Inteligencia artificial, lógicamente

José A. Alonso Jiménez
http://www.cs.us.es/∼jalonso

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Definición de problemas de estados

- **Elementos que describen un problema:**
  - **Estado inicial.**
  - **Operadores.**
  - **Estados finales.**
- **Suposiciones subyacentes:**
  - **Agente único.**
  - **Conocimiento completo.**

Ejemplo de PES: problema de las jarras

- **Enunciado:**
  - **Se tienen dos jarras, una de 4 litros de capacidad y otra de 3.**
  - **Ninguna de ellas tiene marcas de medición.**
  - **Se tiene una bomba que permite llenar las jarras de agua.**
  - **Averiguar cómo se puede lograr tener exactamente 2 litros de agua en la jarra de 4 litros de capacidad.**
- **Representación de estados: (x y) con x en {0,1,2,3,4} e y en {0,1,2,3}.**
- **Número de estados: 20.**

Planteamiento del problema de las jarras

- **Estado inicial: (0 0).**
- **Estados finales: (2 y).**
- **Operadores:**
  - **Llenar la jarra de 4 litros con la bomba.**
  - **Llenar la jarra de 3 litros con la bomba.**
  - **Vaciar la jarra de 4 litros en el suelo.**
  - **Vaciar la jarra de 3 litros en el suelo.**
  - **Llenar la jarra de 4 litros con la jarra de 3 litros.**
  - **Llenar la jarra de 3 litros con la jarra de 4 litros.**
  - **Vaciar la jarra de 3 litros en la jarra de 4 litros.**
  - **Vaciar la jarra de 4 litros en la jarra de 3 litros.**

Implementación del problema de las jarras

- **Representación de estados**

```
(defun crea-estado (x y)
  (list x y))

(defun contenido-jarra-4 (estado)
  (first estado))

(defun contenido-jarra-3 (estado)
  (second estado))
```

- **Estado inicial**

```
(defparameter *estado-inicial*
  (crea-estado 0 0))
```

- **Estados finales**

```
(defun es-estado-final (estado)
  (= 2 (contenido-jarra-4 estado)))
```

Implementación del problema de las jarras

- **Operadores**

```
(defparameter *operadores*
  '(llenar-jarra-4
    llenar-jarra-3
    vaciar-jarra-4
    vaciar-jarra-3
    llenar-jarra-4-con-jarra-3
    llenar-jarra-3-con-jarra-4
    vaciar-jarra-3-en-jarra-4
    vaciar-jarra-4-en-jarra-3))

(defun llenar-jarra-4 (estado)
  (when (< (contenido-jarra-4 estado) 4)
        (crea-estado 4
                     (contenido-jarra-3 estado))))
```

Implementación del problema de las jarras

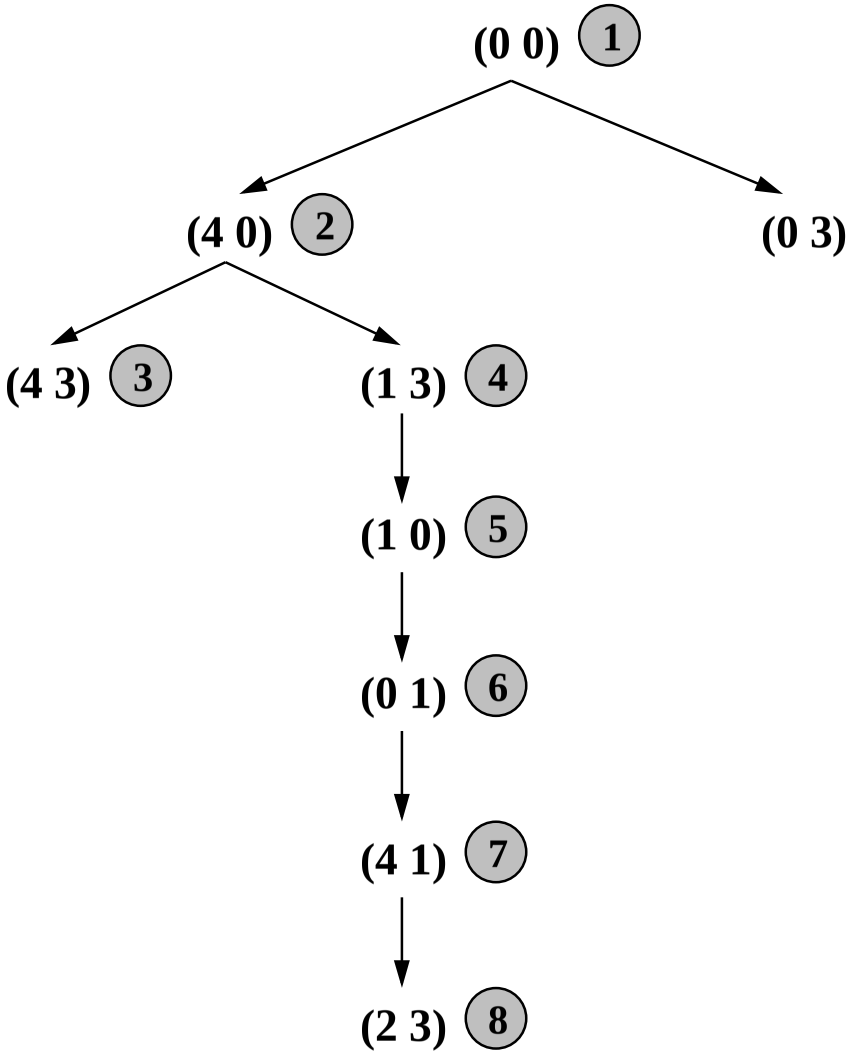
```
(defun vaciar-jarra-4 (estado)
  (when (> (contenido-jarra-4 estado) 0)
        (crea-estado 0
                     (contenido-jarra-3 estado))))

(defun llenar-jarra-4-con-jarra-3 (estado)
  (let ((x (contenido-jarra-3 estado))
        (y (contenido-jarra-4 estado)))
    (when (and (> x 0)
               (< y 4)
               (> (+ y x) 4))
          (crea-estado 4 (- x (- 4 y))))))

(defun vaciar-jarra-3-en-jarra-4 (estado)
  (let ((x (contenido-jarra-3 estado))
        (y (contenido-jarra-4 estado)))
    (when (and (> x 0)
               (<= (+ y x) 4))
          (crea-estado (+ x y) 0))))
```

Búsqueda de solución

- **Grafo de búsqueda en profundidad:**



Búsqueda de solución

- **Tabla de búsqueda en profundidad:**

Nodo	Actual	Sucesores	Abiertos
			((0 0))
1	(0 0)	((4 0) (0 3))	((4 0) (0 3))
2	(4 0)	((4 3) (1 3))	((4 3) (1 3) (0 3))
3	(4 3)	()	((1 3) (0 3))
4	(1 3)	((1 0))	((1 0) (0 3))
5	(1 0)	((0 1))	((0 1) (0 3))
6	(0 1)	((4 1))	((4 1) (0 3))
7	(4 1)	((2 3))	((2 3) (0 3))
8	(2 3)		

- **Estados de la solución:**

```
((2 3) (4 1) (0 1) (1 0) (1 3) (4 0) (0 0))
```

Procedimiento de búsqueda en profundidad

1. Crear las siguientes variables locales
   1.1. ABIERTOS (para almacenar los nodos generados aún no analizados) con valor la lista formada por el nodo inicial (es decir, el nodo cuyo estado es el estado inicial y cuyo camino es la lista vacía);
   1.2. CERRADOS (para almacenar los nodos analizados) con valor la lista vacía;
   1.3. ACTUAL (para almacenar el nodo actual) con valor la lista vacía.
   1.4. NUEVOS-SUCESORES (para almacenar la lista de los sucesores del nodo actual) con valor la lista vacía.

