Tema 9: Aplicaciones de SBC en CLIPS

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo
Francisco J. Martín Mateos

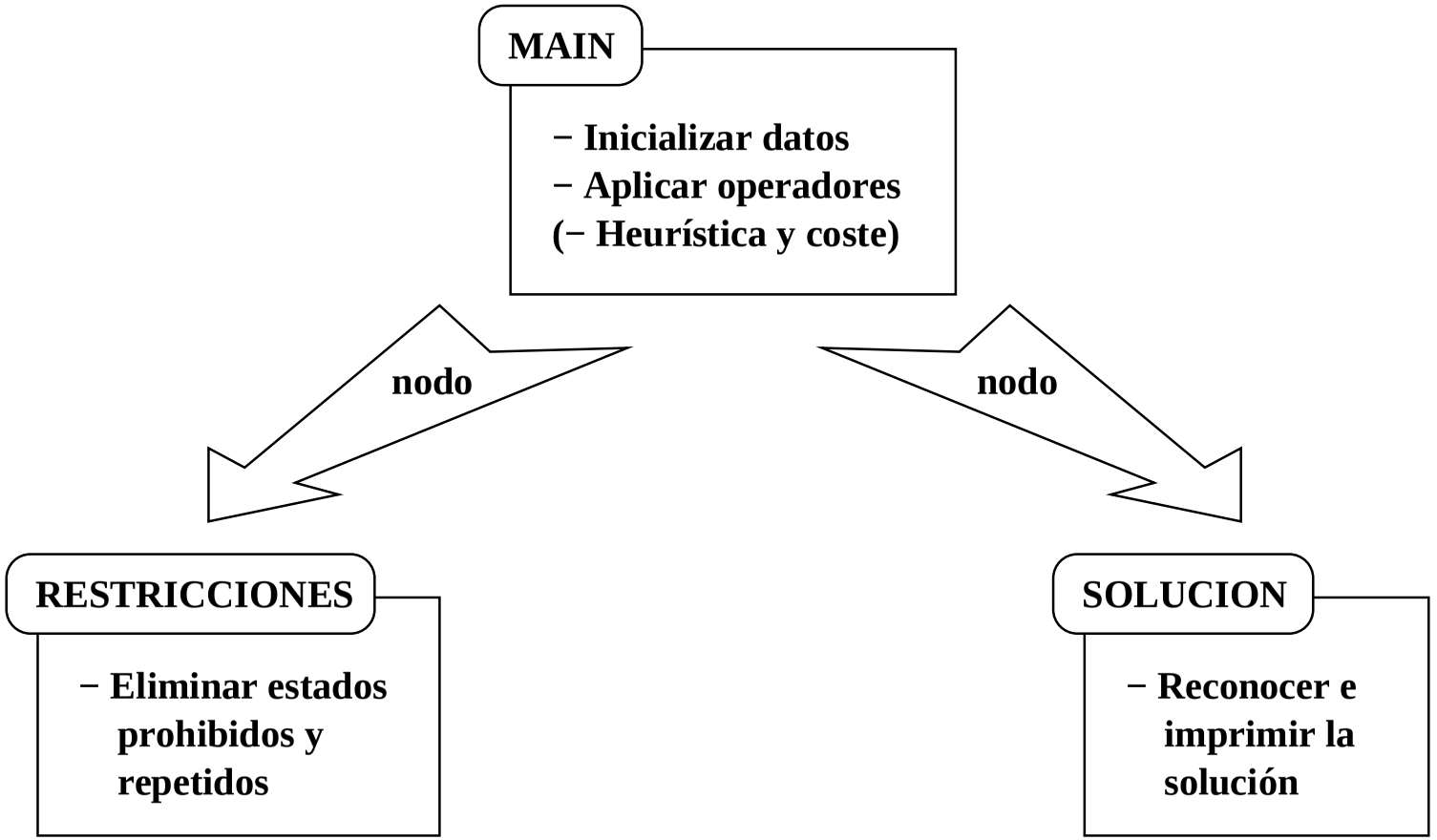
Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Aplicaciones de SBC en CLIPS

- **Búsqueda**
  - **Búsqueda ciega**
  - **Búsqueda con heurística**
  - **Búsqueda con coste y heurística**
- **Sistemas de control**
- **Árboles de decisión**
- **Bibliografía:**
  - **Giarratano, J.C. y Riley, G. "Expert Systems Principles and Programming (2nd ed.)" (PWS Pub. Co., 1994)**
    *** Cap. 12.3: "Decision Trees"**
    *** Cap. 12.5: "A Monitoring Problem"**

Esquema general de los problemas de búsqueda



Búsqueda ciega

- **Enunciado**
  - **La situación inicial es**

  - **La situación final es**

  - **Los movimientos permitidos consisten en desplazar una ficha al hueco saltando, como máximo, sobre otras dos.**

- **Módulo MAIN**

```
(defmodule MAIN
  (export deftemplate nodo))

(deftemplate MAIN::nodo
  (multislot estado)
  (multislot camino))

(deffacts MAIN::nodo-inicial
  (nodo (estado B B B H V V V)
        (camino "B B B H V V V")))
```

Búsqueda ciega

```
(defrule MAIN::movimiento-izquierda
  (nodo (estado $?x
                H
                $?y&:(<= (length $?y) 2)
                ?ficha
                $?z)
        (camino $?movimientos))
  =>
  (bind $?nuevo-estado
        (create$ $?x ?ficha $?y H $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado)))))

(defrule MAIN::movimiento-derecha
  (nodo (estado $?x
                ?ficha
                $?y&:(<= (length $?y) 2)
                H
                $?z)
        (camino $?movimientos))
  =>
  (bind $?nuevo-estado
        (create$ $?x H $?y ?ficha $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado)))))
```

Búsqueda ciega

- **Módulo RESTRICCIONES**

```
(defmodule RESTRICCIONES
  (import MAIN deftemplate nodo))

(defrule RESTRICCIONES::repeticion-de-nodo
  (declare (auto-focus TRUE))
  (nodo (estado $?actual)
        (camino $?movimientos))
  ?nodo <- (nodo (estado $?actual)
                 (camino $?movimientos ? $?))
  =>
  (retract ?nodo))
```

