

Síntesis inductiva de programas lógicos

- Programación automática (o *síntesis de programas*): técnicas para el diseño de componentes software a partir de especificaciones.
- Aproximaciones:
 - **Síntesis deductiva**: se aplica a la especificación transformaciones que preserven el significado hasta alcanzar un algoritmo. Ej: Sato and Tamaki, Lau and Ornaghi, Hogger, ...
 - **Síntesis constructiva**: el algoritmo se extrae desde una prueba constructiva de la satisfacibilidad de la especificación.



síntesis inductiva

Síntesis inductiva de programas lógicos (2)

- Síntesis inductiva
 - Síntesis basada en la traza
 - Síntesis basada en modelos
- Problemas con la síntesis inductiva:
 - la mayoría de aproximaciones están más orientadas al aprendizaje en general y no a la síntesis.
 - los ejemplos son una aproximación de la especificación bastante débil.

Síntesis inductiva de programas lógicos (3)

DIFERENCIAS ENTRE ILP Y SÍNTESIS

	ILP	SÍNTESIS
clase de hipótesis	cualquier descripción	algoritmos
especificador	humano o máquina	humano
concepto intencionado	a veces desconocido	siempre conocido
consistencia ejemplos	cualquier actitud	consistente
número ejemplos	cualquiera	unos pocos
número predicados	al menos 1	exáctamente 1
reglas de inferencia	selectivo & constructivo	necesariamente constructivo
corrección hipótesis	cualquier actitud	total corrección
esquemas	normalmente no	sí
nº hipótesis correctas	usualmente sólo pocas	siempre muchas

Síntesis usando esquemas

■ Plantilla generica:

$$r(X,Y,Z) \leftarrow \underline{c}(X,Y,Z), p(X,Y,Z)$$

$$r(X,Y,Z) \leftarrow \underline{d}(X,H,X_1,\dots,X_t,Y_1,\dots,Y_t,Z), r(X_1,Y_1,Z),\dots, r(X_t,Y_t,Z), \\ q(X,H,X_1,\dots,X_t,Y_1,\dots,Y_t,Z)$$

■ Ejemplos:

- $r(X,Y) \leftarrow \text{primitive}(X), \text{solve}(X,Y)$
- $r(X,Y) \leftarrow \text{nonprimitive}(X), \text{decompose}(X,H,X_1,X_2), \right.$
 $\left. r(X_1,Y_1), r(X_2,Y_2), \text{compose}(H,Y_1,Y_2,Y)$
- $\text{reverse}(L,R) \leftarrow L=[], R=[]$
- $\text{reverse}(L,R) \leftarrow L=[HL|TL], \text{reverse}(TL,TR), \text{compose}(HL,TR,R)$

divide
y
vencerás

Síntesis usando esquemas: algoritmo genérico

generic algorithm

input: E_r (evidencia del predicado r), O_r (oráculo para r), C (un programa en el que sólo r está indefinido)

output: P_r (programa para r que contiene C y que cubre E_r)

sbis(E_r, O_r, C, P_r) $\leftarrow$

 selectSchema(S), *%S es algún esquema apropiado para r (renombrado)*

 close-cd(S, C, V), *% $V = C \cup T \cup \underline{CD}$, T -plantilla de S -, $\underline{CD}$ define $\underline{c}$ y $\underline{d}$*

 abduce(V, E_r, O_r, E_p, E_q). *% E_p (E_q) evidencia clausal de p (q) abducidas*
 % intentado demostrar que V cubre E_r usando O_r

 induce(E_p, E_q, P_p, P_q). *% P_p (P_q) es un programa no recursivo para p (q)*

 acceptable(P_q) $\rightarrow$ *% determinar si la invención de predicados es necesaria*

$P_r = V \cup P_p \cup P_q$ *% si no*

 ; recurse($E_r, E_p, E_q, O_r, V, P_p, P_q, P_r$) *% inventar usando sbis*

Síntesis usando esquemas: CRUSTACEAN

■ CRUSTACEAN:

- Evidencia: literales ground
- No hay restricciones (bias), el lenguaje de hipótesis son cláusulas definidas con la plantilla

$r(\dots) \leftarrow$

$r(\dots) \leftarrow r(\dots)$

- síntesis pasiva y conducida por los datos
- no hay background ni invención de predicados (plantilla)
- 1 predicado (cláusula base **B**+ cláusula recursiva **R**) con manipulación sintáctica.
- no es una instancia del algoritmo genérico

[illegible]

-

$$Pi = \text{last}(a, [c, a]), Bi = \text{last}(a, [a]), R = \text{last}(A, [B, C|T]) \leftarrow \text{last}(A, [C|T])$$

lo que sirve para encontrar B y R

- 1

$\text{last}(a,[c,a]); \text{last}(b,[e,d,b]) \rightarrow \text{lgg}\{\text{last}(a,c), \text{last}(b,d)\} \rightarrow \text{last}(A,B) \leftarrow$
 $\text{lgg}\{\text{last}(a,[c]), \text{last}(b,[d])\} \rightarrow \text{last}(A,[A]) \leftarrow$
 incorrecto
 correcto

