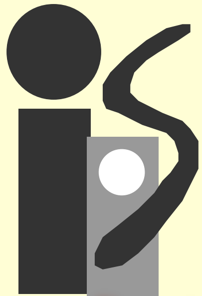
© José Angel Bañares

Resolución de Problemas en LBR



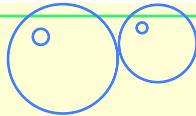
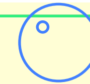
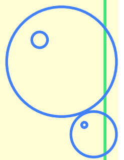
J.A. Bañares Bañares

Departamento de Informática e Ingeniería de Sistemas
C.P.S. Universidad de Zaragoza

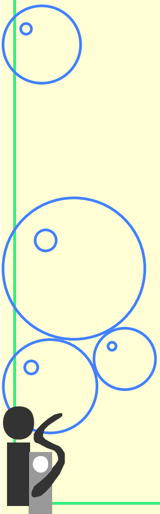
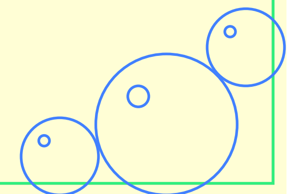
Copyright © 1998 JoséAngel Bañares

Última revisión: Oct., 2009

© José Angel Bañares

- **Objetivo:**
  - Presentación de varios ejemplos programados en CLIPS
- **Ejemplos presentados**
  - Problemas de espacios de estados
  - Árboles de decisión
  - Emulación del Encadenamiento regresivo
  - Problema de monitorización

© José Angel Bañares

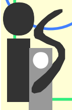
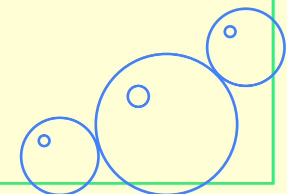
1. Problemas de búsqueda

- **Problema del 8-Puzzle: enunciado**

```
+---+---+---+          +---+---+---+
| 2 | 8 | 3 |          | 1 | 2 | 3 |
+---+---+---+          +---+---+---+
| 1 | 6 | 4 |          | 8 |   | 4 |
+---+---+---+          +---+---+---+
| 7 |   | 5 |          | 7 | 6 | 5 |
+---+---+---+          +---+---+---+
Estado inicial         Estado final
```

- **Modulo principal**

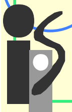
```
(defmodule MAIN (export deftemplate nodo))

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

(deffacts MAIN::nodo-inicial
   (nodo (estado 2 8 3 1 6 4 7 H 5)
         (camino)))
```

© José Angel Bañares

Problema del 8-puzzle

```
(defrule MAIN::arriba
   (nodo (estado $?a ?b ?c ?d H $?e)
          (camino $?movimientos))
=>
   (assert (nodo (estado $?a H ?c ?d ?b $?e)
                 (camino $?movimientos ^))))

(defrule MAIN::abajo
   (nodo (estado $?a H ?b ?c ?d  $?e)
          (camino $?movimientos))
=>
   (assert (nodo (estado $?a ?d ?b ?c H $?e)
                 (camino $?movimientos v))))

(defrule MAIN::izquierda
   (nodo (estado $?a&:(neq (mod (length $?a) 3) 2)
                  ?b H $?c)
          (camino $?movimientos))
=>
   (assert (nodo (estado $?a H ?b $?c)
                 (camino $?movimientos <))))
```



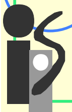
© José Angel Bañares

Problema del 8-puzzle

```
(defrule MAIN::derecha
   (nodo (estado $?a H ?b
                 $?c&:(neq (mod (length $?c) 3) 2))
         (camino $?movimientos))
 =>
   (assert (nodo (estado $?a ?b H $?c)
                 (camino $?movimientos >))))
```

- **Modulo de restricciones**

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

(defrule RESTRICCIONES::repeticiones-de-nodo
   (declare (auto-focus TRUE))
   (nodo (estado $?actual)
         (camino $?movimientos-1))
   ?nodo <- (nodo (estado $?actual)
                  (camino $?movimientos-1 ? $?))
 =>
   (retract ?nodo))
```

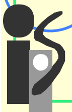


© José Angel Bañares

Problema del 8-puzzle

- **Modulo solución**

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

(defrule SOLUCION::reconoce-solucion
   (declare (auto-focus TRUE))
   ?nodo <- (nodo (estado 1 2 3 8 H 4 7 6 5)
                  (camino $?movimientos))
 =>
   (retract ?nodo)
   (assert (solucion $?movimientos)))

(defrule SOLUCION::escribe-solucion
   (solucion $?movimientos)
  =>
   (printout t "Solucion:" $?movimientos crlf)
   (halt))
```