Búsqueda ciega

• Módulo SOLUCION

```
(defmodule SOLUCION
  (import MAIN deftemplate nodo))

(defrule SOLUCION::reconoce-solucion
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (estado V V V H B B B)
                 (camino $?estados))
  =>
  (retract ?nodo)
  (assert (solucion $?estados)))

(defrule SOLUCION::escribe-solucion
  (solucion $?estados)
  =>
  (bind ?longitud (length $?estados))
  (printout t "La solucion, de longitud " ?longitud
              " es " crlf)
  (loop-for-count (?i 1 ?longitud)
    (bind ?estado (nth ?i $?estados))
    (printout t ?estado crlf))
  (retract *))
```

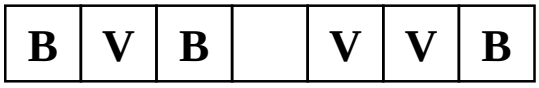
Búsqueda ciega

- **Sesión con estadística**

```
CLIPS> (load "fichas-1.clp")
+%$*******
TRUE
CLIPS> (watch statistics)
CLIPS> (reset)
CLIPS> (run)
La solucion, de longitud 83 es:
B B B H V V V
H B B B V V V
B H B B V V V
B B H B V V V
B B V B H V V
.............
V B V V H B B
V H V V B B B
H V V V B B B
V V H V B B B
V V V H B B B
238 rules fired        Run time is 34.50 seconds.
6.898550724637682 rules per second.
43 mean number of facts (84 maximum).
1 mean number of instances (1 maximum).
103 mean number of activations (205 maximum).
```

Búsqueda con heurística

- **Heurística:**
  - **Definición: La heurística de un estado es la suma de piezas blancas situadas a la izquierda de cada una de las piezas verdes.**
  - **Ejemplo: La heurística del siguiente estado es 1+2+2 = 5.**



- **Módulo MAIN**

```
(defmodule MAIN
  (export deftemplate nodo))

(deftemplate MAIN::nodo
  (multislot estado)
  (multislot camino)
  (slot heuristica)
  (slot clase (default abierto)))

(defglobal MAIN
  ?*estado-inicial* = (create$ B B B H V V V))

(defrule MAIN::inicial
  =>
  (assert (nodo (estado ?*estado-inicial*)
                (camino (implode$ ?*estado-inicial*))
                (heuristica
                      (heuristica ?*estado-inicial*))
                (clase cerrado))))
```

Búsqueda con heurística

```
(deffunction MAIN::heuristica ($?estado)
  (bind ?resultado 0)
  (bind ?verdes 3)
  (loop-for-count (?i 1 7)
    (bind ?ficha (nth ?i $?estado))
    (if (eq ?ficha B)
        then (bind ?resultado
                   (+ ?resultado ?verdes))
        else (if (eq ?ficha V)
                 then (bind ?verdes
                            (- ?verdes 1)))))
  ?resultado)

(defrule MAIN::movimiento-izquierda
  (nodo (estado $?x H
                $?y&:(<= (length $?y) 2)
                ?ficha $?z)
        (camino $?movimientos)
        (clase cerrado))
  =>
  (bind $?nuevo-estado
        (create$ $?x ?ficha $?y H $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (heuristica
                      (heuristica $?nuevo-estado)))))
```

Búsqueda con heurística

```
(defrule MAIN::movimiento-derecha
  (nodo (estado $?x ?ficha
                $?y&:(<= (length $?y) 2)
                H $?z)
        (camino $?movimientos)
        (clase cerrado))
  =>
  (bind $?nuevo-estado
        (create$ $?x H $?y ?ficha $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (heuristica
                      (heuristica $?nuevo-estado)))))

(defrule MAIN::pasa-el-mejor-a-cerrado
  (declare (salience -10))
  ?nodo <- (nodo (clase abierto)
                 (heuristica ?h1))
  (not (nodo (clase abierto)
             (heuristica ?h2&:(< ?h2 ?h1))))
  =>
  (modify ?nodo (clase cerrado)))
```

- **Módulo RESTRICCIONES: Ningún cambio**
- **Módulo SOLUCION: Ningún cambio**

Búsqueda con heurística

- **Sesión con estadística**

```
CLIPS> (watch statistics)
CLIPS> (reset)
CLIPS> (run)
La solucion, de longitud 15 es:
B B B H V V V
B B B V V V H
B B B V V H V
B B H V V B V
B B V V H B V
B H V V B B V
B V V H B B V
B V V V B B H
B V V V B H B
B V V V H B B
B V V H V B B
H V V B V B B
V V H B V B B
V V V B H B B
V V V H B B B
93 rules fired        Run time is 1.83 seconds.
50.7272727270043 rules per second.
26 mean number of facts (49 maximum).
1 mean number of instances (1 maximum).
5 mean number of activations (12 maximum).
```

Búsqueda con heurística y coste

- **Coste de un nodo = número de movimientos**
- **Módulo MAIN**

```
(defmodule MAIN
  (export deftemplate nodo))

(deftemplate MAIN::nodo
  (multislot estado)
  (multislot camino)
  (slot heuristica)
  (slot coste)
  (slot clase (default abierto)))

(defglobal MAIN
  ?*estado-inicial* = (create$ B B B H V V V))

(deffunction MAIN::heuristica ($?estado)
  (bind ?resultado 0)
  (bind ?verdes 3)
  (loop-for-count (?i 1 7)
    (bind ?ficha (nth ?i $?estado))
    (if (eq ?ficha B)
        then (bind ?resultado
                   (+ ?resultado ?verdes))
        else (if (eq ?ficha V)
                 then (bind ?verdes
                            (- ?verdes 1)))))
  ?resultado)
```

Búsqueda con heurística y coste

```
(defrule MAIN::inicial
  =>
  (assert (nodo (estado ?*estado-inicial*)
                (camino (implode$ ?*estado-inicial*))
                (heuristica
                      (heuristica ?*estado-inicial*))
                (coste 0)
                (clase cerrado))))

(defrule MAIN::movimiento-izquierda
  (nodo (estado $?x H
                $?y&:(<= (length $?y) 2)
                ?ficha $?z)
        (camino $?movimientos)
        (coste ?coste)
        (clase cerrado))
  =>
  (bind $?nuevo-estado
        (create$ $?x ?ficha $?y H $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (coste (+ ?coste 1))
                (heuristica
                      (heuristica $?nuevo-estado)))))
```