Síntesis usando esquemas: CRUSTACEAN (3)

- Cómputo de R: la cabeza se selecciona de las descomposiciones de los ejemplos

$\text{last}(a,[c,a]) \quad \text{last}(b,[e,d,b]) \quad \text{last}(b,[d,b])$

$\text{lbg} = \text{last}(A,[B,C|T])$

El cuerpo se construye usando las anotaciones

$\text{last}(A,[C|T])$

$\text{last}(A,[A]) \leftarrow$

$\text{last}(A,[B,C|T]) \leftarrow \text{last}(A,[C|T])$

} programa final

Síntesis usando esquemas: CILP

- Interactivo, manipulación sintáctica, sin background
- Sigue el algoritmo genérico:
 - E_r : literales básicos
 - O_r : especificador
 - C: programa vacío
 - selectSchema: $r(\dots) \leftarrow$
 $r(\dots) \leftarrow r(\dots), q(\dots)$ } no hay $\underline{c}$, $\underline{d}$, p
 - close- $\underline{cd}$, abduce, induce: juntos en un paso
 - cláusula recursiva: diferencias entre parámetros de los ejemplos seleccionados e inversión del mayor número de pasos de resolución entre los ejemplos.
 $\text{length}([a,b],s^2(0))$ y $\text{length}([a,b,c,d],s^4(0)) \Rightarrow \text{length}([H|T],s(N)) \leftarrow \text{length}(T,N)$
 - cláusula base: lgg de los hechos sin resolver
 $\text{length}([], 0) \Rightarrow \text{length}([],0) \leftarrow$
 - parámetros de q: todas las variables de la cláusula recursiva

Síntesis usando esquemas: CILP (2)

- **acceptable:** invención de predicados si el programa inducido por cada par de ejemplos positivos cubre algún ejemplo negativo
- **recurse:** $\text{sbis}(E_q, \blacksquare, \blacksquare, Q), P_r = V \cup P_p \cup Q$

Síntesis usando esquemas: FORCE2

- Evidencia: literales básicos
- Funciones: complejidad, determinar si un átomo es una instancia del caso base
 - $\text{maxdepth}(\text{append}(X,Y,Z)) = \text{length}(X) + 1$
 - $\text{basecase}(\text{append}(X,Y,Z)) = \text{if } X = [] \text{ then true else false}$
- Lenguaje de hipótesis: programas definidos ij-determinados, con 2 cláusulas lineales y cerradamente recursivas

cota superior

condición pertenencia

$r(\dots) \leftarrow \underline{c}(\dots)$

$r(\dots) \leftarrow \underline{d}(\dots), r(\dots)$

plantilla

BK
ij determinados

sin variables salida

Síntesis usando esquemas: FORCE2 (2)

- 1 predicado, sin invención de predicados
- Background: predicados de aridad j (o menos) y profundidad I .
- pasiva, conducida por los datos.
- Técnica: dividir los ejemplos positivos usando el *basecase* y usar *rlgg*
`append([ ],[1],[1]) append([ ],[2,2],[2,3]) append([1],[ ],[1]) append([1,2],[3],[1,2,3])`



`append(X,Y,Z) ← Y=[W|V], X=[ ], Z=Y (B)`

`append(X,Y,Z) ← X=[W|V], Z=[T|U], W=T (R)`

Síntesis usando esquemas: FORCE2 (3)

- Se eligen átomos recursivos con las variables de R, se prueban los ejemplos positivos y se calculan rlgg

$\text{append}(X,Y,Z) \leftarrow Y=[W|V], X=[], Z=Y \quad (B)$

$\text{append}(X,Y,Z) \leftarrow X=[W|V], Z=[T|U], W=T \quad (R)$

$\text{append}(V,Y,U)$

$e = \text{append}([1,2],[3],[1,2,3]) \Rightarrow$ no caso base $\Rightarrow R$ no se generaliza ya que cubre e

$i1 = \text{append}([2],[3],[2,3]) \Rightarrow$ no caso base $\Rightarrow \text{lgg}(R,i1)=R$

...

$\text{in} = \text{append}([],[],[]) \Rightarrow$ caso base $\Rightarrow \text{lgg}(B,\text{in}) = \text{append}(A,B,C) \leftarrow A=[], C=B$

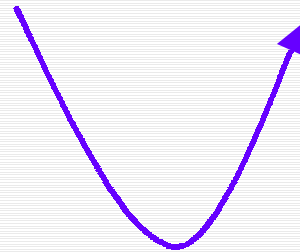
Síntesis usando esquemas: FORCE2 (4)

■ Eliminación de errores

recursión infinita  maxdepth

otros  ejemplos negativos

append(X,X,Z)  append([1,2],[3],[1,2,3])



Síntesis usando esquemas

- Otros sistemas:

SIERES

TIM

SYNAPSE

DIALOGS

METAINDUCE

Síntesis sin esquemas: SPECTRE II

- Entrada:
 - Evidencia: literales básicos seleccionados (E)
 - Una teoría inicial recursiva bastante general (T).
- Múltiples predicados
- no background, no bias, no invención predicados
- Lenguaje de hipótesis: programas definidos
- Técnica:
 - asunciones: $T \models E$, n° finito de refutaciones que no repiten secuencia de cláusulas
 - Primero: cláusulas usadas en refutaciones de ejemplos positivos y negativos $\Rightarrow$ desplegar un átomo
 - Segundo: cláusulas usadas en refutaciones de ejemplos negativos y no usadas en los positivos $\Rightarrow$ eliminar cláusula