© José Angel Bañares

Problema del 8-puzzle (con heurística)

- **Heurística**
  - Definición : número de piezas descolocadas
  - Heurística del estado inicial: 5

```
+---+---+---+                   +---+---+---+
| 2 | 8 | 3 |                   | 1 | 2 | 3 |
+---+---+---+                   +---+---+---+
| 1 | 6 | 4 |                   | 8 |   | 4 |
+---+---+---+                   +---+---+---+
| 7 |   | 5 |                   | 7 | 6 | 5 |
+---+---+---+                   +---+---+---+
Estado inicial                  Estado final
```

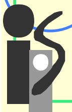
- **Modulo principal**

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

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

© José Angel Bañares

Problema del 8-puzzle (con heurística)

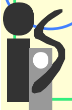
```
(defglobal MAIN
   ?*estado-inicial* = (create$ 2 8 3 1 6 4 7 H 5)
   ?*estado-final* = (create$ 1 2 3 8 H 4 7 6 5))

(deffunction MAIN::heuristica ($?estado)
   (bind ?res 0)
   (loop-for-count (?i 1 9)
    (if (neq (nth ?i $?estado)
             (nth ?i ?*estado-final*))
         then (bind ?res (+ ?res 1))
     )
    )
   ?res)

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



© José Angel Bañares

Problema del 8-puzzle (con heurística)

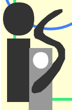
```
(defrule MAIN::arriba
   (nodo (estado $?a ?b ?c ?d H $?e)
         (camino $?movimientos)
         (clase cerrado))
=>
   (bind $?nuevo-estado (create$  $?a H ?c ?d ?b $?e))
   (assert (nodo
              (estado $?nuevo-estado)
              (camino $?movimientos ^)
              (heuristica (heuristica $?nuevo-estado)))))

(defrule MAIN::abajo
   (nodo (estado $?a H ?b ?c ?d  $?e)
         (camino $?movimientos)
         (clase cerrado))
=>
   (bind $?nuevo-estado (create$ $?a ?d ?b ?c H $?e))
   (assert (nodo
              (estado $?nuevo-estado)
              (camino $?movimientos v)
              (heuristica (heuristica $?nuevo-estado)))))
```



© José Angel Bañares

Problema del 8-puzzle (con heurística)

```
(defrule MAIN::izquierda
   (nodo (estado $?a&:(neq (mod (length $?a) 3) 2)
                     ?b H $?c)
          (camino $?movimientos)
          (clase cerrado))
=>
   (bind $?nuevo-estado (create$ $?a H ?b $?c))
   (assert (nodo
              (estado $?nuevo-estado)
              (camino $?movimientos <)
              (heuristica (heuristica $?nuevo-estado)))))

(defrule MAIN::derecha
   (nodo (estado $?a H ?b
                 $?c&:(neq (mod (length $?c) 3) 2))
          (camino $?movimientos)
          (clase cerrado))
 =>
   (bind $?nuevo-estado (create$ $?a ?b H $?c))
   (assert (nodo
              (estado $?nuevo-estado)
              (camino $?movimientos >)
              (heuristica (heuristica $?nuevo-estado)))))
```



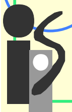
© José Angel Bañares

Problema del 8-puzzle (con heurística)

```
(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)))
```

- **Modulo de restricciones**

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

(defrule RESTRICCIONES::repeticiones-de-nodo
   (declare (auto-focus TRUE))
   (nodo (estado $?actual)
         (camino $?movimientos-1))
   ?nodo <- (nodo (estado $?actual)
                  (camino $?movimientos-1 ? $?))
 =>
   (retract ?nodo))
```

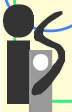


© José Angel Bañares

Problema del 8-puzzle (con heurística)