Búsqueda con heurística y coste

```
(defrule MAIN::movimiento-derecha
  (nodo (estado $?x ?ficha
                $?y&:(<= (length $?y) 2)
                H $?z)
        (camino $?movimientos)
        (coste ?coste)
        (clase cerrado))
  =>
  (bind $?nuevo-estado
        (create$ $?x H $?y ?ficha $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (coste (+ ?coste 1))
                (heuristica
                      (heuristica $?nuevo-estado)))))

(defrule MAIN::pasa-el-mejor-a-cerrado
  (declare (salience -10))
  ?nodo <- (nodo (clase abierto)
                 (heuristica ?h1)
                 (coste ?c1))
  (not (nodo (clase abierto)
             (heuristica ?h2&:(< ?h2 ?h1))))
  (not (nodo (clase abierto) (heuristica ?h1)
             (coste ?c2&:(< ?c2 ?c1))))
  =>
  (modify ?nodo (clase cerrado)))
```

Búsqueda con heurística y coste

- **Módulo RESTRICCIONES**

```
(defmodule RESTRICCIONES
  (import MAIN deftemplate nodo))

(defrule RESTRICCIONES::repeticion-de-nodo
  (declare (auto-focus TRUE))
  (nodo (estado $?actual)
        (coste ?coste-1))
  ?nodo <- (nodo (estado $?actual)
                 (coste ?coste-2&:(> ?coste-2 ?coste-1)))
  =>
  (retract ?nodo))
```

- **Módulo SOLUCION: Ningún cambio**

Búsqueda con heurística y coste

- **Sesión con estadística**

```
CLIPS> (load "fichas-3.clp")
+%:!*********
TRUE
CLIPS> (watch statistics)
CLIPS> (reset)
CLIPS> (run)
La solucion, de longitud 13 es:
B B B H V V V
B B B V V H V
B B H V V B V
B B V V H B V
B H V V B B V
B V V H B B V
H V V B B B V
V V H B B B V
V V B H B B V
V V B V B B H
V V B V B H B
V V H V B B B
V V V H B B B
120 rules fired        Run time is 0.65 seconds.
184.6153846170378 rules per second.
25 mean number of facts (47 maximum).
1 mean number of instances (1 maximum).
5 mean number of activations (11 maximum).
```

Comparación de soluciones

	Solución 1	Solución 2 (heurística)	Solución 3 (heurística y coste)
Longitud de la solución	83	15	13
Tiempo (segundos)	34.50	1.83	0.65
Número de disparos	238	93	120
Número máximo de hechos	84	49	47
Número medio de hechos	43	26	25
Número máximo de activaciones	205	12	11
Número medio de activaciones	103	5	5

Problema del 8-puzzle

- **Enunciado**

2	**8**	**3**
1	**6**	**4**
7		**5**

Estado inicial

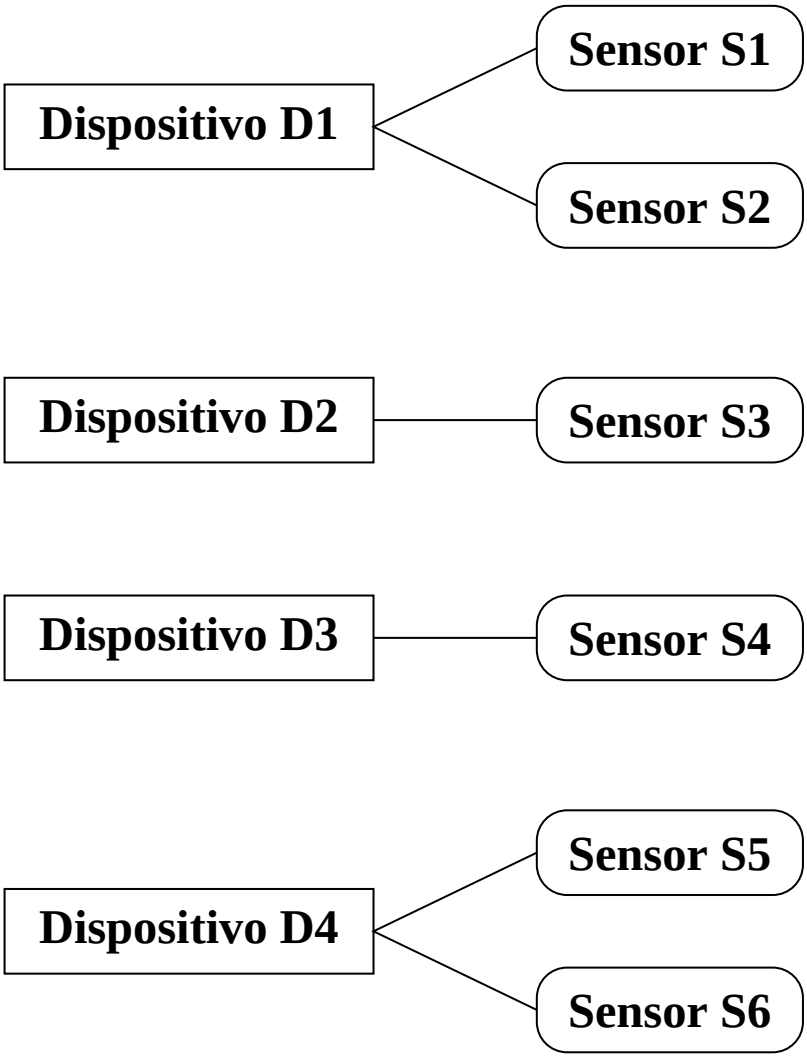
1	**2**	**3**
8		**4**
7	**6**	**5**

Estado final

- **Heurística**
  - **Definición: número de piezas descolocadas**
  - **Heurística del estado inicial: 4**

Sistemas de control

- Sistema formado por 4 dispositivos con sensores asociados:



Sistemas de control

- **Sensor:**
  - **Límite Crítico Alto**
  - **Límite Peligroso Alto**
  - **Límite Peligroso Bajo**
  - **Límite Crítico Bajo**
- **Límites asociados a los sensores:**

Sensor	LCA	LPA	LPB	LCB	N
S1	130	120	70	60	3c
S2	180	160	40	20	5c
S3	130	120	70	60	4c
S4	130	120	70	60	4c
S5	125	120	70	65	4c
S6	130	125	115	110	2c

- **Dispositivos:**
  - **Estado crítico: Sensor asociado con Valor $\geq$ LCA o Valor $\leq$ LCB. Desconectar dispositivo**
  - **Estado peligroso: Sensor asociado con LCA $>$ Valor $\geq$ LPA o LPB $\geq$ Valor $>$ LCB. Descontectar dispositivo si la situación se mantiene durante N ciclos.**