Procedimiento de búsqueda en profundidad

```
2. Mientras que ABIERTOS no esté vacía,
   2.1 Hacer ACTUAL el primer nodo de ABIERTOS
   2.2 Hacer ABIERTOS el resto de ABIERTOS
   2.3 Poner el nodo ACTUAL en CERRADOS.
   2.4 Si el nodo ACTUAL es un final,
       2.4.1 devolver el nodo ACTUAL y terminar.
       2.4.2 en caso contrario, hacer
             2.4.2.1 NUEVOS-SUCESORES la lista de sucesores del nodo
                     ACTUAL que no están en ABIERTOS ni en CERRADOS y
             2.4.2.2 ABIERTOS la lista obtenida añadiendo los
                     NUEVOS-SUCESORES al principio de ABIERTOS.
3. Si ABIERTOS está vacía, devolver NIL.
```

Implementación de la búsqueda en profundidad

```
(defun busqueda-en-profundidad ()
  (let ((abiertos (list (crea-nodo :estado *estado-inicial*       ;1.1
                                   :camino nil)))
        (cerrados nil)                                            ;1.2
        (actual nil)                                              ;1.3
        (nuevos-sucesores nil))                                   ;1.4
    (loop until (null abiertos) do                                ;2
          (setf actual (first abiertos))                          ;2.1
          (setf abiertos (rest abiertos))                         ;2.2
          (setf cerrados (cons actual cerrados))                  ;2.3
          (cond ((es-estado-final (estado actual))                ;2.4
                 (return actual))                                 ;2.4.1
                (t (setf nuevos-sucesores                         ;2.4.2.1
                         (nuevos-sucesores actual abiertos cerrados))
                   (setf abiertos                                 ;2.4.2.2
                         (append nuevos-sucesores abiertos)))))))
```

Soluciones de los problemas en profundidad

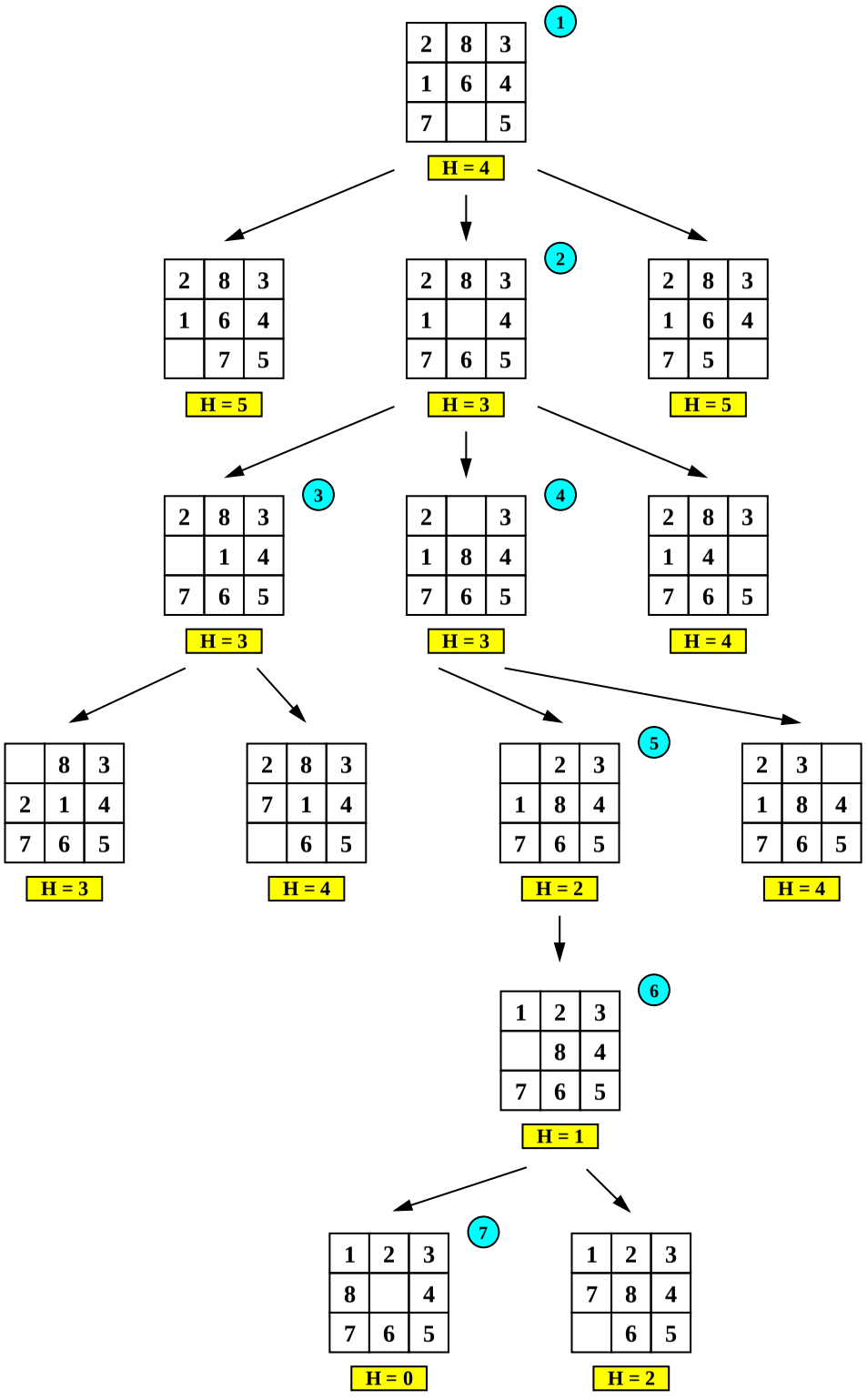
- **Problema de las jarras:**

```
> clisp
Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
...
Copyright (c) Bruno Haible, Sam Steingold 1999-2002
> (load "p-jarras-1.lsp")
T
> (load "b-profundidad.lsp")
T
> (busqueda-en-profundidad)
#S(NODO :ESTADO (2 3)
        :CAMINO (LLENAR-JARRA-3-CON-JARRA-4
                 LLENAR-JARRA-4
                 VACIAR-JARRA-4-EN-JARRA-3
                 VACIAR-JARRA-3
                 LLENAR-JARRA-3-CON-JARRA-4
                 LLENAR-JARRA-4))
```

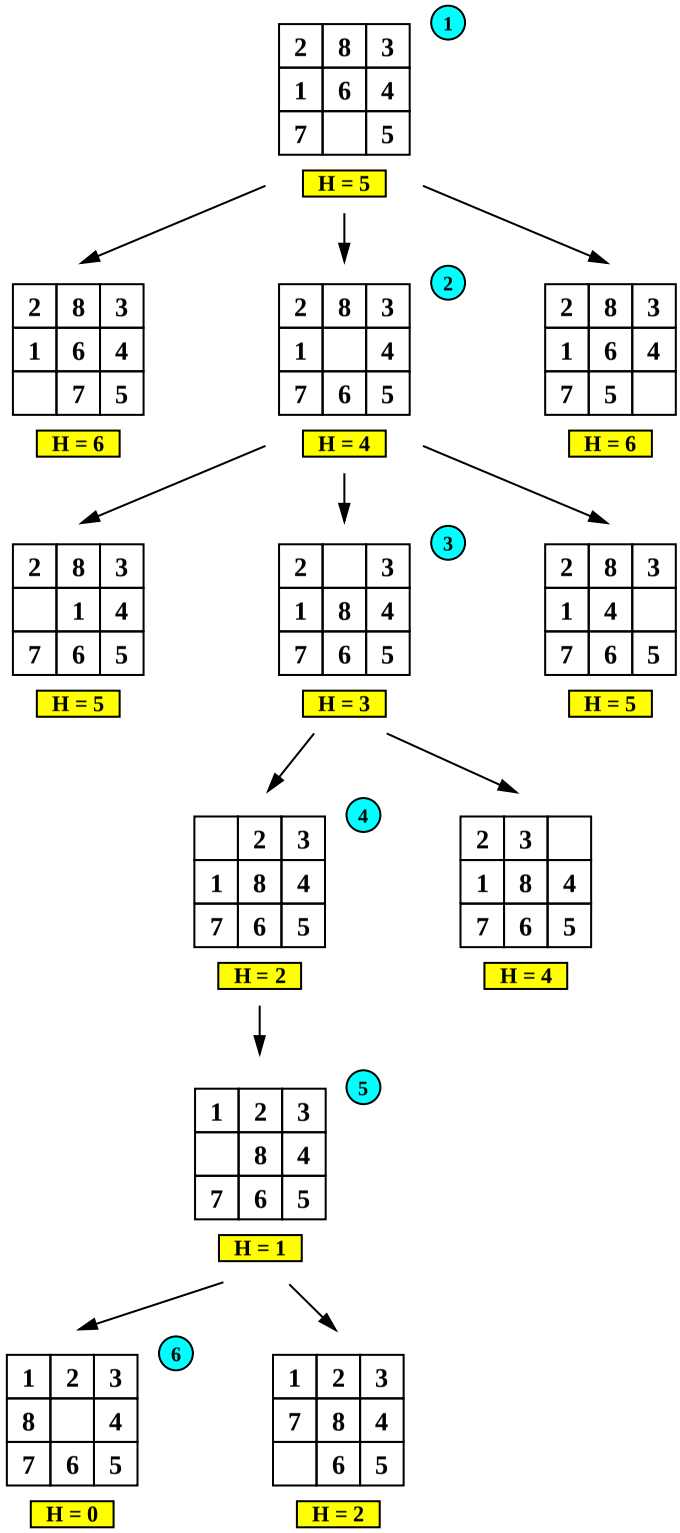
Soluciones de los problemas en profundidad

```
> (trace es-estado-final)
(ES-ESTADO-FINAL)
> (busqueda-en-profundidad)
1. Trace: (ES-ESTADO-FINAL '(0 0))
1. Trace: (ES-ESTADO-FINAL '(4 0))
1. Trace: (ES-ESTADO-FINAL '(4 3))
1. Trace: (ES-ESTADO-FINAL '(1 3))
1. Trace: (ES-ESTADO-FINAL '(1 0))
1. Trace: (ES-ESTADO-FINAL '(0 1))
1. Trace: (ES-ESTADO-FINAL '(4 1))
1. Trace: (ES-ESTADO-FINAL '(2 3))
#S(NODO :ESTADO (2 3)
        :CAMINO (LLENAR-JARRA-3-CON-JARRA-4 LLENAR-JARRA-4
                 VACIAR-JARRA-4-EN-JARRA-3  VACIAR-JARRA-3
                 LLENAR-JARRA-3-CON-JARRA-4 LLENAR-JARRA-4))
```