- **Modulo solución**

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

(defrule SOLUCION::reconoce-solucion
   (declare (auto-focus TRUE))
   ?nodo <- (nodo (heuristica 0)
                  (camino $?movimientos))
 =>
   (retract ?nodo)
   (assert (solucion $?movimientos)))

(defrule SOLUCION::escribe-solucion
   (solucion $?movimientos)
  =>
   (printout t "Solucion:" $?movimientos crlf)
   (halt))
```



© José Angel Bañares

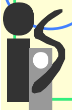
Problema del 8-puzzle (con heurística-2)

- **En la versión anterior el modulo main mezcla control (estrategia local/metareglas) con Operadores (resolución del problema)**

© José Angel Bañares

Problema del 8-puzzle (con heurística-2)

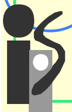
```
(defmodule MAIN (export deftemplate nodo)
                (export deffunction heuristica))

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

   (defglobal MAIN
   ?*estado-inicial* = (create$ 2 8 3 1 6 4 7 H 5)
   ?*estado-final* = (create$ 1 2 3 8 H 4 7 6 5))

(deffunction MAIN::heuristica ($?estado)
   ;;; idem
   ?res)
```



© José Angel Bañares

Problema del 8-puzzle (con heurística-2)

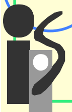
```
(defrule MAIN::inicial
   =>
  (assert (nodo  (estado ?*estado-inicial*)
    (camino) (heuristica (heuristica ?*estado-inicial*))
                )))

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



© José Angel Bañares

Problema del 8-puzzle (con heurística-2)

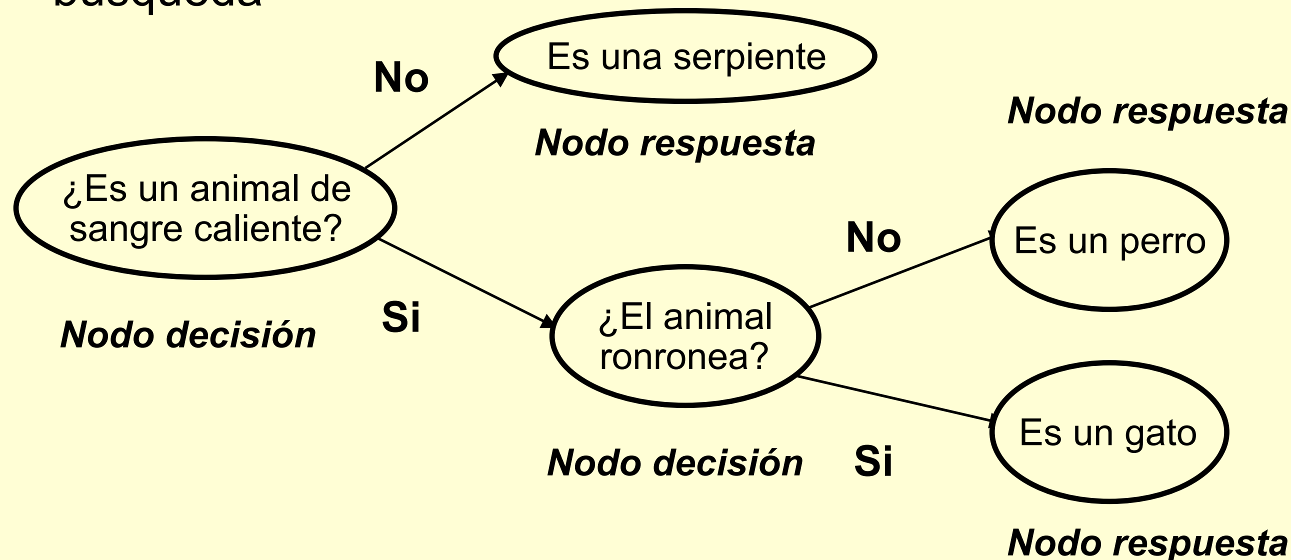
```
(defmodule OPERADORES
  (import MAIN deftemplate nodo)
  (import MAIN deffunction heuristica))

(defrule OPERADORES::arriba
   (nodo (estado $?a ?b ?c ?d H $?e)
         (camino $?movimientos)
         (clase cerrado))
=>
   (bind $?nuevo-estado (create$  $?a H ?c ?d ?b $?e))
   (assert (nodo
              (estado $?nuevo-estado)
              (camino $?movimientos ^)
              (heuristica (heuristica $?nuevo-estado)))))
…
```