Sistemas de control

- **Ciclo:**
  - **Leer datos de entrada**
  - **Detectar estados de los dispositivos**
  - **Realizar acciones**
- **Fuente de datos:**
  - **Hechos**
  - **Sensores**
  - **Usuario**
  - **Fichero**
- **Final:**
  - **Todos los dispositivos desconectados**
  - **Límite de ciclos**
- **Valores de los sensores por ciclos:**

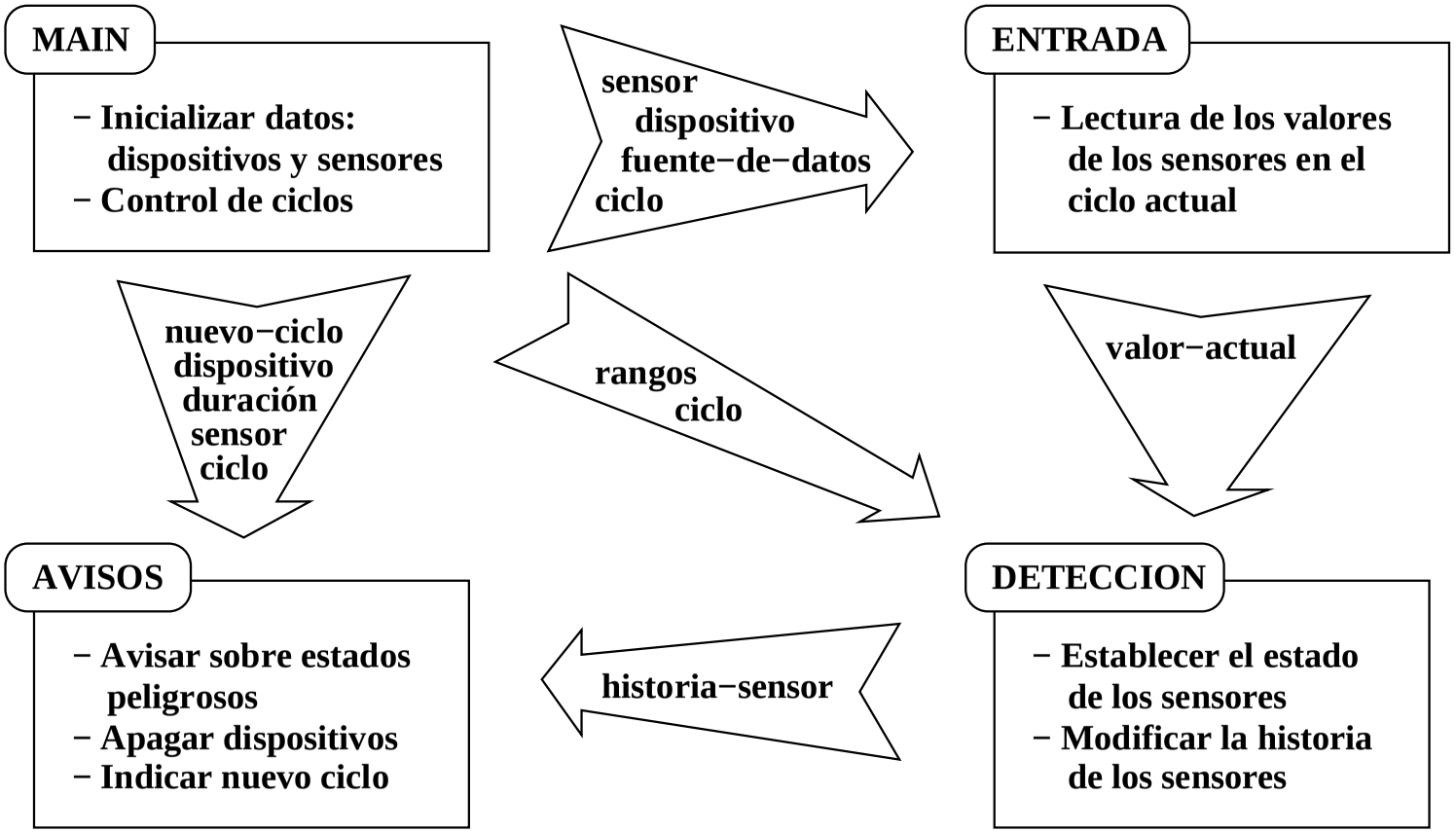
Sensor	Ciclo1	Ciclo2	Ciclo3	Ciclo4	Ciclo5	Ciclo6
S1	100	100	110	110	115	120
S2	110	120	125	130	130	135
S3	100	120	125	130	130	125
S4	120	120	120	125	130	135
S5	110	120	125	130	135	135
S6	115	120	125	135	130	135

Sistemas de control

- **Sesión**

```
CLIPS> (load "monitor.clp")
...
CLIPS> (run)
Ciclo 1 - Sensor S6 en peligroso-bajo  durante 1 ciclos.
Ciclo 1 - Sensor S4 en peligroso-alto  durante 1 ciclos.
Ciclo 2 - Sensor S5 en peligroso-alto  durante 1 ciclos.
Ciclo 2 - Sensor S4 en peligroso-alto  durante 2 ciclos.
Ciclo 2 - Sensor S3 en peligroso-alto  durante 1 ciclos.
Ciclo 3 - Sensor S6 en peligroso-alto  durante 1 ciclos.
Ciclo 3 - Sensor S5 en critico-alto.
  Apagar el dispositivo D4.
Ciclo 3 - Sensor S4 en peligroso-alto  durante 3 ciclos.
Ciclo 3 - Sensor S3 en peligroso-alto  durante 2 ciclos.
Ciclo 4 - Sensor S4 en peligroso-alto  durante 4 ciclos.
  Apagar el dispositivo D3.
Ciclo 4 - Sensor S3 en critico-alto.
  Apagar el dispositivo D2.
Ciclo 6 - Sensor S1 en peligroso-alto  durante 1 ciclos.
```

Esquema del sistema de control



Sistemas de control

- **Módulo MAIN**

```
(defmodule MAIN
  (export deftemplate ?ALL))

(deftemplate MAIN::dispositivo
  (slot nombre)
  (slot estado
    (default activo)))

(deffacts MAIN::informacion-dispositivos
  (dispositivo (nombre D1))
  (dispositivo (nombre D2))
  (dispositivo (nombre D3))
  (dispositivo (nombre D4)))

(deftemplate MAIN::sensor
  (slot nombre)
  (slot dispositivo-asociado))

(deffacts MAIN::informacion-sensores
  (sensor (nombre S1) (dispositivo-asociado D1))
  (sensor (nombre S2) (dispositivo-asociado D1))
  (sensor (nombre S3) (dispositivo-asociado D2))
  (sensor (nombre S4) (dispositivo-asociado D3))
  (sensor (nombre S5) (dispositivo-asociado D4))
  (sensor (nombre S6) (dispositivo-asociado D4)))
```