8–puzzle por primero el mejor: 1ª heurística



8–puzzle por primero-el-mejor: 2ª heurística



Problemas de espacio de estados y lógica

- **Relación histórica:**
  - **1957: Newell, Shaw y Simon: "General Problem Solver".**
  - **1956: Newell y Simon: lógico teórico.**
- **Relación conceptual:**
  - **Estado inicial: teorema a demostrar.**
  - **Operadores: reglas de inferencia.**
  - **Estados finales: axiomas.**
- **Relación instrumental:**
  - **Lisp y lambda cálculo.**

Resolución SLD: Problema

- **Base de conocimiento de animales:**
  - **Regla 1: Si un animal es ungulado y tiene rayas negras, entonces es una cebra.**
  - **Regla 2: Si un animal rumia y es mamífero, entonces es ungulado.**
  - **Regla 3: Si un animal es mamífero y tiene pezuñas, entonces es ungulado.**
  - **Hecho 1: El animal tiene es mamífero.**
  - **Hecho 2: El animal tiene pezuñas.**
  - **Hecho 3: El animal tiene rayas negras.**
- **Objetivo:**
  - **Demostrar a partir de la base de conocimientos que el animal es una cebra.**

Resolución SLD: Representación

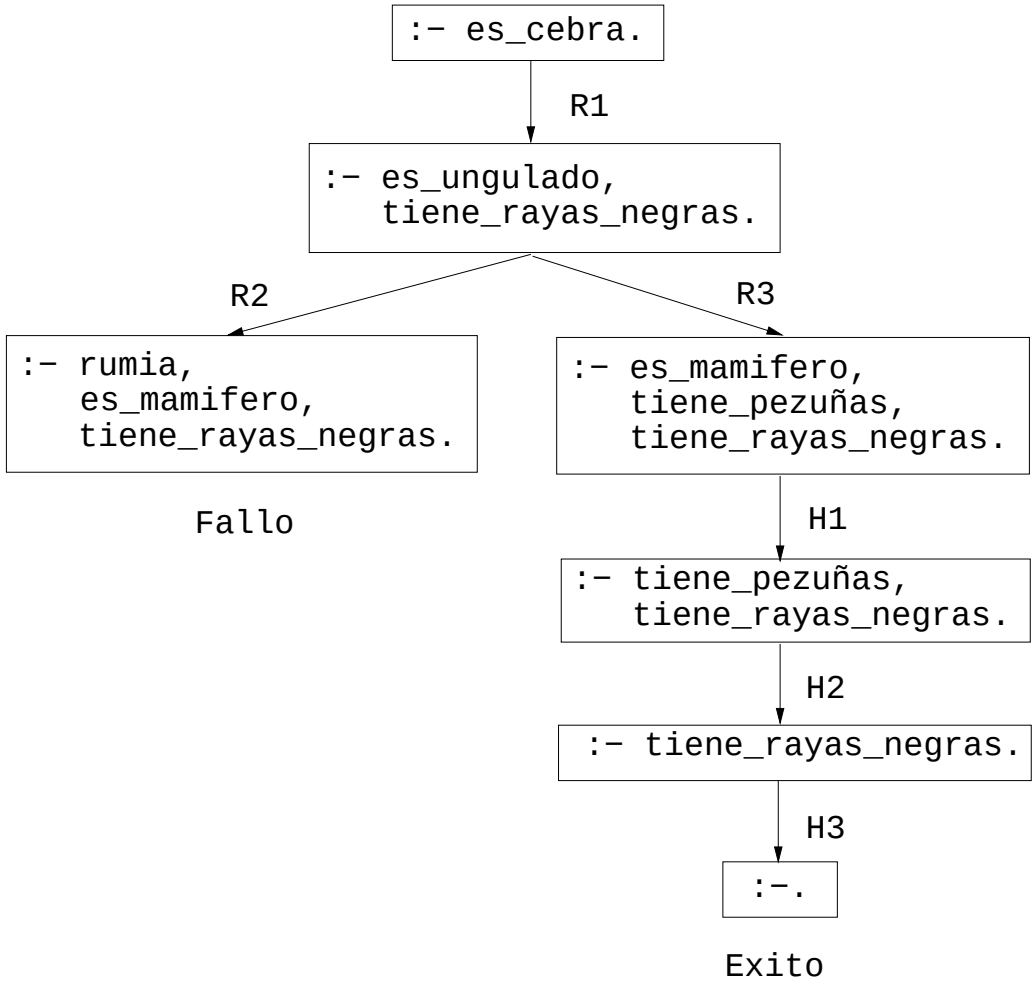
- **Representación lógica de la base de conocimiento:**

```
es_cebra    :- es_ungulado, tiene_rayas_negras.  % Regla 1
es_ungulado :- rumia, es_mamífero.               % Regla 2
es_ungulado :- es_mamífero, tiene_pezuñas.       % Regla 3
es_mamífero.                                     % Hecho 1
tiene_pezuñas.                                   % Hecho 2
tiene_rayas_negras.                              % Hecho 3
```

- **Sesión:**

```
> pl
Welcome to SWI-Prolog (Version 5.0.3)
Copyright (c) 1990-2002 University of Amsterdam.
?- [animales].
Yes
?- es_cebra.
Yes
```

Resolución SLD: Arbol de resolución



Resolución con unificación

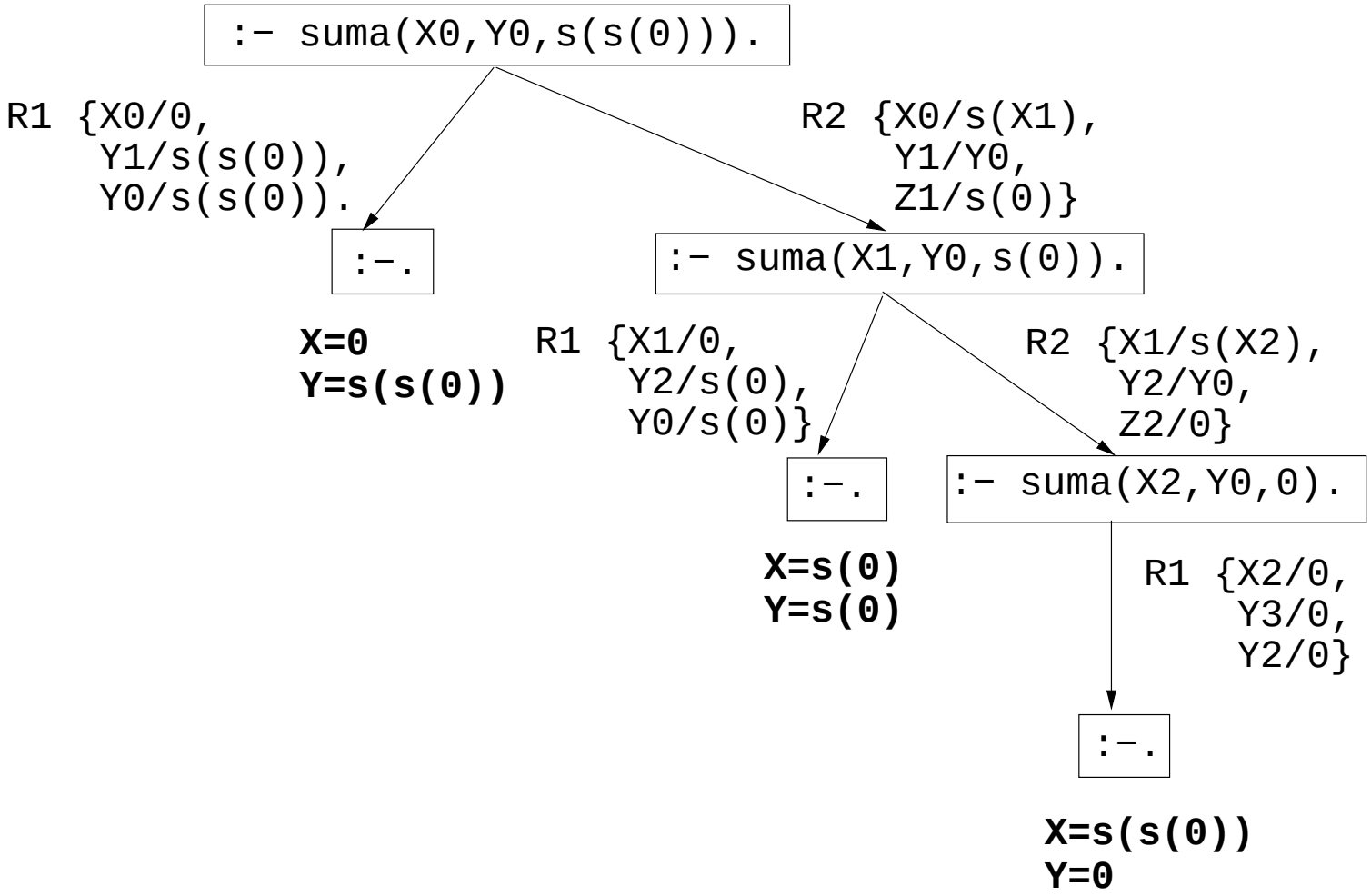
- **Programa lógico** suma

```
suma(0,X,X).                          % R1
suma(s(X),Y,s(Z)) :- suma(X,Y,Z).     % R2
```