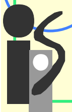


© José Angel Bañares

2. Árboles de decisión

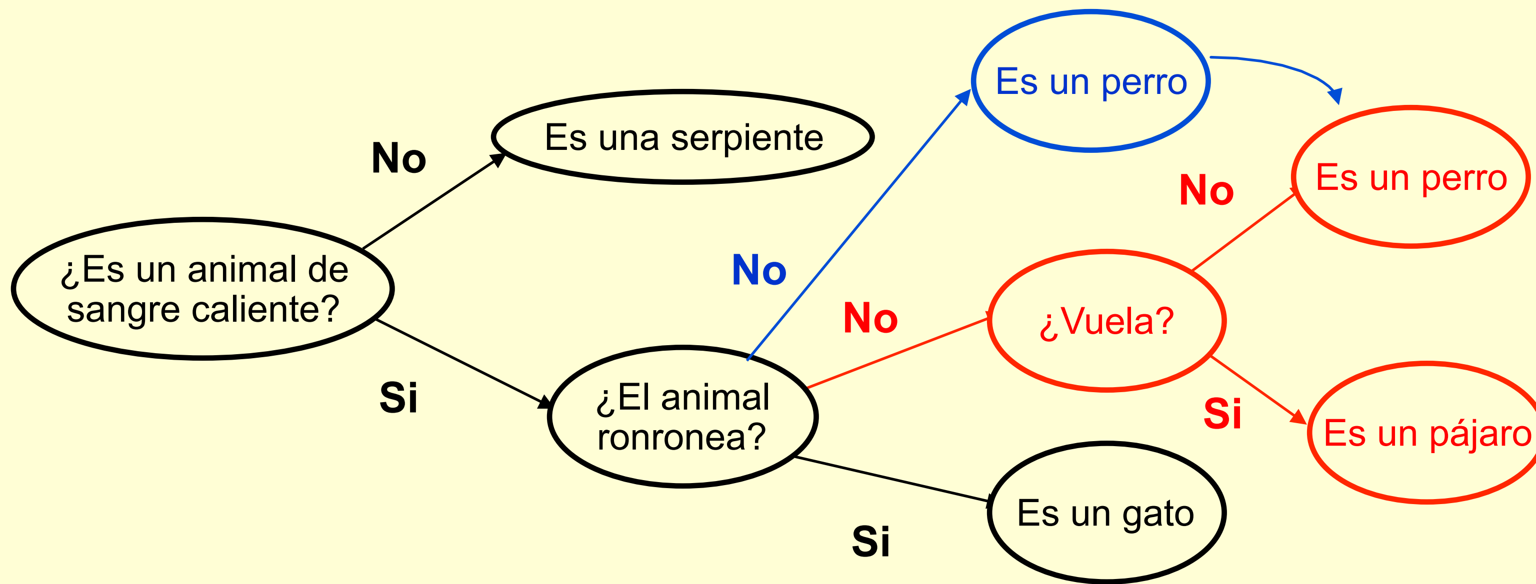
- **Paradigma útil para resolver problemas de clasificación**
  - Ofrecen la solución a un problema a partir de un conjunto de respuestas predeterminadas
    - Adecuado para clasificación, diagnóstico
    - No adecuado para planificación, o diseño
  - Reducen el conjunto de posibles soluciones mediante una serie de decisiones o cuestiones que podan el espacio de búsqueda