Sistemas de control

```
(deftemplate MAIN::rangos
  (slot nombre)
  (slot minimo-critico) (slot minimo-peligroso)
  (slot maximo-peligroso) (slot maximo-critico))

(deffacts MAIN::rangos-de-los-sensores
  (rangos (nombre S1) (minimo-critico 60)
          (minimo-peligroso 70)
          (maximo-peligroso 120)
          (maximo-critico 130))
  (rangos (nombre S2) (minimo-critico 20)
          (minimo-peligroso 40)
          (maximo-peligroso 160)
          (maximo-critico 180))
  (rangos (nombre S3) (minimo-critico 60)
          (minimo-peligroso 70)
          (maximo-peligroso 120)
          (maximo-critico 130))
  (rangos (nombre S4) (minimo-critico 60)
          (minimo-peligroso 70)
          (maximo-peligroso 120)
          (maximo-critico 130))
  (rangos (nombre S5) (minimo-critico 65)
          (minimo-peligroso 70)
          (maximo-peligroso 120)
          (maximo-critico 125))
  (rangos (nombre S6) (minimo-critico 110)
          (minimo-peligroso 115)
          (maximo-peligroso 125)
          (maximo-critico 130)))
```

Sistemas de control

```
(deftemplate MAIN::duracion
  (slot sensor)
  (slot tiempo))

(deffacts MAIN::duracion-sensores-en-estado-peligroso
  (duracion (sensor S1) (tiempo 3))
  (duracion (sensor S2) (tiempo 5))
  (duracion (sensor S3) (tiempo 4))
  (duracion (sensor S4) (tiempo 4))
  (duracion (sensor S5) (tiempo 4))
  (duracion (sensor S6) (tiempo 2)))

(deffacts MAIN::comienzo
  (fuente-de-datos hechos)
  (ciclo 0)
  (nuevo-ciclo 1))

(defrule MAIN::control
  ?h1 <- (ciclo ?)
  ?h2 <- (nuevo-ciclo ?nciclo&:(< ?nciclo 7))
  =>
  (retract ?h1 ?h2)
  (assert (ciclo ?nciclo))
  (focus ENTRADA DETECCION AVISOS))
```

Sistemas de control

• Módulo ENTRADA

```
(defmodule ENTRADA
  (import MAIN deftemplate dispositivo
                           sensor
                           ciclo
                           fuente-de-datos)
  (export deftemplate valor-actual))

(deftemplate ENTRADA::valor-actual
  (slot sensor)
  (slot valor)
  (slot ciclo))

(deffacts ENTRADA::hechos-de-valores
  (hecho-de-datos S1 100 100 110 110 115 120)
  (hecho-de-datos S2 110 120 125 130 130 135)
  (hecho-de-datos S3 100 120 125 130 130 125)
  (hecho-de-datos S4 120 120 120 125 130 135)
  (hecho-de-datos S5 110 120 125 130 135 135)
  (hecho-de-datos S6 115 120 125 135 130 135))

(defrule ENTRADA::no-hay-dispositivos-encendidos
  (not (dispositivo (estado activo)))
  =>
  (printout t "Todos los dispositivos "
            "estan apagados. " crlf)
  (halt))
```

Sistemas de control

```
(defrule ENTRADA::toma-datos-de-hechos
  (fuente-de-datos hechos)
  (ciclo ?ciclo)
  (sensor (nombre ?id)
          (dispositivo-asociado ?dis))
  (dispositivo (nombre ?dis)
               (estado activo))
  ?h <- (hecho-de-datos ?id ?valor $?resto)
  (not (valor-actual (sensor ?id)
                     (ciclo ?ciclo)))
  =>
  (retract ?h)
  (assert (hecho-de-datos ?id ?resto))
  (assert (valor-actual (sensor ?id)
                        (valor ?valor)
                        (ciclo ?ciclo))))
```

Sistemas de control

```
(defrule ENTRADA::toma-datos-del-sensor
  (fuente-de-datos sensores)
  (ciclo ?ciclo)
  (sensor (nombre ?id) (dispositivo-asociado ?dis))
  (dispositivo (nombre ?dis) (estado activo))
  =>
  (bind ?valor (toma-valor-del-sensor ?id))
  (assert (valor-actual (sensor ?id) (valor ?valor)
                        (ciclo ?ciclo))))

(defrule ENTRADA::toma-datos-del-usuario
  (fuente-de-datos usuario)
  (ciclo ?ciclo)
  (sensor (nombre ?id) (dispositivo-asociado ?dis))
  (dispositivo (nombre ?dis) (estado activo))
  =>
  (printout t "Introduce el valor para el sensor "
            ?id "." crlf)
  (bind ?valor (read))
  (if (not (numberp ?valor)) then (halt))
  (assert (valor-actual (sensor ?id) (valor ?valor)
                        (ciclo ?ciclo))))

(defrule ENTRADA::toma-datos-de-fichero
  (fuente-de-datos fichero)
  ...)
```

Sistemas de control

• Módulo DETECCION

```
(defmodule DETECCION
  (import MAIN deftemplate rangos
                           ciclo)
  (import ENTRADA deftemplate valor-actual)
  (export deftemplate historia-sensor))

(deftemplate DETECCION::estado-sensor
  (slot sensor)
  (slot estado)
  (slot valor)
  (slot ciclo))

(deftemplate DETECCION::historia-sensor
  (slot sensor)
  (slot estado)
  (slot valor)
  (slot ciclo-comienzo)
  (slot ciclo-final))

(deffacts DETECCION::historia-inicial
  (historia-sensor (sensor S1))
  (historia-sensor (sensor S2))
  (historia-sensor (sensor S3))
  (historia-sensor (sensor S4))
  (historia-sensor (sensor S5))
  (historia-sensor (sensor S6)))
```