- **Sesión**

```
?- suma(s(0),s(s(0)),X).
X = s(s(s(0)))
Yes
?- suma(X,Y,s(s(0))).
X = 0
Y = s(s(0)) ;
X = s(0)
Y = s(0) ;
X = s(s(0))
Y = 0 ;
No
```

Cálculo de respuestas



Base de conocimiento CLIPS

- BC `animales.clp`

```
(deffacts hechos-iniciales
  (tiene-pelos)
  (tiene-pezugnas)
  (tiene-rayas-negras))

(defrule mamifero-1
  (tiene-pelos)
  =>
  (assert (es-mamifero)))

(defrule mamifero-2
  (da-leche)
  =>
  (assert (es-mamifero)))

(defrule ungulado-1
  (es-mamifero)
  (tiene-pezugnas)
  =>
  (assert (es-ungulado)))

(defrule ungulado-2
  (es-mamifero)
  (rumia)
  =>
  (assert (es-ungulado)))
```

Base de conocimiento CLIPS

```
(defrule jirafa
  (es-ungulado)
  (tiene-cuello-largo)
  =>
  (assert (es-jirafa)))

(defrule cebra
  (es-ungulado)
  (tiene-rayas-negras)
  =>
  (assert (es-cebra)))
```

Base de conocimiento CLIPS

- **Sesión**

```
CLIPS> (load "animales.clp")
$******
TRUE
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (watch activations)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (tiene-pelos)
==> Activation 0      mamifero-1: f-1
==> f-2     (tiene-pezugnas)
==> f-3     (tiene-rayas-negras)
CLIPS> (run)
FIRE    1 mamifero-1: f-1
==> f-4     (es-mamifero)
==> Activation 0      ungulado-1: f-4,f-2
FIRE    2 ungulado-1: f-4,f-2
==> f-5     (es-ungulado)
==> Activation 0      cebra: f-5,f-3
FIRE    3 cebra: f-5,f-3
==> f-6     (es-cebra)
```

Base de conocimiento CLIPS

- **Tabla de seguimiento:**

Hechos	E	Agenda	D
`f0 (initial-fact)`	0		
`f1 (tiene-pelos)`	0	`mamifero-1: f1`	1
`f2 (tiene-pezugnas)`	0		
`f3 (tiene-rayas-negras)`	0		
`f4 (es-mamifero)`	1	`ungulado-1: f4,f2`	2
`f5 (es-ungulado)`	2	`cebra: f5,f3`	3
`f6 (es-cebra)`			

Cuadrados mágicos en CLIPS

- **Problema de cuadrados mágicos**
  - **Enunciado**

```
ABC          {A,B,C,D,E,F,G,H,I} = {1,2,3,4,5,6,7,8,9}
DEF          A+B+C = D+E+F = G+H+I = A+D+G = B+E+F
GHI                = C+F+I = A+E+I = C+E+G
```

  - **Sesión**

```
CLIPS> (run)
Solucion 1:
   492
   357
   816

....
```

  - **Programa** `cuadrado-magico.clp`**:**

```
(deffacts datos
  (numero 1) (numero 2) (numero 3) (numero 4)
  (numero 5) (numero 6) (numero 7) (numero 8)
  (numero 9) (solucion 0))

(deffunction suma-15 (?x ?y ?z)
  (= (+ ?x ?y ?z) 15))
```

Cuadrados mágicos en CLIPS

```
(defrule busca-cuadrado
  (numero ?e)
  (numero ?a&~?e)
  (numero ?i&~?e&~?a&:(suma-15 ?a ?e ?i))
  (numero ?b&~?e&~?a&~?i)
  (numero ?c&~?e&~?a&~?i&~?b&:(suma-15 ?a ?b ?c))
  (numero ?f&~?e&~?a&~?i&~?b&~?c&:(suma-15 ?c ?f ?i))
  (numero ?d&~?e&~?a&~?i&~?b&~?c&~?f
          &:(suma-15 ?d ?e ?f))
  (numero ?g&~?e&~?a&~?i&~?b&~?c&~?f&~?d
          &:(suma-15 ?a ?d ?g)&:(suma-15 ?c ?e ?g))
  (numero ?h&~?e&~?a&~?i&~?b&~?c&~?f&~?d&~?g
          &:(suma-15 ?b ?e ?h)&:(suma-15 ?g ?h ?i))
  =>
  (assert (escribe-solucion ?a ?b ?c ?d ?e
                            ?f ?g ?h ?i)))

(defrule escribe-solucion
  ?f <- (escribe-solucion ?a ?b ?c
                          ?d ?e ?f
                          ?g ?h ?i)
  ?solucion <- (solucion ?n)
  =>
  (retract ?f ?solucion)
  (assert (solucion (+ ?n 1)))
  (printout t "Solucion " (+ ?n 1) ":" crlf)
  (printout t "   " ?a ?b ?c crlf)
  (printout t "   " ?d ?e ?f crlf)
  (printout t "   " ?g ?h ?i crlf)
  (printout t crlf))
```

Razonamiento con OTTER

- Base de conocimiento
  - Base de reglas:
    * R1: Si el animal tiene pelos es mamífero.
    * R2: Si el animal da leche es mamífero.
    * R3: Si el animal es un mamífero y tiene pezuñas es ungulado.
    * R4: Si el animal es un mamífero y rumia es ungulado.
    * R5: Si el animal es un ungulado y tiene cuello largo es una jirafa.
    * R6: Si el animal es un ungulado y tiene rayas negras es una cebra.

  - Base de hechos:
    * H1: El animal tiene pelos.
    * H2: El animal tiene pezuñas.
    * H3: El animal tiene rayas negras.

  - Consecuencia
    * El animal es una cebra.