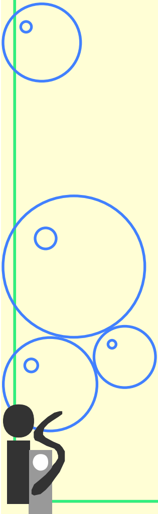




© José Angel Bañares

Árboles de decisión que aprenden

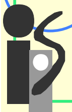




© José Angel Bañares

Árboles de decisión que aprenden

```
Algoritmo Resuelve-Árbol-y-Aprende
  Establece como nodo actual en el árbol el nodo raíz;
  MientrasQue  el nodo actual es  de decisión hacer
    Preguntar la pregunta del nodo actual;
    Si la respuesta es si
      entonces Establece como actual el nodo de la rama si
      sino Establece como actual el nodo de la rama no
    fsi;
  fmq;
  Preguntar si la respuesta en el nodo actual es correcta;
  Si la respuesta es correcta
   entonces Determina respuesta correcta
   sino Pide la respuesta correcta;
         Pide qué pregunta cuando se responda “si” distinguirá
        la respuesta del nodo actual de la repuesta correcta;
        Reemplaza el nodo respuesta por un nodo decisión que
        tiene como rama no el nodo respuesta actual y como
        rama si la respuesta correcta. La pregunta del nodo
         de decisión es la que distingue los dos nodos
        respuesta.
  fsi;
Fin.
```



© José Angel Bañares

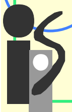
Árboles de decisión que aprenden

```
Algoritmo Resuelve-Árbol-y-Aprende
  Establece como nodo actual en el árbol el nodo raíz;
  MientrasQue  el nodo actual es de decisión hacer
    Preguntar la pregunta del nodo actual;
    Si la respuesta es si
      entonces Establece como actual el nodo de la rama si
      sino Establece como actual el nodo de la rama no
    fsi;
  fmq;
  Preguntar si la respuesta en el nodo actual es correcta;
  Si la respuesta es correcta
   entonces Determina respuesta correcta
   sino Pide la respuesta correcta;
        Pide qué pregunta cuando se responda “si” distinguirá
        la respuesta del nodo actual de la respuesta correcta;
        Reemplaza el nodo respuesta por un nodo decisión que
        tiene como rama no el nodo respuesta actual y como
        rama si la respuesta correcta. La pregunta del nodo
        de decisión es la que distingue los dos nodos
        respuesta.
  fsi;
Fin.
```

¡Algoritmo!

No usar Reglas

Sólo para ilustrar CLIPS



© José Angel Bañares

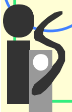
Árboles de decisión con reglas

- **Nodos del árbol de decisión**

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

- **Árbol inicial** Fichero texto animal.dat

```
(nodo (nombre raiz) (tipo decision)
      (pregunta "Es un animal de sangre caliente?")
      (nodo-si nodo1) (nodo-no nodo2))
(nodo (nombre nodo1) (tipo decision)
      (pregunta "El animal ronronea?")
      (nodo-si nodo3) (nodo-no nodo4))
(nodo (nombre nodo2) (tipo respuesta)
      (respuesta serpiente))
(nodo (nombre nodo3) (tipo respuesta) (respuesta gato))
(nodo (nombre nodo4) (tipo respuesta) (respuesta perro))
```



© José Angel Bañares

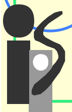
Árboles de decisión con reglas

- **Regla para inicializar árbol**

```
(defrule inicializa
   (not (nodo (nombre raiz)))
   =>
   (load-facts "animal.dat")
   (assert (nodo-actual raiz)))
```

- **Reglas que preguntan cuestión asociada a nodo decisión**

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

(defrule respuesta-incorrecta
   ?respuesta <- (respuesta ~si&~no)
   =>
   (retract ?respuesta))
```

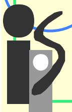


© José Angel Bañares

Árboles de decisión con reglas

- **Reglas actualizar nodo**

```
(defrule ir-a-rama-si
   ?nodo <- (nodo-actual ?nombre)
   (nodo (nombre ?nombre)
         (tipo decision)
         (nodo-si ?rama-si))
   ?respuesta <- (respuesta si)
   =>
   (retract ?nodo ?respuesta)
   (assert (nodo-actual ?rama-si)))

(defrule ir-a-rama-no
   ?nodo <- (nodo-actual ?nombre)
   (nodo (nombre ?nombre)
         (tipo decision)
         (nodo-no ?rama-no))
   ?respuesta <- (respuesta no)
   =>
   (retract ?nodo ?respuesta)
   (assert (nodo-actual ?rama-no)))
```



© José Angel Bañares

Árboles de decisión con reglas

- **Regla para preguntar si la respuesta es correcta**

