol

- Recientemente se ha usado un meta-iterprete (*Meta Interpretive Learning* o MIL)
- La idea es definir una serie de *meta-reglas* que se puedan usar para aprender más rápido cláusulas con más expresividad
- Primero se trata de satisfacer la meta, y si no se puede, unificar la meta con la cabeza de una meta-regla y buscar cómo instanciar los argumentos
- Con una substitución prueba las metas de la meta-regla

Meta-Interprete

```
prove([],H,H).
prove([Atom|Atoms],H1,H2):-
    prove_aux(Atom,H1,H3),
    prove(Atoms,H3,H2).
prove_aux(Atom,H,H):-
    call(Atom).
prove_aux(Atom,H1,H2):-
    member(sub(Name,Subs),H1),
    metarule(Name,Subs,(Atom :- Body)),
    prove(Body,H1,H2),
prove_aux(Atom,H1,H2):-
    metarule(Name,Subs,(Atom :- Body)),
    new_metasub(H1,sub(Name,Subs)),
    abduce(H1,H3,sub(Name,Subs)),
    prove(Body,H3,H2).
```

Ejemplo

```
body_pred(mother/2).
body_pred(father/2).
% BK:
mother(ann,amy).
mother(ann,andy).
mother(amy,amelia).
mother(linda,gavin).
father(steve,amy).
father(steve,andy).
father(gavin,amelia).
father(andy,spongebob).
% Metareglas:

metarule([P,Q],[P,A,B],[[Q,A,B]]).
metarule([P,Q,R],[P,A,B],
        [[Q,A,B],[R,A,B]]).
metarule([P,Q,R],[P,A,B],
        [[Q,A,C],[R,C,B]]).
```

```
% Ejemplos positivos:
grandparent(ann,amelia),
grandparent(steve,amelia),
grandparent(ann,spongebob),
grandparent(steve,spongebob),
grandparent(linda,amelia)
% Ejemplos negativos
grandparent(amy,amelia)
% Resultado:

grandparent(A,B):-
        gp_1(A,C), gp_1(C,B).
gp_1(A,B):-mother(A,B).
gp_1(A,B):-father(A,B).
```

Meta-reglas e Invensión de Predicados

- En la invensión de predicados (*predicate invention*) se crean automáticamente nuevos predicados
- Uno de los efectos de esto es la descomposición de problemas en subproblemas que se pueden utilizar en otros dominios
- Por ejemplo, se puede tener una metaregla:
  $P(A, B) \leftarrow Q(A, C), R(C, B)$
- Con esto se puede aprender una regla:
  $f(A, B) : -resto(A, C), resto(C, B)$ que quita dos elementos de una lista
- Si queremos quitar más, podemos usar la misma regla y obtener:
  $f(A, B) : -resto(A, C), f1(C, B)$
  $f1(A, B) : -resto(A, C), resto(C, B)$

Probabilistic Inductive Logic Programming

- En PILP en lugar de ejemplos positivos y negativos, son observados y no observados
- Se consideran dos tareas: Estimación de parámetros y aprendizaje de la estructura
- Se asocian probabilidades a los hechos y a las cláusulas y se toman como opciones estocásticas dentro de resolución
- Hay una gran cantidad de propuestas diferentes de cómo hacerlo: modelos relacionales probabilíticos, modelos de Markov relacionales, redes Bayesianas orientadas a objetos, etc.

Differentiable Inductive Logic Programming

- ∂ILP implementa una deducción diferenciable sobre valores contínuos
- Se implementan probabilididades condicionales de los ejemplos
- Se busca predecir la probabilidad de los ejemplos positivos y negativos usando *log-likelihood* como función de pérdida

Algunas Extensiones Recientes

El uso de representaciones relacionales ha permeado a prácticamente todas las áreas de aprendizaje computacional

- Aprendizaje de árboles de decisión (y de regresión) relacionales, donde cada nodo contiene una conjunción lógica y los nodos dentro de un camino del árbol, pueden compartir variables entre sí
- Definición de una medida de distancia relacional que permite calcular la similaridad entre dos objetos, para aprendizaje basado en instancias relacional y para *clustering*

Algunas Extensiones Recientes

- Aprendizaje de reglas de asociación de primer orden
- Aprendizaje por refuerzo relacional
- Aprendizaje de lenguajes lógicos (LLL) o gramáticas aprovechando la expresividad de la lógica de predicados.
- Combinar ideas de programación lógica inductiva con probabilidad, e.g., implicación probabilísta, aprendizaje basado en interpretaciones probabilísticas, y aprendizaje basado en pruebas lógicas probabilísticas
- Inducir ensambles de clasificadores, tales como *Bagging* y *Boosting* en ILP

Aplicaciones

Aunque no se ha tenido el auge de otras áreas de aprendizaje, ILP ha tenido algunos resultados importantes que son difíciles de obtener con otras técnicas:

- Predicción de relaciones en estructura-actividad, incluyendo la mutagénesis de compuestos moleculares que pueden causar cancer
- Predicción de la estructura tridimensional secundaria de proteinas a partir de su estructura primaria o secuencia de aminoácidos
- Diseño de elemento finito de malla para analizar tensión en estructuras físicas

Aplicaciones

- Aprendizaje de reglas para diagnóstico temprano de enfermedades de reumatismo
- Construcción de programas a partir de especificaciones de alto nivel
- Aprendizaje de reglas de control para sistemas dinámicos a partir de trazas
- Clasificación biológica de calidad de agua de ríos
- Aprendizaje de modelos cualitativos de sistemas dinámicos.

Trabajo Actual

- Determinar relevancia
- Cómo manejar ruido
- Neuro-Symbolic
- Explicabilidad

Retos

- *Data efficiency*: Mientras los métodos simbólicos funcionan bien con pocos datos, las NNs requieren muchos datos, pero si se tienen muchos datos, los simbólicos tienen problemas y las NNs los manejan bien
- Generalización: A veces pueden cambiar sus hipótesis o no aprender un modelo adecaudo si se cambian los datos
- Aplicaciones: En general sus aplicaciones son limitadas
- Explicabilidad: Aunque por su naturaleza sus modelos son explicables, no está claro cómo integrarlos a otros sistemas y mantener su explicabilidad