Síntesis sin esquemas: SPECTRE II (2)

Evidencia positiva: $\text{odd}(s(0))$, $\text{odd}(s^3(0))$, $\text{odd}(s^5(0))$

Evidencia negativa: $\text{odd}(0)$, $\text{odd}(s^2(0))$, $\text{odd}(s^4(0))$

Teoría inicial: $\text{odd}(0) \leftarrow (c1)$

$\text{odd}(s(X)) \leftarrow \text{odd}(X) \quad (c2)$

refutación $\text{odd}(0)$

paso 1

$c1$

refutaciones E^+

desplegar $c2$

$\text{odd}(0) \leftarrow (c1)$
 $\text{odd}(s(0)) \leftarrow (c3)$
 $\text{odd}(s^2(X)) \leftarrow \text{odd}(X) \quad (c4)$

eliminar

paso 2

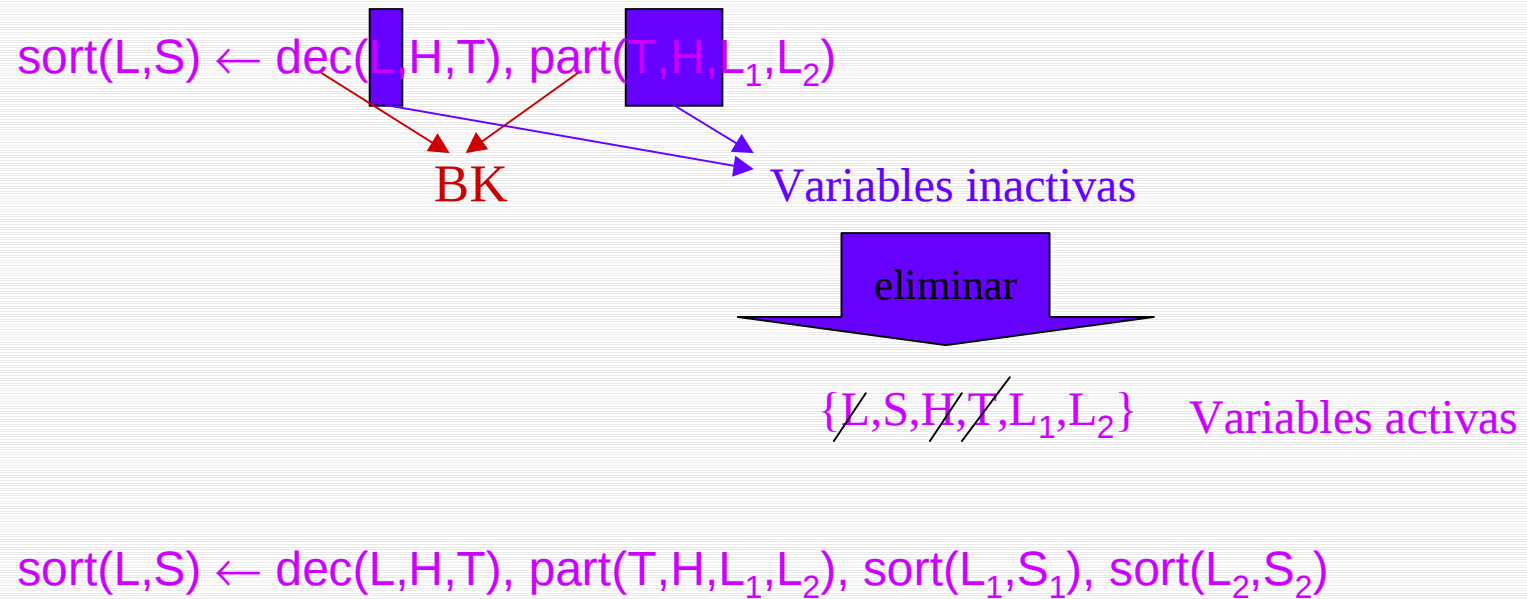
$c1$

$\text{odd}(s(0)) \leftarrow (c3)$
 $\text{odd}(s^2(X)) \leftarrow \text{odd}(X) \quad (c4)$

Síntesis sin esquemas: SMART

- Evidencia: literales básicos de un predicado
- Hipótesis: programas definidos
- Background con información de tipos
- Bias de búsqueda: cota superior longitud de las cláusulas
- Primer paso: inducir la cláusula base generando sus propios ejemplos negativos
- Segundo paso: se induce la cláusula recursiva enumerando top-down las cláusulas candidatas y seleccionando una correcta con respecto a los ejemplos

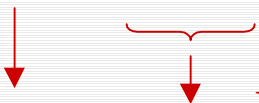
Síntesis sin esquemas: SMART(2)