```
(defrule pregunta-si-respuesta-nodo-es-correcta
   ?nodo <- (nodo-actual ?nombre)
   (nodo (nombre ?nombre)(tipo respuesta)(respuesta ?valor))
   (not (respuesta ?))
   =>
   (printout t "Creo que es un(a) " ?valor crlf)
   (printout t "He acertado? (si o no) ")
   (assert (respuesta (read))))
(defrule respuesta-nodo-adivinado-es-correcto
   ?nodo <- (nodo-actual ?nombre)
   (nodo (nombre ?nombre) (tipo respuesta))
   ?respuesta <- (respuesta si)
   =>
   (assert (pregunta-nuevo-intento))
   (retract ?nodo ?respuesta))
(defrule respuesta-nodo-adivinado-es-incorrecto
   ?nodo <- (nodo-actual ?nombre)
   (nodo (nombre ?nombre) (tipo respuesta))
   ?respuesta <- (respuesta no)
   =>
   (assert (cambia-respuesta-nodo ?nombre))
   (retract ?respuesta ?nodo))
```

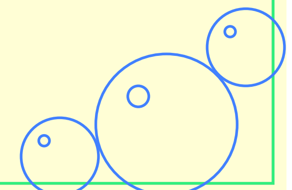
© José Angel Bañares

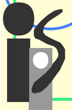
Árboles de decisión con reglas

- **Reglas para continuar o parar**

```
(defrule pregunta-nuevo-intento
   (pregunta-nuevo-intento)
   (not (respuesta ?))
   =>
   (printout t "Lo intento otra vez? (si o no) ")
   (assert (respuesta (read))))

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

(defrule no-mas
   ?fase <- (pregunta-nuevo-intento)
   ?respuesta <- (respuesta no)
   =>
   (retract ?fase ?respuesta)
   (save-facts "animal.dat" local nodo))
```

Hechos de las plantillas definidas en el módulo en curso



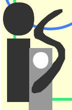


© José Angel Bañares

Árboles de decisión con reglas

- **Actualización del árbol**

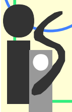
```
(defrule cambia-respuesta-nodo
   ?fase <- (cambia-respuesta-nodo ?nombre)
   ?data <- (nodo (nombre ?nombre) (tipo respuesta)
                  (respuesta ?valor))
   =>
   (retract ?fase) ; Determina que se deberia haber adivinado
   (printout t "Cual es el animal? ")
   (bind ?nuevo-animal (read)) ; Obtiene la pregunta
   (printout t "Que pregunta cuando la respuesta es si ")
   (printout t "distinguira " crlf "   un(a) ")
   (printout t ?nuevo-animal " de un " ?valor "? ")
   (bind ?pregunta (readline))
   (printout t "Ya puedo distinguir " ?nuevo-animal crlf)
   (bind ?nuevonodo1 (gensym*)) ; Crea los nodos nuevos
   (bind ?nuevonodo2 (gensym*))
   (modify ?data (tipo decision) (pregunta ?pregunta)
                 (nodo-si ?nuevonodo1) (nodo-no ?nuevonodo2))
   (assert (nodo (nombre ?nuevonodo1) (tipo respuesta)
                 (respuesta ?nuevo-animal)))
   (assert (nodo (nombre ?nuevonodo2) (tipo respuesta)
                 (respuesta ?valor)))
   (assert (pregunta-nuevo-intento))) ; Pregunta si continua
```



© José Angel Bañares

Traza árbol decisión

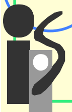
```
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (reset)
==> f-0      (initial-fact)
CLIPS> (agenda)
0       inicializa: f-0,
For a total of 1 activation.
CLIPS> (run 1)
FIRE    1 inicializa: f-0,
==> f-1 (nodo (nombre raiz) ...)
==> f-2 (nodo (nombre nodo1) ...)
==> f-3 (nodo (nombre nodo2) ...)
==> f-4 (nodo (nombre nodo3) ...)
==> f-5 (nodo (nombre nodo4) ...)
==> f-6 (nodo-actual raiz)
CLIPS> (agenda)
0       pide-decision-nodo-pregunta: f-6,f-1,
For a total of 1 activation.
```



© José Angel Bañares

Traza árbol decisión

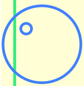
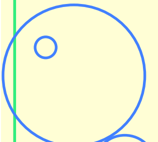
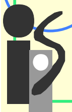
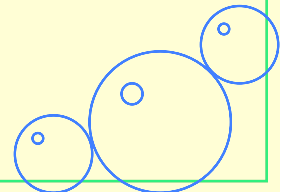
```
CLIPS> (run 2)
FIRE    1 pide-decision-nodo-pregunta: f-6,f-1,
Es un animal de sangre caliente? (si o no) si
==> f-7     (respuesta si)
FIRE    2 ir-a-rama-si: f-6,f-1,f-7
<== f-6     (nodo-actual raiz)
<== f-7     (respuesta si)
==> f-8     (nodo-actual nodo1)
CLIPS> (agenda)
0      pide-decision-nodo-pregunta: f-8,f-2,
For a total of 1 activation.
CLIPS> (run 2)
FIRE    1 pide-decision-nodo-pregunta: f-8,f-2,
El animal ronronea? (si o no) no
==> f-9     (respuesta no)
FIRE    2 ir-a-rama-no: f-8,f-2,f-9
<== f-8     (nodo-actual nodo1)
<== f-9     (respuesta no)
==> f-10    (nodo-actual nodo4)
CLIPS> (agenda)
0      pregunta-si-respuesta-nodo-es-correcta: f-10,f-5,
For a total of 1 activation.
```



© José Angel Bañares

Traza árbol decisión

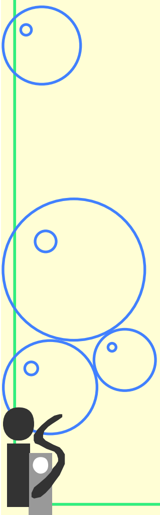
```
CLIPS> (run 2)
FIRE    1 pregunta-si-respuesta-nodo-es-correcta: f-10,f-5,
Creo que es un(a) perro
He acertado? (si o no) no
==> f-11    (respuesta no)
FIRE          2  respuesta-nodo-adivinado-is-incorrecto:
   f-10,f-5,f-11
==> f-12    (cambia-respuesta-nodo nodo4)
<== f-11    (respuesta no)
<== f-10    (nodo-actual nodo4)
CLIPS> (agenda)
0       cambia-respuesta-nodo: f-12,f-5
For a total of 1 activation.
```

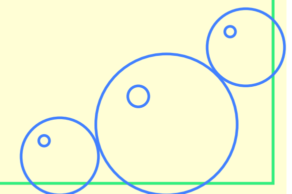





© José Angel Bañares

Traza árbol decisión