Razonamiento con OTTER

- **Solución con OTTER**
  - **Representación en OTTER** (`animales.in`)

```
formula_list(sos).
tiene_pelos | da_leche -> es_mamifero.
es_mamifero & (tiene_pezuñas | rumia) -> es_ungulado.
es_ungulado & tiene_cuello_largo -> es_jirafa.
es_ungulado & tiene_rayas_negras -> es_cebra.

tiene_pelos & tiene_pezuñas & tiene_rayas_negras.

-es_cebra.
end_of_list.

set(binary_res).
```

Razonamiento con OTTER

- **Solución con OTTER**

```
> otter <animales.in
-------> sos clausifies to:
list(sos).
1 [] -tiene_pelos | es_mamifero.
2 [] -da_leche | es_mamifero.
3 [] -es_mamifero | -tiene_pezuñas | es_ungulado.
4 [] -es_mamifero | -rumia | es_ungulado.
5 [] -es_ungulado | -tiene_cuello_largo | es_jirafa.
6 [] -es_ungulado | -tiene_rayas_negras | es_cebra.
7 [] tiene_pelos.
8 [] tiene_pezuñas.
9 [] tiene_rayas_negras.
10 [] -es_cebra.
end_of_list.
set(binary_res).
   dependent: set(factor).
   dependent: set(unit_deletion).

======= end of input processing =======
```

Razonamiento con OTTER

```
========== start of search ==========

given clause #1: (wt=1) 7 [] tiene_pelos.

given clause #2: (wt=1) 8 [] tiene_pezuñas.

given clause #3: (wt=1) 9 [] tiene_rayas_negras.

given clause #4: (wt=1) 10 [] -es_cebra.

given clause #5: (wt=2) 1 [] -tiene_pelos | es_mamifero.
** KEPT (pick-wt=1): 11 [binary,1.1,7.1] es_mamifero.
11 back subsumes 2.
11 back subsumes 1.

given clause #6: (wt=1) 11 [binary,1.1,7.1] es_mamifero.

given clause #7: (wt=3) 3 [] -es_mamifero
                             | -tiene_pezuñas
                             | es_ungulado.
** KEPT (pick-wt=1): 12 [binary,3.1,11.1,unit_del,8]
                       es_ungulado.
12 back subsumes 4.
12 back subsumes 3.

given clause #8: (wt=1) 12 [binary,3.1,11.1,unit_del,8]
                           es_ungulado.

given clause #9: (wt=3) 6 [] -es_ungulado
                             | -tiene_rayas_negras
                             | es_cebra.
** KEPT (pick-wt=0): 13 [binary,6.1,12.1,unit_del,9,10]
                       $F.
```

Razonamiento con OTTER

```
Length of proof is 2.  Level of proof is 2.

---------------- PROOF ----------------

1 [] -tiene_pelos | es_mamifero.
3 [] -es_mamifero | -tiene_pezuñas | es_ungulado.
6 [] -es_ungulado | -tiene_rayas_negras | es_cebra.
7 [] tiene_pelos.
8 [] tiene_pezuñas.
9 [] tiene_rayas_negras.
10 [] -es_cebra.
11 [binary,1.1,7.1] es_mamifero.
12 [binary,3.1,11.1,unit_del,8] es_ungulado.
13 [binary,6.1,12.1,unit_del,9,10] $F.

------------ end of proof -------------
```

Representación del conocimiento

- **Demostrar la validez del siguiente argumento:**
  Los caballos son más rápidos que los perros. Algunos galgos son más rápidos que los conejos. Lucero es un caballo y Orejón es un conejo. Por tanto, Lucero es más rápido que Orejón.
- **Nuevos problemas en la decisión de la validez de una argumentación:**
  - **Representación del conocimiento**
  - **Explicitación del conocimiento implícito**
- **Lenguaje del problema:**

Símbolos:	Significado:
Lucero	Lucero
Orejon	Orejón
CABALLO(x)	x es un caballo
CONEJO(x)	x es un conejo
GALGO(x)	x es un galgo
PERRO(x)	x es un perro
MAS_RAPIDO(x,y)	x es más rápido que y

Representación del conocimiento

- **Entrada** `ej-3a1.in`

```
formula_list(sos).
% Los caballos son más rápidos que los perros.
all x y (CABALLO(x) & PERRO(y) -> MAS_RAPIDO(x,y)).

% Algunos galgos son más rápidos que los conejos
exists x (GALGO(x) &
          (all y (CONEJO(y) -> MAS_RAPIDO(x,y)))).

% Lucero es un caballo
CABALLO(Lucero).

% Orejón es un conejo.
CONEJO(Orejon).

% Lucero no es más rápido que Orejón
-MAS_RAPIDO(Lucero,Orejon).
end_of_list.

set(binary_res).
```

- **Salida**

```
list(sos).
1 [] -CABALLO(x)| -PERRO(y)|MAS_RAPIDO(x,y).
2 [] GALGO($c1).
3 [] -CONEJO(y)|MAS_RAPIDO($c1,y).
4 [] CABALLO(Lucero).
5 [] CONEJO(Orejon).
6 [] -MAS_RAPIDO(Lucero,Orejon).
end_of_list.
```

Representación del conocimiento

```
given clause #1: (wt=2) 2 [] GALGO($c1).
given clause #2: (wt=2) 4 [] CABALLO(Lucero).
given clause #3: (wt=2) 5 [] CONEJO(Orejon).
given clause #4: (wt=3) 6 [] -MAS_RAPIDO(Lucero,Orejon).
given clause #5: (wt=5) 3 [] -CONEJO(y)|MAS_RAPIDO($c1,y).
** KEPT (pick-wt=3): 7 [binary,3.1,5.1] MAS_RAPIDO($c1,Orejon).

given clause #6: (wt=3) 7 [binary,3.1,5.1] MAS_RAPIDO($c1,Orejon).

given clause #7: (wt=7) 1 [] -CABALLO(x)| -PERRO(y)|MAS_RAPIDO(x,y).
** KEPT (pick-wt=5): 8 [binary,1.1,4.1] -PERRO(x)|MAS_RAPIDO(Lucero,x).
** KEPT (pick-wt=2): 9 [binary,1.3,6.1,unit_del,4] -PERRO(Orejon).

given clause #8: (wt=2) 9 [binary,1.3,6.1,unit_del,4] -PERRO(Orejon).

given clause #9: (wt=5) 8 [binary,1.1,4.1] -PERRO(x)|MAS_RAPIDO(Lucero,x).

Search stopped because sos empty.
```

Representación del conocimiento

- **Búsqueda de modelos con MACE**

```
mace -n2 -p -m1 <ej-3a1.in
```

- **Modelo encontrado**

```
======================== Model #1 at 0.03 seconds:

 Lucero: 1     Orejon: 0    $c1: 0

 CABALLO :     PERRO :      GALGO :      CONEJO :
       0 1           0 1          0 1          0 1
    -------       -------      -------      -------
       F T           F F          T F          T F

 MAS_RAPIDO :
     | 0 1
    --+----
    0 | T F
    1 | F F
end_of_model
```

- **Entrada** `ej-3a2.in`

```
include('ej-03a1.in').

formula_list(sos).
% Los galgos son perros
all x (GALGO(x) -> PERRO(x)).
end_of_list.

set(binary_res).
```

- **Salida**

```
Search stopped because sos empty.
```

Representación del conocimiento

- **Búsqueda de modelos con MACE**

```
mace -n2 -p -m1 <ej-3a2.in
```

- **Modelo encontrado**

```
Lucero: 1    Orejon: 1    $c1: 0

CABALLO :    PERRO :      GALGO :      CONEJO :
       0 1          0 1          0 1          0 1
   -------      -------      -------      -------
       F T          T F          T F          F T

MAS_RAPIDO :
     | 0 1
   --+----
   0 | F T
   1 | T F
```

- **Entrada** `ej-3a3.in`

```
include('ej-03a2.in').

formula_list(sos).
% Si x es más rápido que y e y es más rápido que z,
% entonces x es más rápido que z.
all x y z (MAS_RAPIDO(x,y) & MAS_RAPIDO(y,z)
           -> MAS_RAPIDO(x,z)).
end_of_list.

set(binary_res).
```

Representación del conocimiento

- **Prueba**

```
 1 [] -CABALLO(x)| -PERRO(y)|MAS_RAPIDO(x,y).
 2 [] GALGO($c1).
 3 [] -CONEJO(y)
      | MAS_RAPIDO($c1,y).
 4 [] CABALLO(Lucero).
 5 [] CONEJO(Orejon).
 6 [] -MAS_RAPIDO(Lucero,Orejon).
 7 [] -GALGO(x)|PERRO(x).
 8 [] -MAS_RAPIDO(x,y)
       | -MAS_RAPIDO(y,z)
       | MAS_RAPIDO(x,z).
 9 [binary,7.1,2.1] PERRO($c1).
10 [binary,3.1,5.1] MAS_RAPIDO($c1,Orejon).
11 [binary,1.1,4.1] -PERRO(x)
                     | MAS_RAPIDO(Lucero,x).
16 [binary,11.1,9.1] MAS_RAPIDO(Lucero,$c1).
19 [binary,8.1,16.1] -MAS_RAPIDO($c1,x)
                      | MAS_RAPIDO(Lucero,x).
36 [binary,19.1,10.1] MAS_RAPIDO(Lucero,Orejon).
37 [binary,36.1,6.1] $F.
```

- **Estadísticas**

```
clauses given                18
clauses generated            43
clauses kept                 28
clauses forward subsumed     12
clauses back subsumed         0
```