Síntesis sin esquemas: SKILIT

- Evidencia: literales básicos de un predicado
- Declaraciones de tipo y modo
- Bias de búsqueda: sketches de algoritmos

$$E \leftarrow L_1, \dots, L_m$$


ejemplo Si no hay sketch $\Rightarrow$ sketch de caja negra
Literales básicos con predicados del BK o de la forma \$p

$$r(t_1, \dots, t_n) \leftarrow \$p(t_1, \dots, t_n)$$

- La técnica comienza con el programa vacío y añade una cláusula (refinamiento de sketches de algoritmos)


Reemplazar \$p por predicados del BK o de la evidencia

Síntesis sin esquemas: SKILIT (2)

sort([],[]), sort([3,2,1],[1,2,3])

sort([3,2],[3,2]), sort([],[1])

sort([],[]) ← \$p1([]), \$p2([])

sort([3,2,1],[1,2,3]) ← sort([2,1],[1,2]), \$p3(3,[1,2],[1,2,3])

} sketches

BK: insert, null

\$p1=null (BK |= null([])

\$p2=null

\$p3=insert (BK |= insert(3,[1,2],[1,2,3]))

sort(L,S) ← null(L), null(S)

sort([H|T],S) ← sort(T,Y), insert(H,Y,S)

} Programa correcto

refinar

Síntesis con técnicas de propósito general

■ MIS

- Interactivo, incremental
- Evidencia: literales básicos sobre múltiples predicados
- Bias de búsqueda: Información de tipo, modo y determinismo
- Bias del lenguaje: lista de predicados relevantes del BK
- Inicio: teoría vacía P

repeat

 leer siguiente ejemplo

 repeat

 If P no completo entonces generar cláusula para el ejemplo

 If P no consistente entonces eliminar cláusula

 hasta P completo y consistente

hasta siempre ejemplos

Síntesis con técnicas de propósito general (2)

M: declarar el predicado a inducir

x es un término

S: $\text{member}(+x, +[x])$

M: posibles predicados usados por member?

programa recursivo

S: member/2

M: P0=vacío. Primer ejemplo?

inicio

S: $\text{member}(a, [a])$.

M: $P1 = \{\text{member}(X, [Y|T]) \leftarrow \}$. Siguiendo ejemplo?

P0 incompleto/ generación P1

S: $\neg \text{member}(a, [b, c])$

M: Is $\text{member}(a, [])$?

P1 inconsistente/mas información

S: no

M: $P2 = \{\text{member}(X, [Y]) \leftarrow \}$. Siguiendo ejemplo?

se genera P2

S: $\neg \text{member}(a, [b])$

M: $P3 = \{\text{member}(X, [X|T]) \leftarrow \}$. Siguiendo ejemplo?

P2 inconsistente/generación P3

S: $\text{member}(c, [d, c])$.

M: Is $\text{member}(d, [c])$?

P3 incompleto/más información

S: no

M: $P4 = \{\text{member}(X, [X|T]) \leftarrow, \text{member}(X, [Y|T]) \leftarrow \text{member}(X, T)\}$.

generación P4
fin

Síntesis con técnicas de propósito general (3)

■ FILP:

- Evidencia: átomos básicos de múltiples predicados
- Invención de predicados
- Interactivo (si en el proceso de generación se pierde algún ejemplo positivo se hacen preguntas)
- Hipótesis: programas definidos
- Background: programas definidos con modo
- Bias del lenguaje: conjuntos de cláusulas
- Técnica: bucle de generación de cláusulas. A cada iteración se añade un literal en el cuerpo de la cláusula a especializar (la que cubra menos ejemplos negativos). Backtracking si la cláusula no cubre ningún ejemplo positivo.

Síntesis con técnicas de propósito general (4)

reverse([],[]), reverse([a],[a]), reverse([a,b],[b,a]), reverse([a,b,c],[c,b,a])

BK: append

Bias: reverse(X,Y) $\leftarrow$ {X= [], Y=[], X=[H|T],...,reverse(T,W),...,append(W,[H],Y),...}

Modos: append(in,in,out)
reverse(in,out)

cláusula inicial: reverse(X,Y) $\leftarrow$

computado heurísticamente
cubre $\neg$ reverse([a],[])

reverse(X,Y) $\leftarrow$ Y= []

no cubre ningún ejemplo
positivo/ backtracking

reverse(X,Y) $\leftarrow$ Y= [], X=[H|T]

se añade otro literal
cubre reverse([],[])/elimina

reverse(X,Y) $\leftarrow$ Y= [], X=[]

computado
igual

.....

reverse(X,Y) $\leftarrow$ X=[H|T], reverse(T,W), append(W,[H],Y).