```
CLIPS> (run 1)
FIRE    1 cambia-respuesta-nodo: f-12,f-5
<== f-12    (cambia-respuesta-nodo nodo4)
Cual es el animal? pajaro
Que pregunta cuando la respuesta es si distinguira
   un(a) pajaro de un perro? Vuela?
Ya puedo distinguir pajaro
<== f-5 (nodo (nombre nodo4)(tipo respuesta)(respuesta perro))
==>  f-13     (nodo  (nombre  nodo4)(tipo  decision)(pregunta  "Vuela?")
                (nodo-si gen1) (nodo-no gen2) (respuesta perro))
==> f-14 (nodo (nombre gen1) (tipo respuesta)
               (respuesta pajaro))
==> f-15 (nodo (nombre gen2)(tipo respuesta)(respuesta perro))
==> f-16 (pregunta-nuevo-intento)
CLIPS> (agenda)
0      pregunta-nuevo-intento: f-16,
For a total of 1 activation.
CLIPS> (run 2)
FIRE    1 pregunta-nuevo-intento: f-16,
Lo intento otra ves? (si o no) no
==> f-17    (respuesta no)
FIRE    2 no-mas: f-16,f-17
<== f-16    (pregunta-nuevo-intento)
<== f-17    (respuesta no)
CLIPS> (agenda)
CLIPS>
```

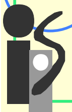




© José Angel Bañares

3.Emulación Backward Chaining

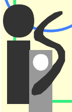
- **Emulación de un sistema con encadenamiento regresivo sencillo con CLIPS**
  - Si el encadenamiento regresivo es más apropiado será mejor un lenguaje que implemente éste directamente.
  - Limitaciones del sistema construido
    - Los hechos se representarán como pares atributo-valor
    - El encadenamiento se iniciará al incluir un objetivo en la MT
    - Sólo se comprueba la igualdad de un atributo a un valor en la LHS de una regla
    - Si el valor de un objetivo no se puede determinar por las reglas se preguntará al usuario. Los atributos no pueden tener valor desconocido.
    - Un atributo sólo puede tener un valor.



© José Angel Bañares

Backward Chaining