Demostración automática de teoremas

- **Problema elemental de grupos**
  - **Teorema:** ***Sea G un grupo y e su elemento neutro. Si, para todo x de G, $x^2 = e$, entonces G es conmutativo.***
  - **Formalización**
    * **Axiomas de grupo:**
      $(\forall x)[e.x = x]$
      $(\forall x)[x.e = x]$
      $(\forall x)[x.x^{-1} = e]$
      $(\forall x)[x^{-1}.x = e]$
      $(\forall x)(\forall y)(\forall z)[(x.y).z = x.(y.z)]$
    * **Hipótesis**
      $(\forall x)[x.x = e]$
    * **Conclusión**
      $(\forall x)(\forall y)[x.y = y.x]$

Demostración automática de teoremas

- **Entrada** `grupos-1a.in`

```
op(400, xfy, *).
op(300, yf, ^).

list(usable).
x = x.                              % Reflexividad
e * x = x.                          % Ax. 1
x * e = x.                          % Ax. 2
x^ * x = e.                         % Ax. 3
x * x^ = e.                         % Ax. 4
(x * y) * z = x * (y * z).          % Ax. 5
end_of_list.

list(sos).
x * x = e.
a * b != b * a.
end_of_list.

set(para_into).
set(para_from).
```

Demostración automática de teoremas

- **Uso de OTTER**

```
otter <grupos-1a.in
```

- **Prueba**

```
2 [] e*x=x.
3 [] x*e=x.
6 [] (x*y)*z=x*y*z.
7 [] x*x=e.
8 [] a*b!=b*a.
19 [para_from,7.1.2,3.1.1.2] x*y*y=x.
20 [para_from,7.1.2,2.1.1.1] (x*x)*y=y.
31 [para_into,19.1.1,6.1.2] (x*y)*y=x.
167 [para_into,20.1.1,6.1.1] x*x*y=y.
170 [para_from,20.1.1,6.1.1] x=y*y*x.
496 [para_into,167.1.1.2,31.1.1] (x*y)*x=y.
755 [para_into,496.1.1.1,170.1.2] x*y=y*x.
756 [binary,755.1,8.1] $F.
```

- **Estadísticas**

Analiz.	Gener.	Reten.	Sub. adel.	Sub. atrás	Seg.
72	5741	747	4994	45	0.26

Demostración automática de teoremas

- **Modo autónomo**
  - **Entrada** `ej-7d.in`

```
set(auto2).

op(400, xfy, *).
op(300, yf, ^).

list(usable).
e * x = x.                          % Ax. 1
x * e = x.                          % Ax. 2
x^ * x = e.                         % Ax. 3
x * x^ = e.                         % Ax. 4
(x * y) * z = x * (y * z).          % Ax. 5
x = x.                              % Ax. 6
x * x = e.
a * b != b * a.
end_of_list.
```

  - **Prueba**

```
1 [] a*b!=b*a.
2 [copy,1,flip.1] b*a!=a*b.
4,3 [] e*x=x.
6,5 [] x*e=x.
11 [] (x*y)*z=x*y*z.
14 [] x*x=e.
18 [para_into,11.1.1.1,14.1.1,demod,4,flip.1] x*x*y=y.
24 [para_into,11.1.1,14.1.1,flip.1] x*y*x*y=e.
34 [para_from,24.1.1,18.1.1.2,demod,6,flip.1] x*y*x=y.
38 [para_from,34.1.1,18.1.1.2] x*y=y*x.
39 [binary,38.1,2.1] $F.
```

  - **Estadísticas**

Analiz.	Gener.	Reten.	Sub. adel.	Sub. atrás	Seg.
12	90	20	87	8	0.18

Problema de las jarras

- **Enunciado:**
  - **Se tienen dos jarras, una de 4 litros de capacidad y otra de 3.**
  - **Ninguna de ellas tiene marcas de medición.**
  - **Se tiene una bomba que permite llenar las jarras de agua.**
  - **Averiguar cómo se puede lograr tener exactamente 2 litros de agua en la jarra de 4 litros de capacidad.**
- **Entrada** `jarras.in`

```
set(prolog_style_variables).
set(input_sequent).
set(output_sequent).
make_evaluable(_+_, $SUM(_,_)).
make_evaluable(_-_, $DIFF(_,_)).
make_evaluable(_<=_, $LE(_,_)).
make_evaluable(_>_, $GT(_,_)).
set(hyper_res).
```

Problema de las jarras

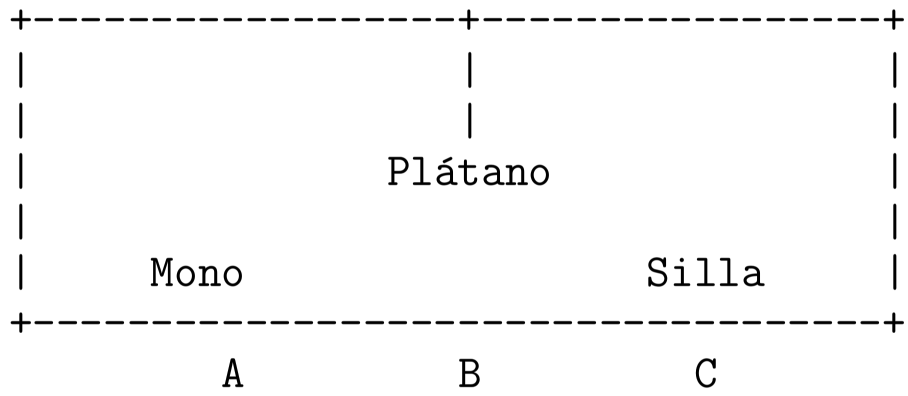
```
list(usable).
e(X,Y)             -> e(3,Y).
e(X,Y)             -> e(0,Y).
e(X,Y)             -> e(X,4).
e(X,Y)             -> e(X,0).
e(X,Y), X+Y <= 4 -> e(0,Y+X).
e(X,Y), X+Y >  4 -> e(X - (4-Y), 4).
e(X,Y), X+Y <= 3 -> e(X+Y, 0).
e(X,Y), X+Y >  3 -> e(3, Y - (3-X)).
end_of_list.

list(sos).
-> e(0,0).  % Estado inicial
e(X,2) ->.  % Estado final
end_of_list.
```

- **Prueba**

```
 2 [] e(X,Y) -> e(0,Y).
 3 [] e(X,Y) -> e(X,4).
 7 [] e(X,Y), X+Y<=3 -> e(X+Y,0).
 8 [] e(X,Y), X+Y>3 -> e(3,Y- (3-X)).
 9 []  -> e(0,0).
10 [] e(X,2) -> .
11 [hyper,9,3]  -> e(0,4).
13 [hyper,11,8,eval,demod]  -> e(3,1).
16 [hyper,13,2]  -> e(0,1).
18 [hyper,16,7,eval,demod]  -> e(1,0).
20 [hyper,18,3]  -> e(1,4).
22 [hyper,20,8,eval,demod]  -> e(3,2).
23 [binary,22.1,10.1]  -> .
```

Planificación: Problema del mono



- **Representación:**
  `vale(pos_mono(X),pos_platano(Y),pos_silla(Z),Plan)`
  significa que en el estado obtenido aplicando el `Plan` **(inverso) al estado inicial se verifica que la posición del mono es** `X`**, la del plátano es** `Y` **y la de la silla es** `Z`

- **Entrada** `mono.in`

```
set(prolog_style_variables).
set(input_sequent).
set(output_sequent).
set(ur_res).

list(usable).
posicion(X), posicion(Y),
vale(pos_mono(X),pos_platano(Pp),pos_silla(Ps),Plan)
->
vale(pos_mono(Y),pos_platano(Pp),pos_silla(Ps),
     [andar(X,Y)|Plan]).

posicion(Y),
vale(pos_mono(X),pos_platano(Pp),pos_silla(X),Plan)
->
vale(pos_mono(Y),pos_platano(Pp),pos_silla(Y),
     [empujar(X,Y)|Plan]).
```

Planificación: Problema del mono

```
vale(pos_mono(P),pos_platano(P),pos_silla(P),Plan)
->
coge_platano([subir|Plan]).
end_of_list.

list(sos).
-> posicion(a).
-> posicion(b).
-> posicion(c).

-> vale(pos_mono(a),pos_platano(b),pos_silla(c),[]).

coge_platano(Plan) -> resp(inversa(Plan,[])).
end_of_list.

list(passive).
resp(Plan) -> $ans(Plan).
end_of_list.

list(demodulators).
-> inversa([X|L1],L2) = inversa(L1,[X|L2]).
-> inversa([],L) = L.
end_of_list.
```