Sistemas de control

```
(defrule DETECCION::estado-normal
  (ciclo ?ciclo)
  (valor-actual (sensor ?id)
                (valor ?valor)
                (ciclo ?ciclo))
  (rangos (nombre ?id)
          (minimo-peligroso ?v1&:(> ?valor ?v1))
          (maximo-peligroso ?v2&:(< ?valor ?v2)))
  =>
  (assert (estado-sensor (sensor ?id)
                         (estado normal)
                         (valor ?valor)
                         (ciclo ?ciclo))))

(defrule DETECCION::estado-peligroso-bajo
  (ciclo ?ciclo)
  (valor-actual (sensor ?id)
                (valor ?valor)
                (ciclo ?ciclo))
  (rangos (nombre ?id)
          (minimo-critico ?v1&:(> ?valor ?v1))
          (minimo-peligroso ?v2&:(<= ?valor ?v2)))
  =>
  (assert (estado-sensor (sensor ?id)
                         (estado peligroso-bajo)
                         (valor ?valor)
                         (ciclo ?ciclo))))
```

Sistemas de control

```
(defrule DETECCION::estado-peligroso-alto
  (ciclo ?ciclo)
  (valor-actual (sensor ?id)
                (valor ?valor)
                (ciclo ?ciclo))
  (rangos (nombre ?id)
          (maximo-peligroso ?v1&:(>= ?valor ?v1))
          (maximo-critico ?v2&:(< ?valor ?v2)))
  =>
  (assert (estado-sensor (sensor ?id)
                         (estado peligroso-alto)
                         (valor ?valor)
                         (ciclo ?ciclo))))

(defrule DETECCION::estado-critico-bajo
  (ciclo ?ciclo)
  (valor-actual (sensor ?id)
                (valor ?valor)
                (ciclo ?ciclo))
  (rangos (nombre ?id)
          (minimo-critico ?v1&:(<= ?valor ?v1)))
  =>
  (assert (estado-sensor (sensor ?id)
                         (estado critico-bajo)
                         (valor ?valor)
                         (ciclo ?ciclo))))
```

Sistemas de control

```
(defrule DETECCION::estado-critico-alto
  (ciclo ?ciclo)
  (valor-actual (sensor ?id)
                (valor ?valor)
                (ciclo ?ciclo))
  (rangos (nombre ?id)
          (maximo-critico ?v1&:(>= ?valor ?v1)))
  =>
  (assert (estado-sensor (sensor ?id)
                         (estado critico-alto)
                         (valor ?valor)
                         (ciclo ?ciclo))))

(defrule DETECCION::historia-no-cambia
  (ciclo ?ciclo)
  ?h <- (historia-sensor
           (sensor ?id)
           (estado ?estado)
           (ciclo-comienzo ?ini)
           (ciclo-final ?fin&=(- ?ciclo 1)))
  (estado-sensor (sensor ?id)
                 (estado ?estado)
                 (valor ?valor)
                 (ciclo ?ciclo))
  =>
  (modify ?h (valor ?valor)
             (ciclo-final ?ciclo)))
```

Sistemas de control

```
(defrule DETECCION::historia-cambia
  (ciclo ?ciclo)
  ?h <- (historia-sensor
           (sensor ?id)
           (estado ?estado)
           (ciclo-comienzo ?ini)
           (ciclo-final ?fin&=(- ?ciclo 1)))
  (estado-sensor (sensor ?id)
                 (estado ?nuevo&~?estado)
                 (valor ?valor)
                 (ciclo ?ciclo))
  =>
  (modify ?h (estado ?nuevo)
             (valor ?valor)
             (ciclo-comienzo ?ciclo)
             (ciclo-final ?ciclo)))

(defrule DETECCION::elimina-hechos
  (ciclo ?ciclo)
  (or ?h <- (estado-sensor (ciclo ~?ciclo))
      ?h <- (valor-actual (ciclo ~?ciclo)))
  =>
  (retract ?h))
```

Sistemas de control

- **Módulo AVISOS**

```
(defmodule AVISOS
  (import MAIN deftemplate dispositivo
                           sensor
                           duracion
                           ciclo
                           nuevo-ciclo)
  (import DETECCION deftemplate historia-sensor))

(defrule AVISOS::inserta-nuevo-ciclo
  (ciclo ?ciclo)
  =>
  (assert (nuevo-ciclo (+ 1 ?ciclo))))

(defrule AVISOS::apaga-en-region-critica
  (ciclo ?ciclo)
  (historia-sensor
          (sensor ?id)
          (estado ?estado&critico-bajo|critico-alto))
  (sensor (nombre ?id)
          (dispositivo-asociado ?dis))
  ?h <- (dispositivo (nombre ?dis)
                     (estado activo))
  =>
  (printout t
    "Ciclo " ?ciclo " - Sensor " ?id " en " ?estado
    "." crlf "  Apagar el dispositivo " ?dis "." crlf)
  (modify ?h (estado inactivo)))
```

