Aplicaciones de ILP
Ingeniería del Software

Mª José Ramírez

2000/2001

El proceso del software

- Análisis, diseño, implementación, prueba (depuración) y mantenimiento.

- Reutilización del software.
- Automatización: prueba

El proceso del Software

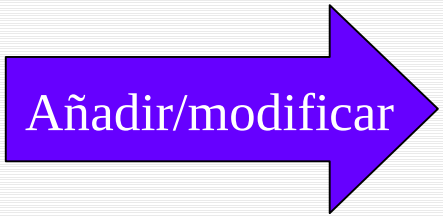
- ILP es una aproximación alternativa a la Ingeniería del Software (=> paradigma de programación lógica)
  - desarrollo
  - mantenimiento
  - reutilización
  - depuración
- Uno de los objetivos de la ILP es la síntesis de programas lógicos.
- Restricciones y ejemplos.

El proceso del software

- El desarrollo de software en un lenguaje de programación lógica inductivo es un proceso incremental que unifica varias etapas clásicas: programación, depuración, mantenimiento y re-ingeniería.

Detección de salidas erróneas

Salida diferente



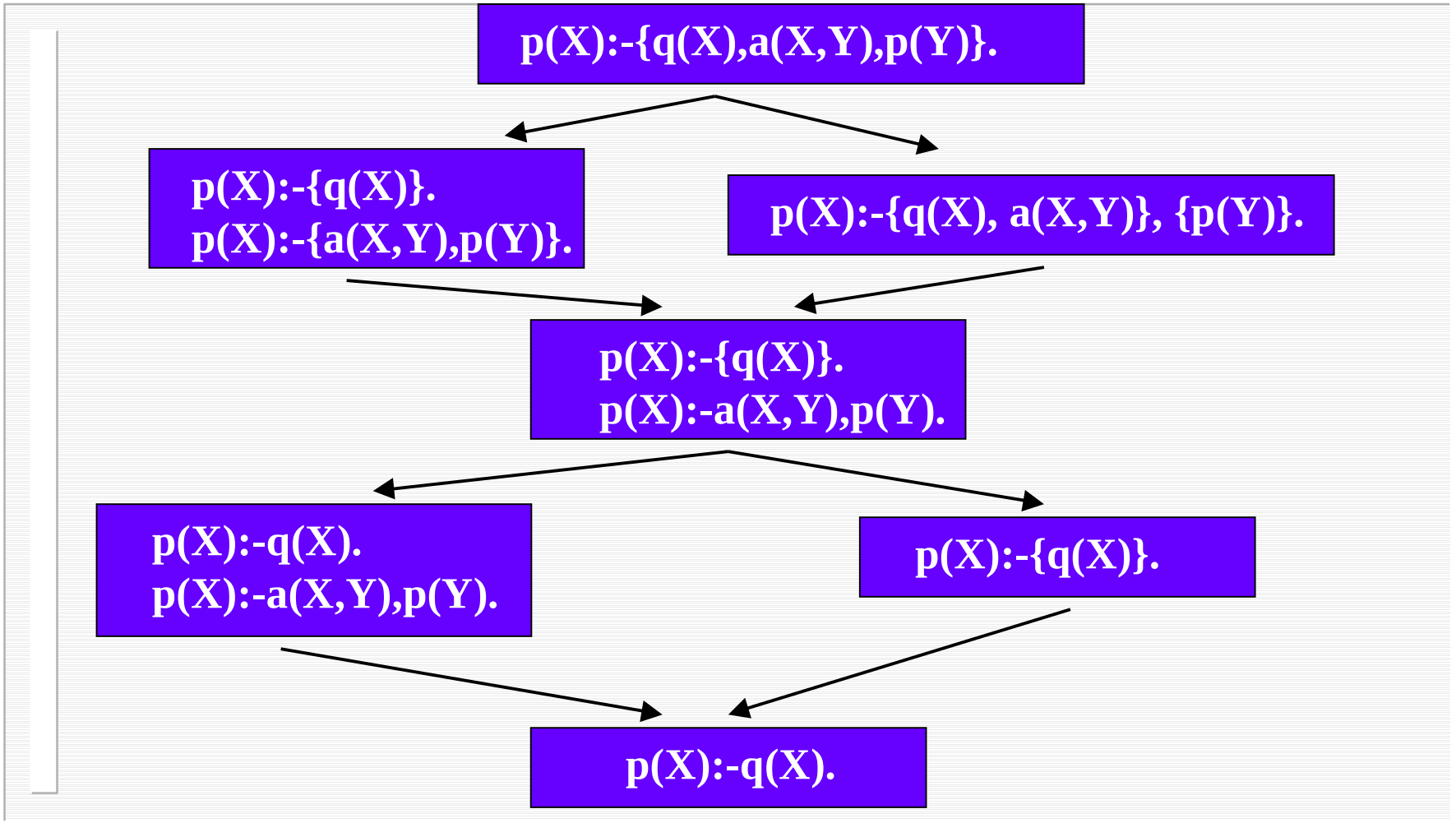
Nuevos ejemplos

Clause Sets

- Bias de lenguaje
- *P* =(Cláusulas-conocidas,Cláusulas-posibles)
- P ∈ *P* si P= cláusulas-conocidas ∪ P1 y P1⊆ Cláusulas-posibles
- *Clause Sets*: Programa Prolog Entre llaves
- Ejemplo:

member(X,[X|_]).
{member(X,[Y|Z]):-cons(X,W,Z).}
cons(X,Y,[X|Y]).
{member(X,[Y|Z]):-X≠Y, member(X,Z).
{member(X,[Y|Z]):-cons(Y,Z,W).
{member(X,[Y|Z]):- member(X, Z).}

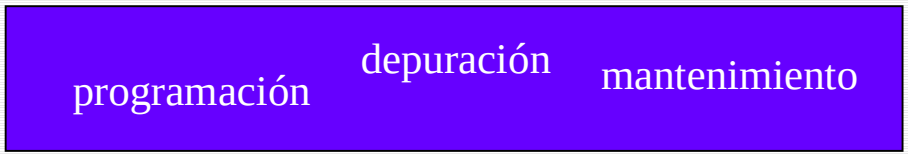
Lenguajes de programación lógica inductivos



El Proceso del software inductivo

- Un programa inductivo define un conjunto de computaciones.
- Inferencia inductiva ≡ Compilación
  - Selección de un programa objeto tal que
    1. Cumple las restricciones
    2. Consistente con los ejemplos
- El desarrollo del software es un proceso que no termina nunca.
- Programa inductivo: inicio ➔restricciones + ejemplos (+,-)
  - Errores se añaden a los ejemplos
  - Si falla la compilación ⟶ relajar/reforzar las restricciones

PROCESO DE DESARROLLO



EJEMPLO: intersección

```
{
int(X,Y,Z):-{null({X,Z}),head(X,X1),tail(X,X2),
                    member(X1,Y),head(Y,Y1),member(Y1,X),
                    int(X2,Y,{Z,W}),cons(X1,W,Z)}.
}
```

```
head([X|_],X).
tail([_|X],X).
cons(X,Y,[X|Y]).
null([ ]).
member(X,[X|_]).
member(X,[_|Y]):-member(X,Y).
```

```
int+([ ],[a],[ ]), int+([a],[a],[a]), int-([ ],[a],[a]).
```

EJEMPLO: intersección (2)

```
int(X,Y,Z):-tail(X,X2).
int(X,Y,Z):-null(Z).
```

```
?- int([a],[ ],[a]).
yes
?- int([a],[a],[ ]).
yes
```

Ejemplos negativos

```
int(X,Y,Z):- null(X),null(Z).
int(X,Y,Z):- head(X,X1), tail(X,X2), int(X2,Y,W), cons(X1,W,Z).
```

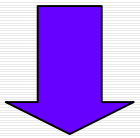
```
?- int([ ],[a,b],[ ]).
yes
?- int([a],[a,b],[a]).
yes
?- int([ ],[a,b],[a]).
no
?- int([b,a],[a,b],[a]).
no
```

```
?- int([b],[a],[ ]).
no
```

Ejemplo positivo

EJEMPLO: intersección (3)

La compilación falla ➪ int+([b],[a],[ ]).



Añadir **notmember**

```
int(X,Y,Z):- null(X),null(Z).
int(X,Y,Z):- head(X,X1), null(Z), notmember(X1,Y).
int(X,Y,Z):- head(X,X1), tail(X,X2), int(X2,Y,W), cons(X1,W,Z).
```

```
?- int([b,a],[a],[a]).
no
```