Planificación: Problema del mono

```
1 [] posicion(X), posicion(Y),
     vale(pos_mono(X),pos_platano(Pp),pos_silla(Ps),Plan)
     -> vale(pos_mono(Y),pos_platano(Pp),pos_silla(Ps),
             [andar(X,Y)|Plan]).
2 [] posicion(Y),
     vale(pos_mono(X),pos_platano(Pp),pos_silla(X),Plan)
     -> vale(pos_mono(Y),pos_platano(Pp),pos_silla(Y),
             [empujar(X,Y)|Plan]).
3 [] vale(pos_mono(P),pos_platano(P),pos_silla(P),Plan)
     -> coge_platano([subir|Plan]).
4 [] -> posicion(a).
5 [] -> posicion(b).
6 [] -> posicion(c).
7 [] -> vale(pos_mono(a),pos_platano(b),pos_silla(c),[]).
8 [] coge_platano(Plan) -> resp(inversa(Plan,[])).
9 [] resp(Plan) -> $ans(Plan).
10 []  -> inversa([X|L1],L2)=inversa(L1,[X|L2]).
11 []  -> inversa([],L)=L.
12 [hyper,7,1,4,6]
   -> vale(pos_mono(c),pos_platano(b),pos_silla(c),
           [andar(a,c)]).
16 [hyper,12,2,5]
   -> vale(pos_mono(b),pos_platano(b),pos_silla(b),
           [empujar(c,b),andar(a,c)]).
33 [hyper,16,3]
   -> coge_platano([subir,empujar(c,b),andar(a,c)]).
40 [hyper,33,8,demod,10,10,10,11]
   -> resp([andar(a,c),empujar(c,b),subir]).
41 [binary,40.1,9.1]
   -> $ans([andar(a,c),empujar(c,b),subir]).
```

Problema de Robbins

- **Axiomas de Huntington (1933):**
  - **(A)** $\mathbf{(x + y) + z = x + (y + z)}$
  - **(C)** $\mathbf{x + y = y + x}$
  - **(H)** $\mathbf{n(n(x) + y) + n(n(x) + n(y)) = x}$
- **Axioma de Robbins (1933):**
  - **(R)** $\mathbf{n(n(n(y) + x) + n(x + y)) = x}$
- **Teorema: (A)+(C)+(H) $\Longrightarrow$ (R)**
- **Problema de Robbins: (A)+(C)+(R) $\Longrightarrow$ (H)**
- **Lemas (Winkler, 1990):**
  - **(A)+(C)+(R)+**$\mathbf{(\exists c)(\exists d)[c + d = c] \Longrightarrow}$ **(H)**
  - **(A)+(C)+(R)+**$\mathbf{(\exists c)(\exists d)[n(c + d) = n(c)] \Longrightarrow}$ **(H)**
- **Teorema (McCune, 1996):**
  - **(A)+(C)+(R)** $\Longrightarrow \mathbf{(\exists c)(\exists d)[c + d = c]}$
- **Entrada a EQP**

```
n(n(n(y)+x)+n(x+y))=x.
x+y != x.
n(x+y) != n(x).
```

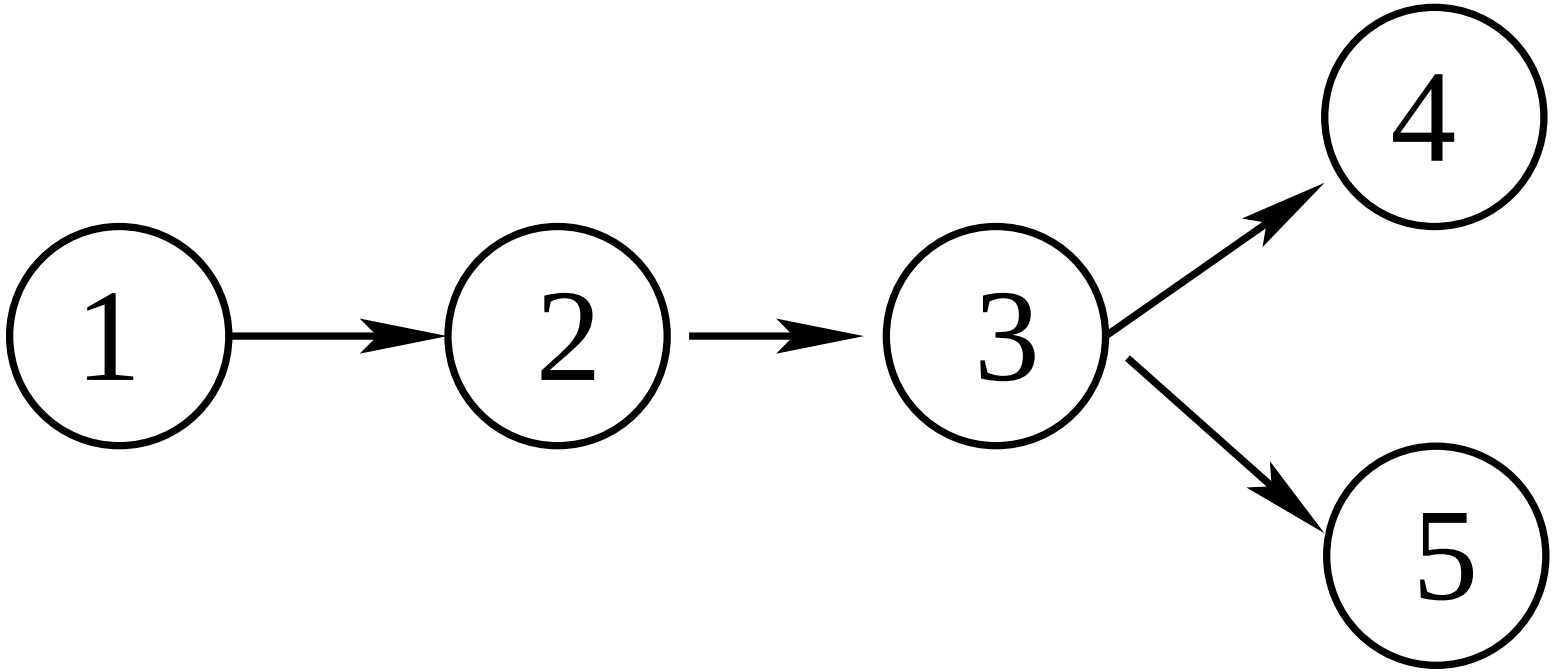
Problema de Robbins

```
2     []           -(n(x+y)=n(x)).
3     []           n(n(n(x)+y)+n(x+y))=y.
5     [3,3]        n(n(n(x+y)+n(x)+y)+y)=n(x+y).
6     [3,3]        n(n(n(n(x)+y)+x+y)+y)=n(n(x)+y).
24    [6,3]        n(n(n(n(x)+y)+x+2y)+n(n(x)+y))=y.
47    [24,3]       n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+z)+n(y+z))=z.
48    [24,3]       n(n(n(n(x)+y)+n(n(x)+y)+x+2y)+y)=n(n(x)+y).
146   [48,3]       n(n(n(n(x)+y)+n(n(x)+y)+x+3y)+n(n(x)+y))=y.
250   [47,3]       n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+n(y+z)+z)+z)=n(y+z).
996   [250,3]      n(n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+n(y+z)+z)+z+u)+n(n(y+z)+u))=u.
16379 [5,996,3]    n(n(n(n(x)+x)+3x)+x)=n(n(x)+x).
16387 [16379,3]    n(n(n(n(n(x)+x)+3x)+x+y)+n(n(n(x)+x)+y))=y.
16388 [16379,3]    n(n(n(n(x)+x)+4x)+n(n(x)+x))=x.
16393 [16388,3]    n(n(n(n(x)+x)+n(n(x)+x)+4x)+x)=n(n(x)+x).
16426 [16393,3]    n(n(n(n(n(x)+x)+n(n(x)+x)+4x)+x+y)+n(n(n(x)+x)+y))=y.
17547 [146,16387]  n(n(n(n(n(x)+x)+n(n(x)+x)+4x)+n(n(n(x)+x)+3x)+x)+x)
                   =n(n(n(x)+x)+n(n(x)+x)+4x).
17666 [24,16426,17547] n(n(n(x)+x)+n(n(x)+x)+4x)=n(n(n(x)+x)+3x).

n(c+d) = n(c), c = n(n(x)+x)+3x, d = n(n(x)+x)+x
```

Aprendizaje automático con FOIL

- Grafo



Aprendizaje automático con FOIL

- **Representación** `camino.pl`
  - **Parámetros**

```
foil_predicates([camino/2, enlace/2]).
foil_cwa(true).                % Usa la hipótesis del mundo cerrado
foil_use_negations(false).     % No usa información negativa
foil_det_lit_bound(0).         % No añade literales determinados
```