Sistemas de control

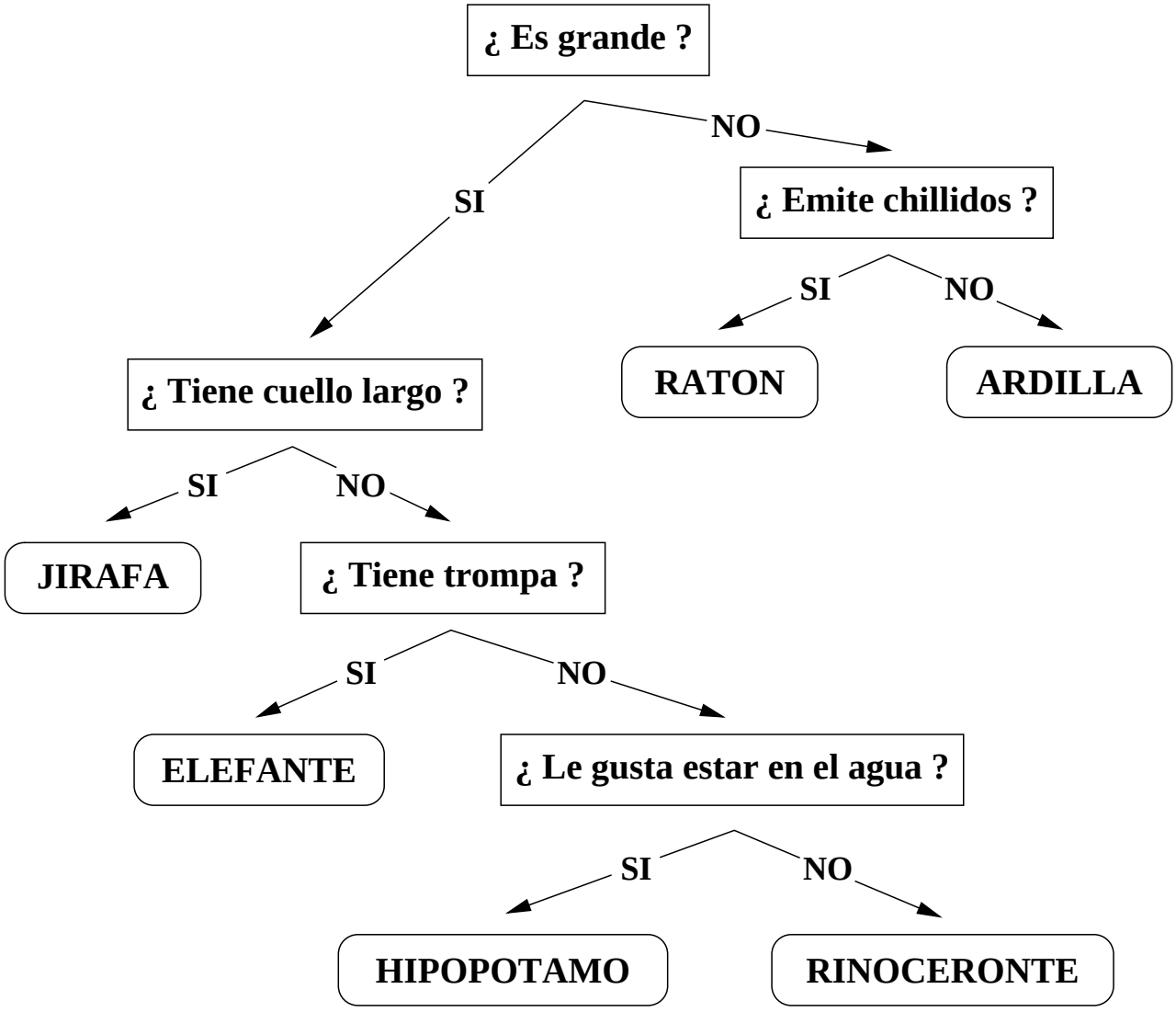
```
(defrule AVISOS::apaga-en-region-peligrosa
  (ciclo ?ciclo)
  (historia-sensor
       (sensor ?id)
       (estado ?estado&peligroso-alto|peligroso-bajo)
       (ciclo-comienzo ?ini)
       (ciclo-final ?fin))
  (duracion (sensor ?id)
            (tiempo ?tiempo&=(+ (- ?fin ?ini) 1)))
  (sensor (nombre ?id)
          (dispositivo-asociado ?dis))
  ?h <- (dispositivo (nombre ?dis)
                     (estado activo))
  =>
  (printout t
    "Ciclo " ?ciclo " - Sensor " ?id " en " ?estado
    " durante " (+ (- ?fin ?ini) 1) " ciclos." crlf
    "  Apagar el dispositivo " ?dis "." crlf)
  (modify ?h (estado inactivo)))
```

Sistemas de control

```
(defrule AVISOS::sensor-en-region-peligrosa
  (ciclo ?ciclo)
  (historia-sensor
       (sensor ?id)
       (estado ?estado&peligroso-alto|peligroso-bajo)
       (ciclo-comienzo ?ini)
       (ciclo-final ?fin))
  (duracion (sensor ?id)
            (tiempo ?tiempo
                  &:(< (+ (- ?fin ?ini) 1) ?tiempo)))
  (sensor (nombre ?id)
          (dispositivo-asociado ?dis))
  (dispositivo (nombre ?dis)
               (estado activo))
  =>
  (printout t
    "Ciclo " ?ciclo " - Sensor " ?id " en " ?estado
    " durante " (+ (- ?fin ?ini) 1) " ciclos." crlf))
```

Árboles de decisión

- Árboles de decisión binarios



Árboles de decisión

- **Sesión: Clasificación**

```
CLIPS> (load "decision.clp")
%%************
TRUE
CLIPS> (reset)
CLIPS> (run)
- Es muy grande.? (si o no): si
- Tiene un cuello largo.? (si o no): no
- Tiene una trompa.? (si o no): si
Creo que es un/a elefante
¿Estoy en lo cierto.? (si o no): si
```

  - **Intentemos clasificar un gato**

```
¿Lo intentamos de nuevo.? (si o no): si
- Es muy grande.? (si o no): no
- Emite chillidos.? (si o no): no
Creo que es un/a ardilla
¿Estoy en lo cierto.? (si o no): no
```

Árboles de decisión

- **Sesión: Aprendizaje**

```
¿Que animal es.?
gato
¿Que pregunta se debe contestar afirmativamente para
   distinguir gato de ardilla.?
Maulla.?
Ahora ya puedo distinguir un/una gato
```

  - **El gato ya esta en la base de conocimiento**

```
¿Lo intentamos de nuevo.? (si o no): si
- Es muy grande.? (si o no): no
- Emite chillidos.? (si o no): no
- Maulla.? (si o no): si
Creo que es un/a gato
¿Estoy en lo cierto.? (si o no): si
¿Lo intentamos de nuevo.? (si o no): no
CLIPS>
```

Árboles de decisión

- **Hojas del árbol de decisión: Respuestas finales.**

```
(deftemplate nodo-respuesta
  (slot nombre)
  (slot valor))
```

- **Nodos internos del árbol de decisión: Preguntas que conducen a la clasificación**

```
(deftemplate nodo-decision
  (slot nombre)
  (slot pregunta)
  (slot respuesta-si)
  (slot respuesta-no))
```

- **Cargar datos de un archivo**

```
(defrule inicializacion
  (not (nodo-decision (nombre raiz)))
  =>
  (load-facts "animales.dat")
  (assert (nodo-actual raiz)))
```