Ejemplo positivo

```
?- int([b],[a],[b]).
yes
?- int([b,a],[a],[ ]).
yes
```

Ejemplos negativos

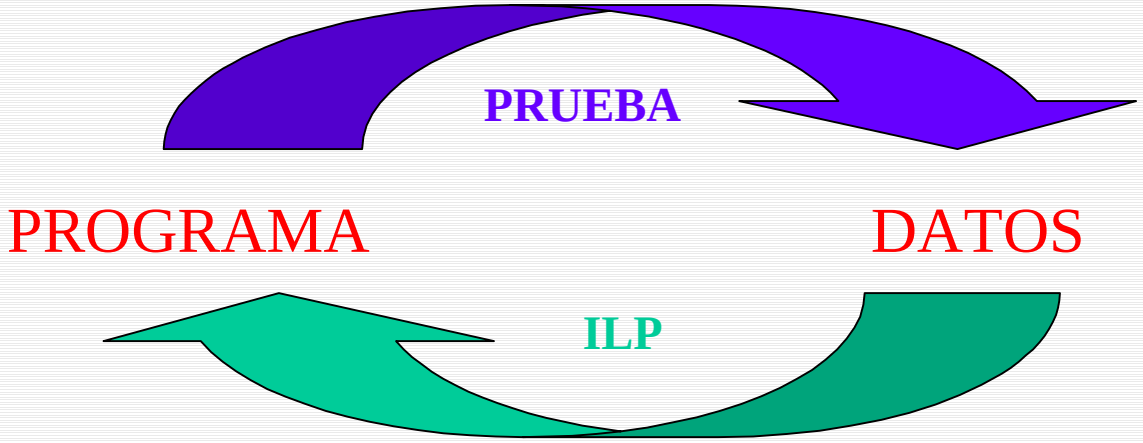
Mantenimiento y Reutilización de SW

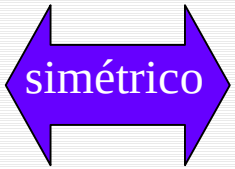
- Aplicaciones similares pueden obtenerse a partir de otras modificando los datos.
- Ejemplo: diferencia de conjuntos.

Mismas restricciones que intersección

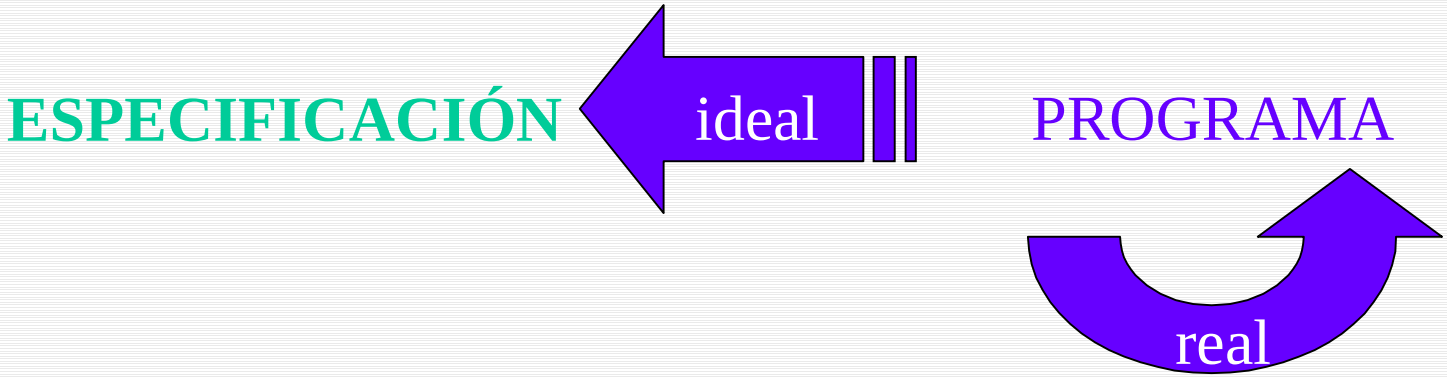
Ejemplos: sd+([b],[a],[b]), sd+([ ],[a],[ ]), sd+([a,b],[a],[b]),
sd-([a],[a],[a]), sd-([ ],[a],[a]), sd-([b],[a],[ ]).

Prueba de programas



- La prueba de programas puede verse como una forma de distinguir un programa de entre todas las alternativas (similar ILP).
- Prueba de programas  ILP

Prueba de programas (2)



- Dado un conjunto test T con valores de entrada para un programa P, los ejemplos de P para T se definen como

$$E(P,T) = \{\langle i,o\rangle \mid i \in T \ \wedge P(i) = o\}$$

- Un conjunto test T es <u>adecuado</u> para probar que un programa P satisface una especificación S sii P_I se infiere inductivamente desde E(P,T) y $P_I \equiv P \equiv S$. Si sólo $P_I \equiv P$ es cierto se dice que T es adecuado al programa y si sólo $P \equiv S$ es cierto, se dice que T es adecuado a la especificación.

Prueba de programas: generación inductiva de casos de prueba.

Procedure generacion casos de prueba
Input: P, P
Output: T
$T \leftarrow \varnothing$
loop:
$\langle P', T \rangle \leftarrow ILPS(E(T,P), P)$
if $P' = fail$ **then return** T
if $(\exists i)\ P'(i) \neq P(i)$ **then** $T \leftarrow T \cup \{i\}$
else $P \leftarrow P - P'$
go to loop

i encontrado aleatoriamente por enumeración

Prueba de programas: un ejemplo (*merge*).