  - **Ejemplos**

```
enlace(1,2).  enlace(2,3).  enlace(3,4).  enlace(3,5).
camino(1,2).  camino(1,3).  camino(1,4).  camino(1,5).
camino(2,3).  camino(2,4).  camino(2,5).
camino(3,4).  camino(3,5).
```

Aprendizaje automático con FOIL

- **Sesión**

```
?- [foil,camino].
?- foil(camino/2).
Uncovered positives: [(1,2),(1,3),(1,4),(1,5),(2,3),(2,4),(2,5),(3,4),(3,5)]

Adding a clause ...
Specializing current clause: camino(A,B).

Covered negatives: [(1,1),(2,1),(2,2),(3,1),(3,2),(3,3),(4,1),(4,2),(4,3),
                    (4,4),(4,5),(5,1),(5,2),(5,3),(5,4),(5,5)]
Covered positives: [(1,2),(1,3),(1,4),(1,5),(2,3),(2,4),(2,5),(3,4),(3,5)]

Ganancia: -2.630 Cláusula: camino(A,B):-enlace(C,A)
Ganancia:  5.503 Cláusula: camino(A,B):-enlace(A,C)
Ganancia:  2.897 Cláusula: camino(A,B):-enlace(C,B)
Ganancia: -1.578 Cláusula: camino(A,B):-enlace(B,C)
Ganancia:  0.000 Cláusula: camino(A,B):-enlace(A,A)
Ganancia:  0.000 Cláusula: camino(A,B):-enlace(B,A)
Ganancia:  5.896 Cláusula: camino(A,B):-enlace(A,B)
Ganancia:  0.000 Cláusula: camino(A,B):-enlace(B,B)
```

Aprendizaje automático con FOIL

```
Clause found: camino(A,B) :- enlace(A,B).

Uncovered positives: [(1,3),(1,4),(1,5),(2,4),(2,5)]

Adding a clause ...
Specializing current clause: camino(A,B).

Covered negatives: [(1,1),(2,1),(2,2),(3,1),(3,2),(3,3),(4,1),(4,2),(4,3),
                    (4,4),(4,5),(5,1),(5,2),(5,3),(5,4),(5,5)]
Covered positives: [(1,3),(1,4),(1,5),(2,4),(2,5)]

Ganancia: -2.034 Cláusula: camino(A,B):-enlace(C,A)
Ganancia:  2.925 Cláusula: camino(A,B):-enlace(A,C)
Ganancia:  1.962 Cláusula: camino(A,B):-enlace(C,B)
Ganancia: -1.017 Cláusula: camino(A,B):-enlace(B,C)
Ganancia:  0.000 Cláusula: camino(A,B):-enlace(A,A)
Ganancia:  0.000 Cláusula: camino(A,B):-enlace(B,A)
Ganancia:  0.000 Cláusula: camino(A,B):-enlace(A,B)
Ganancia:  0.000 Cláusula: camino(A,B):-enlace(B,B)
```

Aprendizaje automático con FOIL

```
Specializing current clause: camino(A,B) :- enlace(A,C).

Covered negatives: [(1,1),(2,1),(2,2),(3,1),(3,2),(3,3)]
Covered positives: [(1,3),(1,4),(1,5),(2,4),(2,5)]

Ganancia:  7.427 Cláusula: camino(A,B):-enlace(A,C),camino(C,B)
Ganancia: -1.673 Cláusula: camino(A,B):-enlace(A,C),enlace(D,A)
Ganancia: -2.573 Cláusula: camino(A,B):-enlace(A,C),enlace(A,D)
Ganancia:  2.427 Cláusula: camino(A,B):-enlace(A,C),enlace(D,B)
Ganancia: -1.215 Cláusula: camino(A,B):-enlace(A,C),enlace(B,D)
Ganancia:  3.539 Cláusula: camino(A,B):-enlace(A,C),enlace(C,D)
Ganancia:  4.456 Cláusula: camino(A,B):-enlace(A,C),enlace(C,B)

Clause found: camino(A,B) :- enlace(A,C), camino(C,B).

Found definition:
camino(A,B) :- enlace(A,C), camino(C,B).
camino(A,B) :- enlace(A,B).
```

Aprendizaje automático con Progol

Ejemplo	Acción	Autor	Tema	Longitud	Sitio
e1	saltar	conocido	nuevo	largo	casa
e2	leer	desconocido	nuevo	corto	trabajo
e3	saltar	desconocido	viejo	largo	trabajo
e4	saltar	conocido	viejo	largo	casa
e5	leer	conocido	nuevo	corto	casa
e6	saltar	conocido	viejo	largo	trabajo
e7	saltar	desconocido	viejo	corto	trabajo
e8	leer	desconocido	nuevo	corto	trabajo
e9	saltar	conocido	viejo	largo	casa
e10	saltar	conocido	nuevo	largo	trabajo
e11	saltar	desconocido	viejo	corto	casa
e12	saltar	conocido	nuevo	largo	trabajo
e13	leer	conocido	viejo	corto	casa
e14	leer	conocido	nuevo	corto	trabajo
e15	leer	conocido	nuevo	corto	casa
e16	leer	conocido	viejo	corto	trabajo
e17	leer	conocido	nuevo	corto	casa
e18	leer	desconocido	nuevo	corto	trabajo

Aprendizaje automático con Progol

- **Conocimiento base**

```
autor(e1,conocido).   autor(e2,desconocido).   ... autor(e18,desconocido).
tema(e1,nuevo).       tema(e2,nuevo).          ... tema(e18, nuevo).
longitud(e1,largo).   longitud(e2,corto).      ... longitud(e18,corto).
sitio(e1,casa).       sitio(e2,trabajo).       ... sitio(e18, trabajo).
```

- **Ejemplos positivos**

```
accion(e1,saltar).    accion(e2,leer).         ... accion(e18,leer).
```

- **Restricciones**

```
:- hypothesis(Cabeza,Cuerpo,_),
   accion(A,C),
   Cuerpo,
   Cabeza = accion(A,B),
   B \= C.
```

Aprendizaje automático con Progol

- **Sesión**

```
> progol softbot
CProgol Version 4.4
...
[Testing for contradictions]
[No contradictions found]
[Generalising accion(e1,saltar).]
[Most specific clause is]

accion(A,saltar) :-
  autor(A,conocido), tema(A,nuevo), longitud(A,largo), sitio(A,casa).
```

Aprendizaje automático con Progol

```
[Learning accion/2 from positive examples]
[C:-39932,18,10000,0 accion(A,saltar).]
[C:-39936,18,10000,0 accion(A,saltar) :- autor(A,conocido).]
[C:-39936,18,10000,0 accion(A,saltar) :- tema(A,nuevo).]
[C:34,28,13,0 accion(A,saltar) :- longitud(A,largo).]
[C:13,12,7,0 accion(A,saltar) :- longitud(A,largo), sitio(A,casa).]
[C:-39936,18,10000,0 accion(A,saltar) :- sitio(A,casa).]
[C:-39940,18,10000,0 accion(A,saltar) :- autor(A,conocido), tema(A,nuevo).]
[C:31,24,11,0 accion(A,saltar) :- autor(A,conocido), longitud(A,largo).]
[C:7,12,7,0 accion(A,saltar) :- autor(A,conocido), longitud(A,largo), sitio(A,casa).]
[C:-39940,18,10000,0 accion(A,saltar) :- autor(A,conocido), sitio(A,casa).]
[C:25,12,5,0 accion(A,saltar) :- tema(A,nuevo), longitud(A,largo).]
[C:-39940,18,10000,0 accion(A,saltar) :- tema(A,nuevo), sitio(A,casa).]
[C:19,12,5,0 accion(A,saltar) :- autor(A,conocido), tema(A,nuevo), longitud(A,largo).]
[C:-39944,18,10000,0 accion(A,saltar) :- autor(A,conocido), tema(A,nuevo), sitio(A,casa)
[14 explored search nodes]
f=34,p=28,n=13,h=0
[Result of search is]

accion(A,saltar) :- longitud(A,largo).
```

Aprendizaje automático con Progol

```
[2 redundant clauses retracted]
accion(A,saltar) :- longitud(A,largo).
accion(A,leer) :- tema(A,nuevo), longitud(A,corto).
accion(A,saltar) :- autor(A,desconocido), tema(A,viejo).
accion(A,leer) :- autor(A,conocido), longitud(A,corto).
[Total number of clauses = 4]

[Time taken 0.090s]
```