Software predictivo

- Visión clásica de la Ingeniería del software:
 - “De la especificación al producto final”
- Ingeniería de requerimientos:
 - “De las necesidades a la especificación”
- Sin embargo esto no es igual a:
 - “De las necesidades al producto final”
- Problemas:
 - Los requerimientos deben revisarse continuamente.
 - Algunos problemas no tienen una especificación completa y estándar (*e.g. reconocimiento de caracteres, muchas tareas de clasificación,...*)

Software predictivo (2)

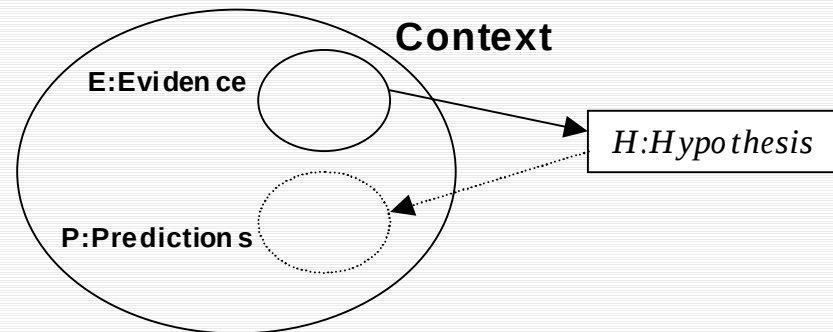
- Minimizar el coste de mantenimiento :
 $f(\text{modificabilidad, probabilidad de modificación})$
- Modificabilidad ha sido direccionada por:
 - Tecnología de evolución del sw: modularidad, encapsulación, polimorfismo...
 - *Nuevas tendencias*: Software inteligente y adaptativo.
- Probabilidad de modificación ha sido menos estudiada desde un punto de vista de la ingeniería del SW .

Software predictivo (3)

- Sin embargo, esta cuestión ha sido tratada por las ciencias de la inducción:

- Machine Learning (ML).
- Filosofía de la ciencia .

- *Inducción:*



- Especialmente para la selección de hipótesis.

Software predictivo (4)

■ **Background desde ML & Ph. Sc.**

– Evaluación de hipótesisHypotheses Evaluation:

- verisimilitud (consistencia con E).
- predecibilidad (extrapolabilidad de P).

■ Algunos criterios de evaluación:

- Aproximaciones Bayesianas
- Principio de la descripción mínima (MDL).
- Coherencia, o consilencia.

■ *Si la evidencia es incremental:*

– *H debe revisarse si:*

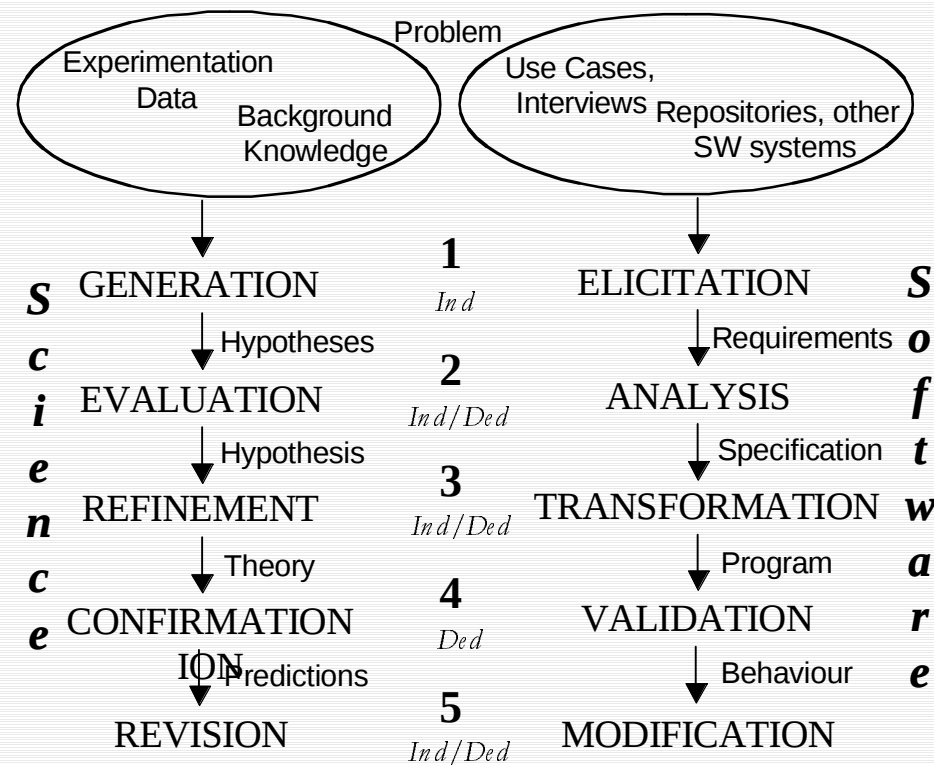
- anomalías: nueva evidencia es erróneamente cubierta.
- novedades: nueva evidencia no es cubierta.