P:

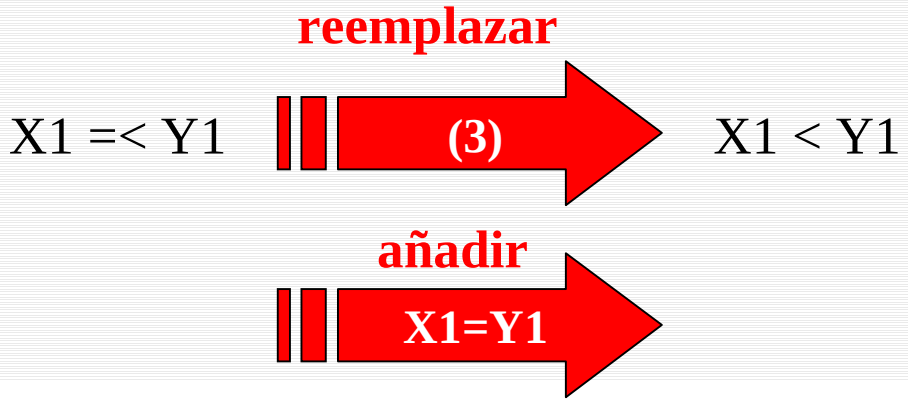
(1) merge(X,Y,Z):-null(X),Y=Z.
(2) merge(X,Y,Z):-null(Y),X=Z.
(3) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(X,X2),X1=<Y1,
merge(X2,Y,W),cons(X1,W,Z).
(4) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(Y,Y2),X1>Y1,
merge(X,Y2,W),cons(Y1,W,Z).

Prueba de programas: un ejemplo (*merge*).

P:

(1) merge(X,Y,Z):-null(X),Y=Z.
(2) merge(X,Y,Z):-null(Y),X=Z.
(3) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(X,X2),X1=<Y1,
merge(X2,Y,W),cons(X1,W,Z).
(4) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(Y,Y2),X1>Y1,
merge(X,Y2,W),cons(Y1,W,Z).

INCORRECTO



Prueba de programas: un ejemplo (*merge*).

P :

```
{
merge(X,Y,Z):-{null(X),null(Y),X=Z,Y=Z,
                    head(X,X1),tail(X,X2), head(Y,Y1),tail(Y,Y2),
                    X1<Y1, X1=Y1,X1>Y1, X1=<Y1, X1>=Y1,
                    merge(X2,Y,W), merge(X,Y2,W), W=Z,
                    cons(X1,W,Z),cons(Y1,W,Z)}
}
```

Prueba de programas: un ejemplo (*merge*).

$\mathbf{T_0} = \varnothing$ $\mathbf{P_0} = \varnothing$

P_0 |- merge(X,Y,Z') P |- merge(X,Y,Z) y Z≠ Z'⇒ ⟨X,Y⟩= ⟨[ ],[ ]⟩
(Z=[ ], Z' no existe)

P:

```
(1) merge(X,Y,Z):-null(X),Y=Z.
(2) merge(X,Y,Z):-null(Y),X=Z.
(3) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(X,X2),X1=<Y1,
                  merge(X2,Y,W),cons(X1,W,Z).
(4) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(Y,Y2),X1>Y1,
                  merge(X,Y2,W),cons(Y1,W,Z).
```

Prueba de programas: un ejemplo (*merge*).

T_1 = {⟨[ ],[ ]⟩} **P_1 = {merge(X,Y,Z):-X=Z.}**

P_1 |- merge(X,Y,Z') P |- merge(X,Y,Z) y Z≠ Z'⇒ ⟨X,Y⟩= ⟨[ ],[1]⟩
(Z=[1], Z'=[ ])

P:

```
(1) merge(X,Y,Z):-null(X),Y=Z.
(2) merge(X,Y,Z):-null(Y),X=Z.
(3) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(X,X2),X1=<Y1,
                  merge(X2,Y,W),cons(X1,W,Z).
(4) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(Y,Y2),X1>Y1,
                  merge(X,Y2,W),cons(Y1,W,Z).
```

Prueba de programas: un ejemplo (*merge*).

$T_2 = T_1 \cup \{\langle [\,],[1]\rangle\}$ **P_2 = {merge(X,Y,Z):-Y=Z.}**

P_2 |- merge(X,Y,Z') P |- merge(X,Y,Z) y Z≠ Z'⇒ ⟨X,Y⟩= ⟨[1],[ ]⟩
(Z=[1], Z'=[ ])

P:

```
(1) merge(X,Y,Z):-null(X),Y=Z.
(2) merge(X,Y,Z):-null(Y),X=Z.
(3) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(X,X2),X1=<Y1,
                  merge(X2,Y,W),cons(X1,W,Z).
(4) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(Y,Y2),X1>Y1,
                  merge(X,Y2,W),cons(Y1,W,Z).
```

Prueba de programas: un ejemplo (*merge*).