Árboles de decisión

- **Representación del ejemplo (animales.dat)**

```
(nodo-decision (nombre raiz)
      (pregunta "Es muy grande.?")
      (respuesta-si nodo1) (respuesta-no nodo2))

(nodo-decision (nombre nodo1)
      (pregunta "Tiene un cuello largo.?")
      (respuesta-si nodo3) (respuesta-no nodo4))

(nodo-decision (nombre nodo2)
      (pregunta "Emite chillidos.?")
      (respuesta-si nodo5) (respuesta-no nodo6))

(nodo-respuesta (nombre nodo3) (valor jirafa))

(nodo-decision (nombre nodo4)
      (pregunta "Tiene una trompa.?")
      (respuesta-si nodo7) (respuesta-no nodo8))

(nodo-respuesta (nombre nodo5) (valor raton))

(nodo-respuesta (nombre nodo6) (valor ardilla))

(nodo-respuesta (nombre nodo7) (valor elefante))

(nodo-decision (nombre nodo8)
      (pregunta "Le gusta estar en el agua.?")
      (respuesta-si nodo9) (respuesta-no nodo10))

(nodo-respuesta (nombre nodo9) (valor hipopotamo))

(nodo-respuesta (nombre nodo10) (valor rinoceronte))
```

Árboles de decisión

- **Tratamiento de los nodos internos**

```
(defrule pregunta-en-nodo-decision
  (nodo-actual ?nombre)
  (nodo-decision (nombre ?nombre)
                 (pregunta ?pregunta))
  (not (respuesta ?))
  =>
  (printout t "- " ?pregunta " (si o no): ")
  (assert (respuesta (read))))
```

- **Respuesta incorrecta**

```
(defrule respuesta-incorrecta
  ?respuesta <- (respuesta ~si&~no)
  =>
  (retract ?respuesta)
  (printout t "Respuesta incorrecta." crlf))
```

- **Respuesta afirmativa en los nodos internos**

```
(defrule procesa-si-en-nodo-decision
  ?nodo <- (nodo-actual ?nombre)
  (nodo-decision (nombre ?nombre)
                 (respuesta-si ?si))
  ?respuesta <- (respuesta si)
  =>
  (retract ?nodo ?respuesta)
  (assert (nodo-actual ?si)))
```

Árboles de decisión

- **Respuesta negativa en los nodos internos**

```
(defrule procesa-no-en-nodo-decision
  ?nodo <- (nodo-actual ?nombre)
  (nodo-decision (nombre ?nombre)
                 (respuesta-no ?no))
  ?respuesta <- (respuesta no)
  =>
  (retract ?nodo ?respuesta)
  (assert (nodo-actual ?no)))
```

- **Tratamiento de los nodos hoja**

```
(defrule pregunta-en-nodo-respuesta
  (nodo-actual ?nombre)
  (nodo-respuesta (nombre ?nombre) (valor ?valor))
  (not (respuesta ?))
  =>
  (printout t "Creo que es un/a " ?valor crlf)
  (printout t "¿Estoy en lo cierto.? (si o no): ")
  (assert (respuesta (read))))
```

- **Clasificación satisfactoria**

```
(defrule procesa-si-en-nodo-respuesta
  ?nodo <- (nodo-actual ?nombre)
  (nodo-respuesta (nombre ?nombre))
  ?respuesta <- (respuesta si)
  =>
  (retract ?nodo ?respuesta)
  (assert (pregunta-de-nuevo)))
```

Árboles de decisión

- **Clasificación incorrecta**

```
(defrule procesa-no-en-nodo-respuesta
  ?nodo <- (nodo-actual ?nombre)
  (nodo-respuesta (nombre ?nombre))
  ?respuesta <- (respuesta no)
  =>
  (retract ?nodo ?respuesta)
  (assert (reemplaza-nodo-respuesta ?nombre)))
```

- **¿Quieres continuar?**

```
(defrule pregunta-de-nuevo
  (pregunta-de-nuevo)
  (not (respuesta ?))
  =>
  (printout t "¿Lo intentamos de nuevo.? (si o no): ")
  (assert (respuesta (read))))
```

- **Se vuelve a hacer una consulta**

```
(defrule una-vez-mas
  ?fase <- (pregunta-de-nuevo)
  ?respuesta <- (respuesta si)
  =>
  (retract ?fase ?respuesta)
  (assert (nodo-actual raiz)))
```

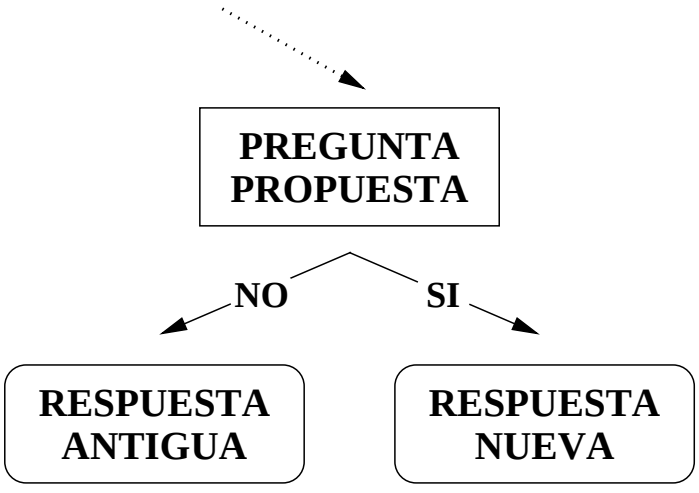
Árboles de decisión

- **Se salvan los datos**

```
(defrule no-mas-por-favor
  ?fase <- (pregunta-de-nuevo)
  ?respuesta <- (respuesta no)
  =>
  (retract ?fase ?respuesta)
  (save-facts "animales.dat"))
```

- **Aprendizaje y modificación de los datos**
  - **Árbol inicial:**

  - **Árbol modificado:**



Árboles de decisión

- **Aprendizaje y modificación de los datos**

```
(defrule aprende
  ?h <- (reemplaza-nodo-respuesta ?nombre)
  ?dato <- (nodo-respuesta (nombre ?nombre)
                           (valor ?valor))
  =>
  (retract ?h ?dato)
  (printout t "¿Que animal es.?" crlf)
  (bind ?animal (read))
  (printout t
    "¿Que pregunta se debe contestar afirmativamente"
    " para distinguir " crlf)
  (printout t "   " ?animal " de " ?valor ".?" crlf)
  (bind ?pregunta (readline))
  (printout t
    "Ahora ya puedo distinguir un/una " ?animal crlf)
  (bind ?nodo1 (gensym*))
  (bind ?nodo2 (gensym*))
  (assert
     (nodo-respuesta (nombre ?nodo1) (valor ?animal))
     (nodo-respuesta (nombre ?nodo2) (valor ?valor))
     (nodo-decision
        (nombre ?nombre) (pregunta ?pregunta)
        (respuesta-si ?nodo1) (respuesta-no ?nodo2))
     (pregunta-de-nuevo)))
```