Journal Pre-proof

<i>Ciencia</i>		<i>Programación</i>
reality	↔	requirements context
problem	↔	problem
experimentation data	↔	cases / interviews / scenarios
construed evidence	↔	requirements
evaluation	↔	analysis
best hypothesis	↔	specification
validated subhypothesis	↔	prototype
refinement	↔	transformation
theory	↔	program
verisimilitude	↔	correctness
anomalies	↔	exceptions
confirmatory experiments	↔	testing
confirmation	↔	validation
revision	↔	modification
background knowledge	↔	SW. repositories / components
technical books	↔	technical/programmer's doc.
science text books	↔	user documentation

Software predictivo (6)

- Inducción y deducción tienen el mismo papel en cada lado de la analogía



Software predictivo (7)

- La calidad del sw se evalúa asumiendo que la especificación es perfecta (nunca ocurre).
- Predictividad del Software

Predictividad es el grado con el que el sistema predice requerimientos presentes o futuros en el contexto en el que se han originado los requerimientos

Software predictivo (8)

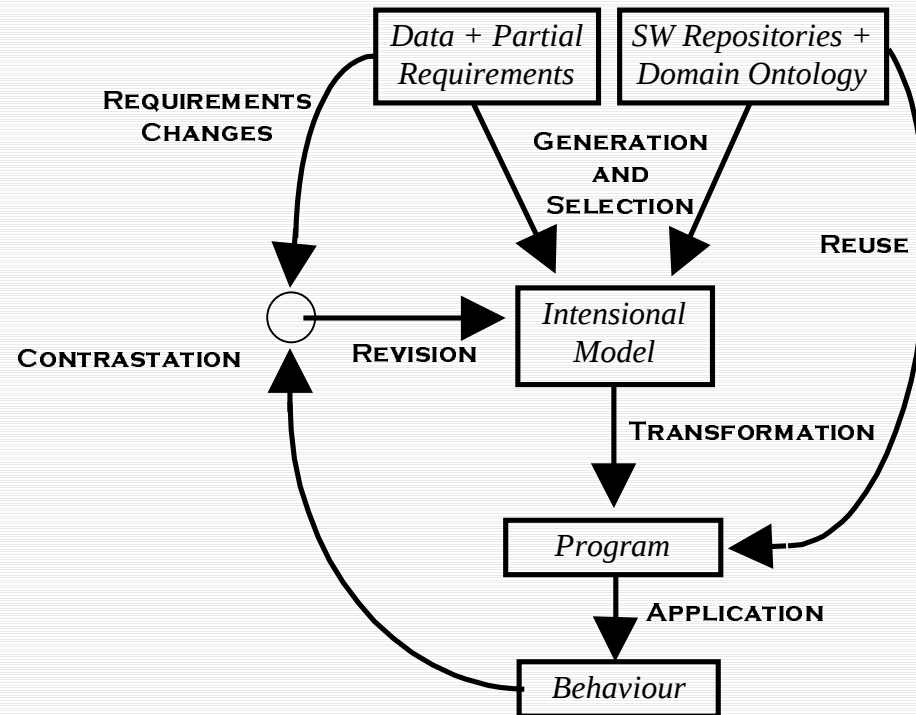
- **Desarrollo del Software como una sesión de aprendizaje incremental**

- Los nuevos sistemas de software *adaptivo* e *inteligente* incluyen técnicas de revisión de ML y razonamiento no monótono. Después de cada error el sistema se modifica a sí mismo (revisa su modelo)

- La relevancia se pone en las fases inductivas (generación y selección) y en la revisión.

Software predictivo (9)

■ Ciclo de vida predictivo



Software predictivo (10)

- La automatización del desarrollo del sw recae sobre la automatización de la inducción.
- 👍 VENTAJAS:
 - La selección de hipótesis puede hacerse automáticamente. Los criterios de evaluación pueden aplicarse en la etapa de análisis.
- 👎 INCONVENIENTES:
 - La generación de hipótesis ha sido estudiada por ML pero ML no está preparada para abordar los problemas complejos que se tratan en IS .

Software predictivo (11)

■ **Hacia la automatización:** dos posibles formas

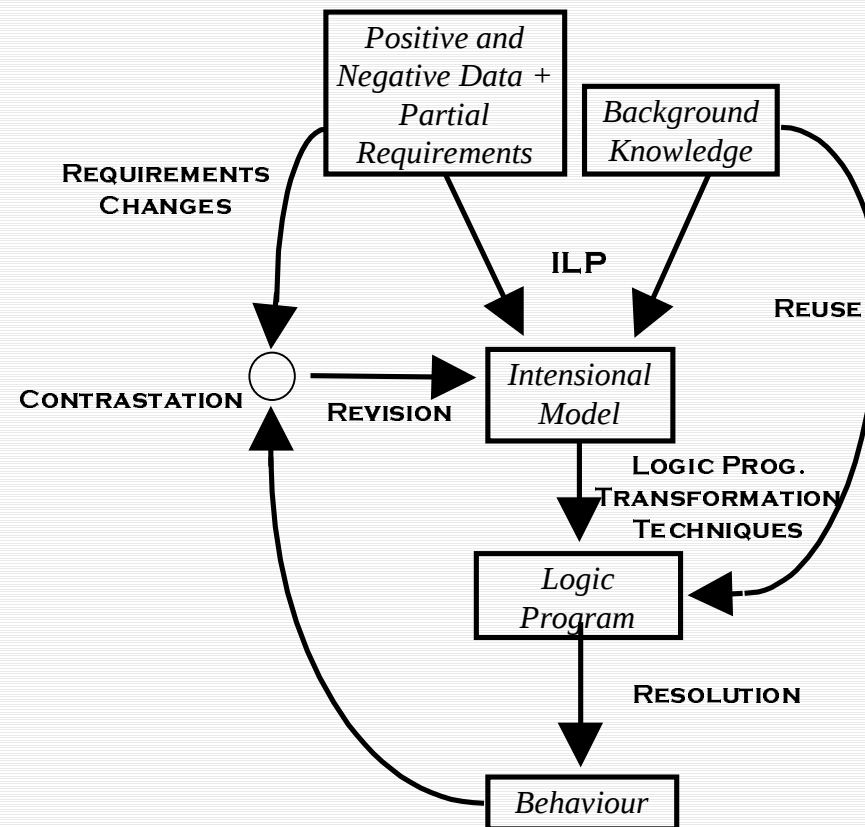
- Desarrollo completamente automático de sistemas simples. Sólo para lenguajes de descripción para los que las fases previas hayan sido automatizadas (generación, transformación, revisión).
- Incrementar el grado de automatización de sistemas complejos. Sólo para lenguajes de descripción comprensibles.