$T_3 = T_2 \cup \{\langle[1],[\ ]\rangle\}$

P_3 = {merge(X,Y,Z):-null(X),Y=Z.
merge(X,Y,Z):-head(X,X1),X=Z.
merge(X,Y,Z):-head(Y,Y1),Y=Z.}

P_3 |- merge(X,Y,Z') P |- merge(X,Y,Z) y Z≠ Z'⇒ ⟨X,Y⟩= ⟨[1],[2]⟩
(Z=[1,2], Z'=[1])

P:
(1) merge(X,Y,Z):-null(X),Y=Z.
(2) merge(X,Y,Z):-null(Y),X=Z.
(3) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(X,X2),X1=<Y1,
merge(X2,Y,W),cons(X1,W,Z).
(4) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(Y,Y2),X1>Y1,
merge(X,Y2,W),cons(Y1,W,Z).

Prueba de programas: un ejemplo (*merge*).

$T_4 = T_3 \cup \{\langle[1],[2]\rangle\}$

P_4 = {merge(X,Y,Z):-null(Y),X=Z.
merge(X,Y,Z):- null(X), Y=Z.
merge(X,Y,Z):-head(X,X1),tail(X,X2),
merge(X2,Y,W),cons(X1,W,Z).}

P_4 |- merge(X,Y,Z') P |- merge(X,Y,Z) y Z≠ Z'⇒ ⟨X,Y⟩= ⟨[2],[1]⟩
(Z=[1,2], Z'=[2,1])

P:
(1) merge(X,Y,Z):-null(X),Y=Z.
(2) merge(X,Y,Z):-null(Y),X=Z.
(3) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(X,X2),X1=<Y1,
merge(X2,Y,W),cons(X1,W,Z).
(4) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(Y,Y2),X1>Y1,
merge(X,Y2,W),cons(Y1,W,Z).

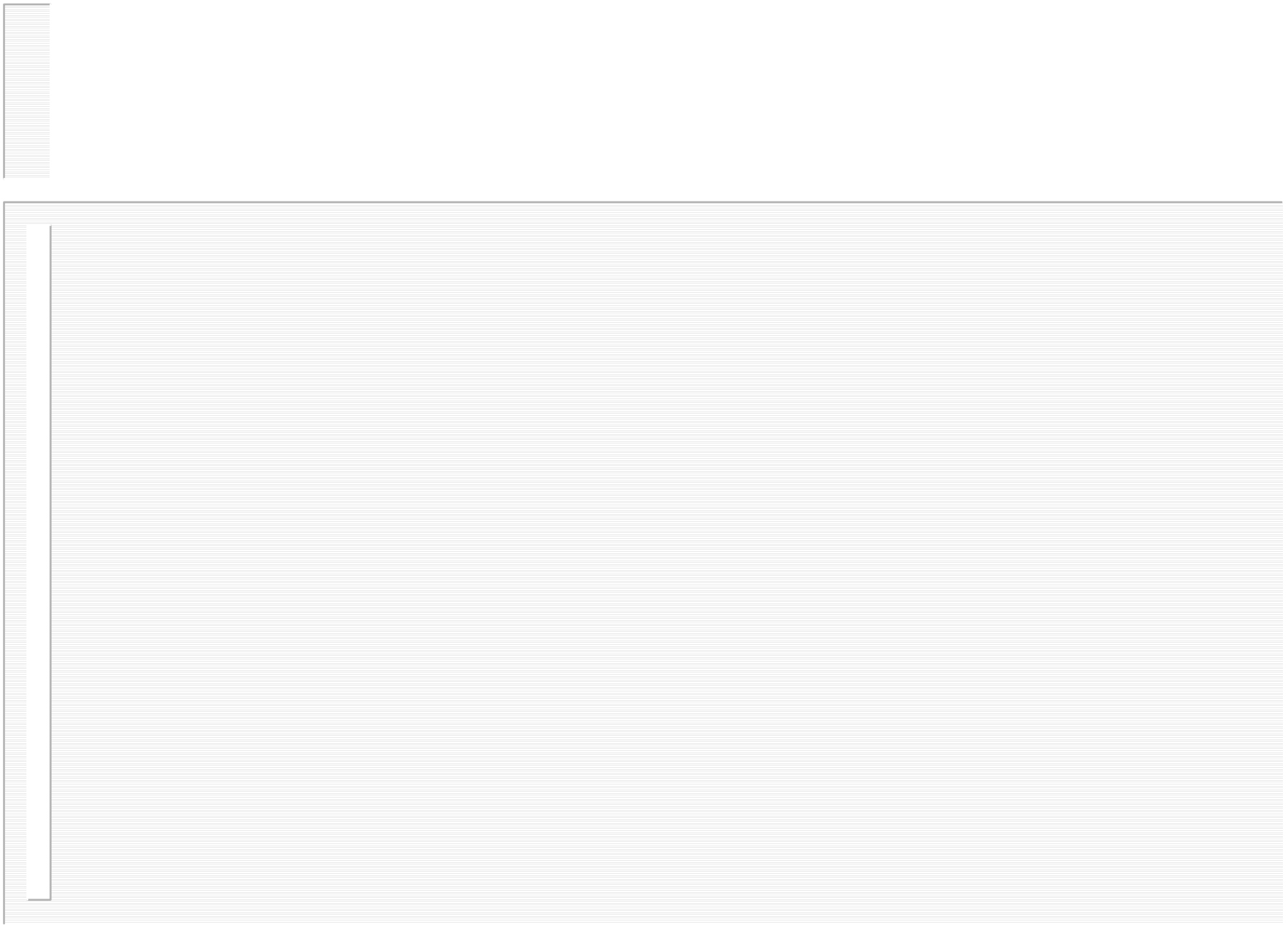
Prueba de programas: un ejemplo (*merge*).

$T_5 = T_4 \cup \{\langle[2],[1]\rangle\}$

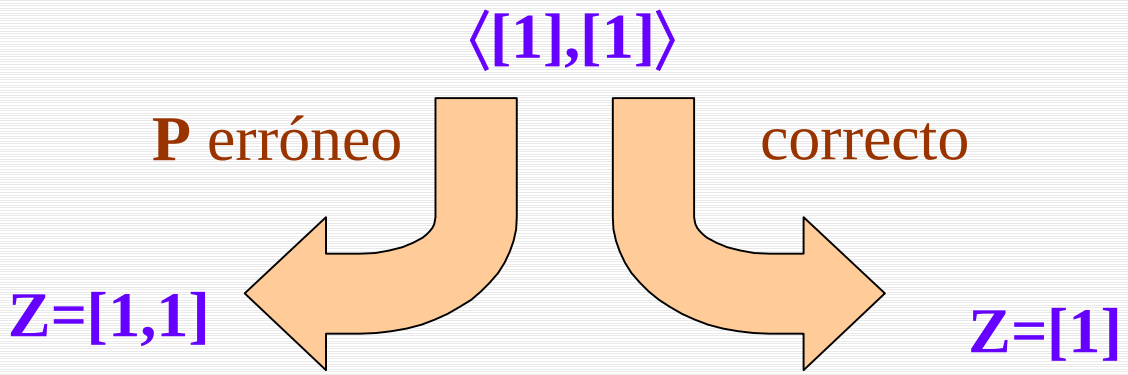
P_5 = {merge(X,Y,Z):-null(Y),X=Z.
merge(X,Y,Z):-null(X),Y=Z.
merge(X,Y,Z):-head(X,X1), head(Y,Y1),
tail(X,X2), X1 < Y1,
merge(X2,Y,W),cons(X1,W,Z).
merge(X,Y,Z):-head(X,X1), head(Y,Y1),
tail(Y,Y2), X1 > Y1,
merge(X,Y2,W),cons(Y1,W,Z).}

P_5 |- merge(X,Y,Z') P |- merge(X,Y,Z) y Z≠ Z'⇒ ⟨X,Y⟩= ⟨[1],[1]⟩
(Z=[1,1], Z' no existe)

P: (1) merge(X,Y,Z):-null(X),Y=Z.
(2) merge(X,Y,Z):-null(Y),X=Z.
(3) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(X,X2),X1=<Y1,
merge(X2,Y,W),cons(X1,W,Z).
(4) merge(X,Y,Z):-head(X,X1),head(Y,Y1),tail(Y,Y2),X1>Y1,
merge(X,Y2,W),cons(Y1,W,Z).



Prueba de programas: un ejemplo (*merge*).



Reificación de datos

- El proceso de representar un tipo abstracto con un tipo concreto.

ret: ConType ⟶ AbsType

Reificación de datos (2)

Ejemplo: Conjuntos (a,b,c) Tipo abstracto
Listas (a,b,c) Tipo concreto

ret([ ]) = { }
ret([a]) = {a}
ret([a,b]) = {a,b}
ret([b,a]) = {a,b}
ret([c,a,b]) = {a,b,c}
. . .

Reificación de datos (3)

- Funciones abstractas

  $AbsFun: AbsType * AbsType \longrightarrow AbsType$

- Funciones concretas

  $ConFun: ConType * ConType \longrightarrow ConType$

- Relación

  $ret(ConFun(x,y)) = AbsFun(ret(x), ret(y))$

- Ejemplo:

  $ret(append(S1,S2)) = union(ret(S1), ret(S2))$

Reificación de datos (4)

- Reificación de datos en ILP:

  Dados un tipo abstracto AbsType, una función abstracta *AbsFun*, y algún tipo de datos concreto candidato *ConType* con funciones concretas *ConFun*a a ser diseñadas, tales que *ret*(*ConFun*(x,y))= *AbsFun*(ret(x),*ret*(y)).