■ Consecuentemente, sólo para lenguajes declarativos (para los que la inducción ha sido desarrollada).

■ Actualmente, programación lógico (y lógico-funcional).

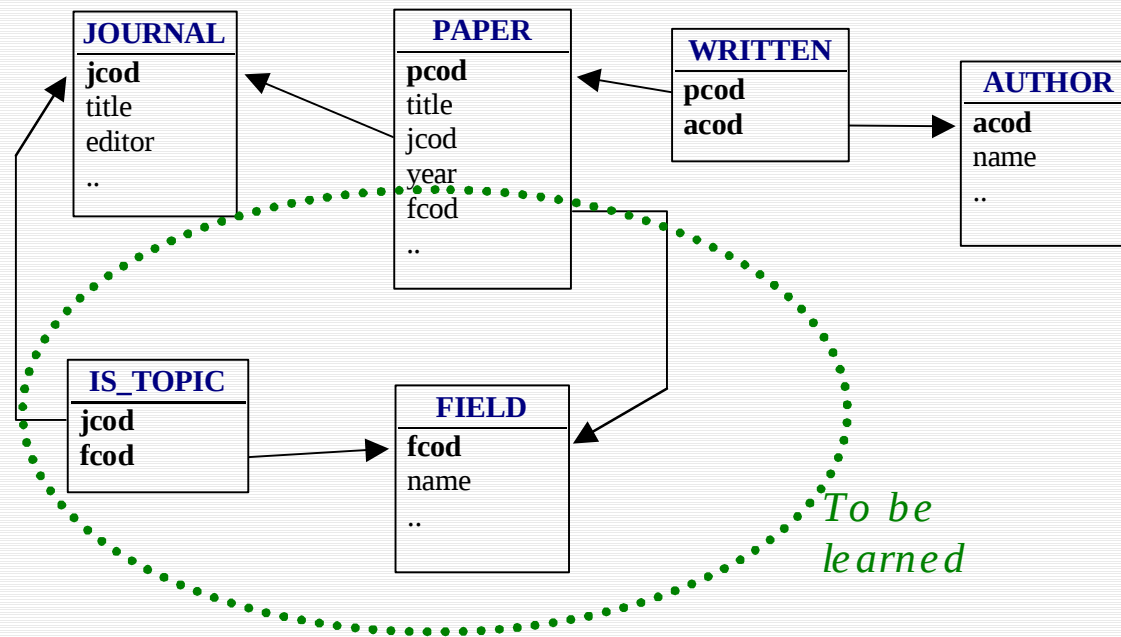
Software predictivo (11)

■ Ciclo de vida predictivo para programación lógica



Software predictivo: ejemplo de comparación de modelos

■ Librería universitaria



Desarrollar un nuevo sistema para clasificar nuevos artículos y revistas para permitir preguntas sobre la materia a la que pertenecen.

Software predictivo: ejemplo de comparación de modelos (2)

J.jcod	J.title	P.pcod	Authors	F.fcod	F.name
ASE	Automated Software Eng.	1	Penix; Alexander	SW	Software Engineering
ASE	Automated Software Eng.	2	Chatzoglou; M.	SW	Software Engineering
ASE	Automated Software Eng.	3	Cousot; Cousot	LP	Logic Programming
JLP	Journal of Logic Prog.	4	Muggleton; DeRaedt	ML	Machine Learning
JLP	Journal of Logic Prog.	5	Lloyd	LP	Logic Programming
JLP	Journal of Logic Prog.	6	Sannella; Wallen	LP	Logic Programming
JLP	Journal of Logic Prog.	7	Cousot; Cousot	LP	Logic Programming
AAI	Applied Artificial Intelligence	8	Blockeel; DeRaedt	DM	Data Mining
TCS	Theo. Computer Science	9	Roscoe; Hoare	SW	Software Engineering
MM	Minds and Machines	10	Fetzer	SW	Software Engineering
DKE	Data & Knowledge Eng.	11	López; Armengol	ML	Machine Learning
CACM	Comm. of the ACM	12	Valiant	ML	Machine Learning
CACM	Comm. of the ACM	13	Hoare	SW	Software Engineering
CACM	Comm. of the ACM	14	Fayyad; Uthurusamy	DM	Data Mining
CACM	Comm. of the ACM	15	Fetzer	SW	Software Engineering
CACM	Comm. of the ACM	16	Genesereth; Ketchpe	SW	Software Engineering
CACM	Comm. of the ACM	17	Muggleton	ML	Machine Learning
DEB	Data Engineering Bulletin	18	Fayyad	DM	Data Mining
NGC	New Generation Computing	19	Muggleton	ML	Machine Learning
MLJ	Machine Learning Journal	20	López de Mántaras	ML	Machine Learning
MLJ	Machine Learning Journal	21	Muggleton	ML	Machine Learning
MLJ	Machine Learning Journal	22	Angluin	ML	Machine Learning
AI	Artificial Intelligence	23	DeRaedt; Dehaspe	ML	Machine Learning