- Ejemplo: diseño de la función *union(S1,S2)*

Reificación de datos (5)

- EJEMPLOS

ret([ ],{ })
ret([a],{a})
ret([b],{b})
ret([c],{c})
ret([a,b],{a,b})
ret([b,a],{a,b})
. . .

union({ },{ },{ })
union({a},{ },{a})
union({b},{ },{b})
union({c},{ },{c})
union({ },{a},{a})
union({a},{b},{a,b})
. . .

append(List1,List2,List3)

```
append(L1,L2,L3):- ret(L1,S1),
                   ret(L2,S2),
                   ret(L3,S3),
                   union(S1,S2,S3).
```

Reificación de datos (6)

- Debemos eliminar algunos tripletes (L1, L2, L3) que cumplen la definición anterior

  append([a],[b],[a,b])

  append([a],[b],[b,a])

- append debe satisfacer:

  $\forall$L1,L2: ret(L1,S1) $\wedge$ ret(L2,S2) $\wedge$ union(S1,S2,S3) $\Rightarrow$

  $\exists$L3: ret(L3,S3) $\wedge$ append(L1,L2,L3)

- Única representación:

```
equiv(C1,C2):- ret(C1,A),
               ret(C2,A).
```

Reificación de datos (7)

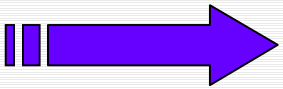
- ILP: **B∧ H |- E**
- Extender el marco ILP para tratar con clases de equivalencia.
- Hipótesis inicial:

append(L1,L2,L):- equiv(L,L3),
append(L1,L2,L3).

- Alternativa:

append(L1,L2,ListOfL3)

append([a],[b],[a,b])
append([a],[b],[b,a])



append0([a],[b],[[a,b],[b,a]])

append0(L1,L2, ListOfL3):-
member(L3, ListOfL3),
append(L1,L2,L3).

BK

Reificación de datos (8)

- BK:

```
equiv([a,b],[b,a])
equiv([a,b,c],[a,c,b])
equiv([a,b,c],[b,a,c])
. . .
member(X,[X|L]).
member(X,[Y|L]):- member(X,L).
```

Reificación de datos (9)

- Ejemplos de append se proporcionan como una tabla exhaustiva de átomos básicos de la forma

  example(append(L1,L2,L3),TruthValue)

  para todas las listas que satisfacen

  example(append(L1,L2,L3),true):-
  ret(L1,S1),
  ret(L2,S2),
  ret(L3,S3),
  union(S1,S2,S3).

  los átomos que no satisfacen la regla anterior

  example(append(L1,L2,L3),false)

Reificación de datos (10)

- Programa inducido por un sistema ILP (Markus):

```
append([X|Y],Z,[X|U]):- not member(X,Z),
                         append(Y,Z,U).
append(X,[Y|Z],U):- member(Y,X),
                    append(Z,X,U).
append([X|Y],Z,U):- member(X,Z),
                    append(Y,Z,U).
append([ ],X,X).
```

redundante

Invariantes de bucles

- La demostración formal de un programa procedural requiere encontrar precondiciones de las sentencias

  Fuertes $\Rightarrow$ verdad de las postcondiciones

  Débiles $\Rightarrow$ demostración en cada paso de la ejecución
- Invariantes de bucles: difíciles de encontrar

Usar las trazas como ejemplos positivos para aprender el invariante

* Ejemplos negativos: asunción de mundo cerrado

* Background knowledge

Invariantes de bucles (2)

- Ejemplo: división entera.

```
begin
        input(A,B);
        Q := 0;
        R := A;
        while R>B do
                begin
                        {invariante p(A,B,Q,R)}
                        Q := Q+1;
                        R := R-B
                end
        output(Q,R)
        {div(A,B,Q,R)}
end
```

$div(A,B,Q,R) \Leftrightarrow A=B*Q+R \wedge 0 \leq R < B$

Invariantes de bucles (3)

- Ejemplos positivos: dividir 9 por 2, dividir 8 por 3

A B Q R

p(9,2,0,9)
p(9,2,1,7)
p(9,2,2,5)
p(9,2,3,3)
p(9,2,4,1)
p(8,3,0,8)
p(8,3,1,5)
p(8,3,2,2)

ejemplos negativos

not p(9,2,0,0)
not p(9,2,1,9)
not p(9,2,2,2)
not p(9,2,3,4)
not p(9,2,4,7)

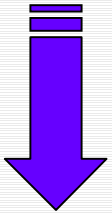
Invariantes de bucles (4)

- El sistema FOIL con los predicados mult/3 y add/3 en el BK induce el siguiente programa para p:

```
p(A,B,C,D):-
        mult(B,C,E),
        add(E,D,A).
```

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:

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

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

- Ejemplos:
  - $r(X,Y) \leftarrow primitive(X),\ solve(X,Y)$

    $r(X,Y) \leftarrow nonprimitive(X),\ decompose(X,H,X_1,X_2),$
    $r(X_1,Y_1),\ r(X_2,Y_2),\ compose(H,Y_1,Y_2,Y)$

    } divide y vencerás

  - $reverse(L,R) \leftarrow L=[\ ],\ R=[\ ]$

    $reverse(L,R) \leftarrow L=[HL|TL],\ reverse(TL,TR),\ compose(HL,TR,R)$

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) ←

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

close-cd(S,C,V), *%V=C ∪T ∪CD, T-plantilla de S-, CD define c y 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)*

aceptable(P_q) → *% determinar si la invención de predicados es necesaria*

P_r =V∪P_p ∪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(…)←

    r(…) ← r(…)
  - 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

Síntesis usando esquemas: CRUSTACEAN (2)

- los ejemplos positivos (Pi) se resuelven con instancias de B(Bi) y con R (haciendo anotaciones)

  Pi=last(a,[c,a]), Bi=last(a,[a]), R=last(A,[B,C|T]) ← last(A,[C|T])

  lo que sirve para encontrar B y R

- Cómputo de B: obtener el lgg de las Bi candidatas; si muy general probar alternativas

  last(a,[c,a]); last(b,[e,d,b]) → lgg{last(a,c),last(b,d)} → last(A,B) ← incorrecto

  lgg{last(a,[c]),last(b,[d])} → last(A,[A]) ← correcto

Síntesis usando esquemas: CRUSTACEAN (3)

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

  last(a,[c,a]) last(b,[e,d,b]) last(b,[d,b])

  lgg= last(A,[B,C|T])

  El cuerpo se construye usando las anotaciones

  last(A,[C|T])

  last(A,[A]) ←

  last(A,[B,C|T]) ← 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: $r(\ldots)\leftarrow$; $r(\ldots) \leftarrow r(\ldots), q(\ldots)$ } no hay c, d, p
  - close-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.

    length([a,b],s^2(0)) y length([a,b,c,d],s^4(0)) $\Rightarrow$ length([H|T],s(N)) $\leftarrow$ length(T,N)
    - cláusula base: lgg de los hechos sin resolver

    length([ ], 0) $\Rightarrow$ length([ ],0) $\leftarrow$
    - parámetros de q: todas las variables de la cláusula recursiva

Síntesis usando esquemas: CILP (2)

- **aceptable**: invención de predicados si el programa inducido por cada par de ejemplos positivos cubre algún ejemplo negativo
- **recurse**: sbis(E_q,■, ■,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
  - maxdepth(append(X,Y,Z))= length(X) + 1
  - basecase(append(X,Y,Z))= *if* X=[ ] *then* true *else* false
- Lenguaje de hipótesis: programas definidos ij-determinados, con 2 cláusulas lineales y cerradamente recursivas

cota superior

condición pertenencia

r(…) ← c(…)

r(…) ← d(…), r(…)

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

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

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

append(V,Y,U)

e= append([1,2],[3],[1,2,3]) ⇒ no caso base ⇒ R no se generaliza ya que cubre e

i1= append ([2],[3],[2,3]) ⇒ no caso base ⇒ lgg(R,i1)=R

. . .

in= append ([ ],[ ],[ ]) ⇒ caso base ⇒ lgg(B,in)= append(A,B,C) ← A=[ ], C=B

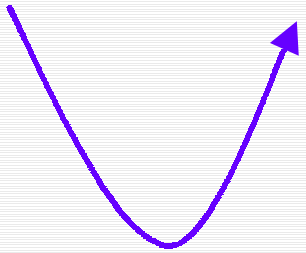
Síntesis usando esquemas: FORCE2 (4)

- 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 |= E, nº finito de refutaciones que no repiten secuencia de cláusulas
  - Primero: cláusulas usadas en refutaciones de ejemplos positivos y negativos ⇒ desplegar un átomo
  - Segundo: cláusulas usadas en refutaciones de ejemplos negativos y no usadas en los positivos ⇒ eliminar cláusula

Síntesis sin esquemas: SPECTRE II (2)

Evidencia positiva: odd(s(0)), odd(s^3(0)), odd(s^5(0))
Evidencia negativa: odd(0), odd(s^2(0)), odd(s^4(0))
Teoría inicial: odd(0) ← (c1)
odd(s(X)) ← odd(X) (c2)