Software predictivo: ejemplo de comparación de modelos (3)

- La primera relación a aprender es `is_topic`, que se extrae de `%topics of journals`
`is_topic(J, F) :- paper(_, _, J, _, F).`
los tópicos de una revista son todos los tópicos de sus artículos
- La segunda relación a aprender es `paper`, cuyo atributo 'fcod' representa la materia de un artículo.
- Sin embargo, podemos considerar varios modelos como hipótesis.
- Seleccionar el más predictivo.

Software predictivo: ejemplo de comparación de modelos (4)

```
%%%%%%Modelo A
%%%%%%%%%%%%% Learnt rules
%Muggleton's articles are about ML.
paper_(P, _, _, 'ML'):- written(P,A),author(A,'Muggleton'), !.
% 'JLP' papers field is 'LP' but Muggleton's articles handled above:
paper_(P, _, 'JLP', _, 'LP').
% MLJ articles are always about ML.
paper_(P, _, 'MLJ', _, 'ML').
% Valiant's articles are about ML.
paper_(P, _, _, 'ML'):- written(P, A), author(A, 'Valiant'), !.
% Fayyad's articles are about DM.
paper_(P, _, _, 'DM'):- written(P, A), author(A, 'Fayyad'), !.
% 'CACM' papers field is 'SW' but Valiant's, Fayyad's and Muggleton's:
paper_(P, _, 'CACM', _, 'SW').
```

Software predictivo: ejemplo de comparación de modelos (5)

```
%%%%%%%%%%%%%% Exceptions // non-predictable
```

```
paper_(1, _, _, _, 'SW').
```

```
paper_(2, _, _, _, 'SW').
```

```
paper_(3, _, _, _, 'LP').
```

```
paper_(8, _, _, _, 'DM').
```

```
paper_(9, _, _, _, 'SW').
```

```
paper_(10, _, _, _, 'SW').
```

```
paper_(11, _, _, _, 'ML').
```

```
paper_(23, _, _, _, 'ML').
```

Software predictivo: ejemplo de comparación de modelos (6)

```
%%%%%%%%Modelo B
%%%%%%%%%% Learnt rules ordered by priority (strength)
%   dependency between authors' other papers field:
paper3(P,_,J,_,F):- written(P,A), written(P2,A), paper(P2,_,J2,_,F),
                    written(P,A2), written(P3,A2), paper(P3,_,J3,_,F),
                    P <> P2, P <> P3, A <> A2.

%   dependency between author's other paper field:
paper2(P,_,J,_,F):- written(P,A), written(P2,A), paper(P2,_,J2,_,F).
%   dependency with journal field:
paper1(P, _, J, _, F) :- is_topic(J, F).
%   Unclassified papers follow these rules:
%   'JLP' papers field is 'LP':
paper0(P, _, 'JLP', _, 'LP').
%   'ASE' papers field is 'SW':
paper0(P, _, 'ASE', _, 'SW').
```

Software predictivo: ejemplo de comparación de modelos (7)

%%%%%%%% uniqueness restrictions for the field of a paper

```
paper_u1(P,_,_,_):- paper1(P,_,_,F1), paper1(P,_,_,F2),  
                    F1 != F2, !, fail.
```

```
paper_u1(P,_,_,F):- paper1(P,_,_,F).
```

```
paper_u2(P,_,_,_):- paper2(P,_,_,F1), paper2(P,_,_,F2),  
                    F1 != F2, !, fail.
```

```
paper_u2(P,_,_,F):- paper2(P,_,_,F).
```

```
paper_u3(P,_,_,_):- paper3(P,_,_,F1), paper3(P,_,_,F2),  
                    F1 != F2, !, fail.
```

```
paper_u3(P,_,_,F):- paper3(P,_,_,F).
```

%%%%%%%% priorities

```
paper_(P,_,_,F) :- paper_u3(P,_,_,F), !.
```

```
paper_(P,_,_,F) :- paper_u2(P,_,_,F), !.
```

```
paper_(P,_,_,F) :- paper_u1(P,_,_,F), !.
```

```
paper_(P,_,_,F) :- paper0(P,_,_,F), !.
```

Software predictivo: ejemplo de comparación de modelos (8)

- El Modelo B es más largo pero tiene menos excepciones.
- El modelo B es más predictivo.
- En muchos casos, puede ser que la inducción del modelo no pueda hacerse de forma completamente automática.
- La evaluación de los modelos puede hacer automáticamente.