```
Algoritmo Resuelve-Objetivo (objetivo)
  objetivo: El objetivo actual a resolver;

  Si el valor del objetivo es conocido
      entonces devuelve el valor del objetivo
  fsi;
  Para cada regla cuyo consecuente contiene el objetivo hacer
    Llama Intenta-Regla con la regla;
    Si Intenta-Regla tiene éxito
      entonces Asigna al objetivo el valor del consecuente;
               Devuelve el valor del objetivo
    fsi;
  fpara;
  Pregunta al usuario por el valor del objetivo;
  Da al objetivo el valor suministrado por el usuario;
  Devuelve el valor del objetivo;
Fin.
```



© José Angel Bañares

Backward Chaining

```
Algoritmo Intenta-Regla (regla)
  Regla: Regla intentada para resolver el objetivo;

  Para cada condición en el antecedente de la regla hacer
    Llama Resuelve-Objetivo con la condición;
       Si el valor devuelto por Resuelve-Objetivo no es
        igual al valor requerido por la condición
      entonces Develve Sin-Éxito;
    fsi;
  fpara;
  Devuelve Éxito;
Fin.
```

© José Angel Bañares

Backward Chaining

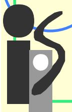
```
Algoritmo Intenta-Regla (regla)
  Regla: Regla intentada para resolver el objetivo;

  Para cada condición en el antecedente de la regla hacer
    Llama Resuelve-Objetivo con la condición;
       Si  el  valor devuelto por Resuelve-Objetivo no  es
        igual al valor requerido por la condición
      entonces Develve Sin-Éxito;
    fsi;
  fpara;
  Devuelve Éxito;
Fin.
```

¡Algoritmo!

=>

No usar Reglas

Sólo para ilustrar CLIPS

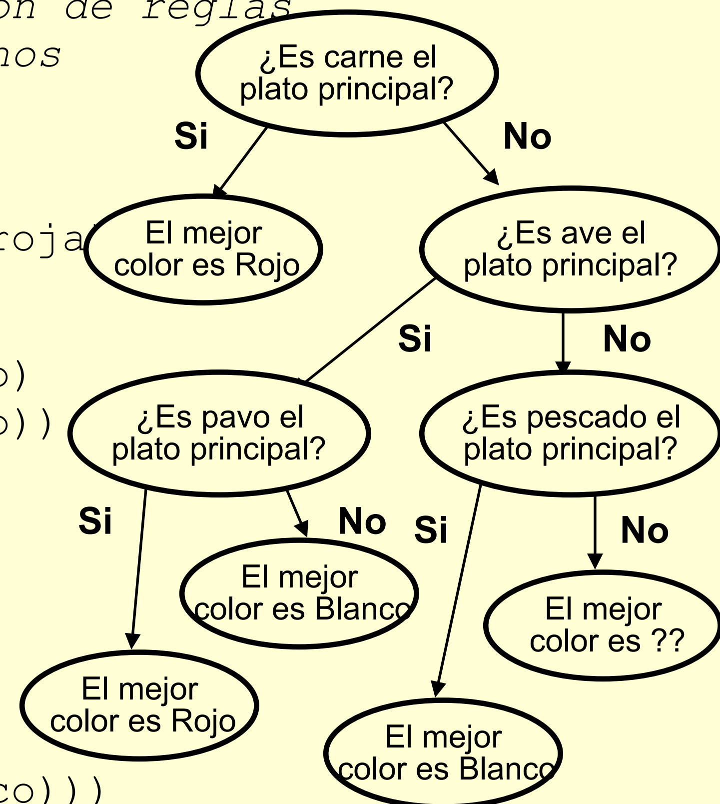


© José Angel Bañares

Backward Chaining

- **Representación del conocimiento**

```
;;; Representación de las reglas
(deftemplate BC::regla    ; Representación de reglas
    (multislot si)        ; mediante hechos
    (multislot entonces))

(deffacts MAIN::reglas-vino
    (regla (si plato-principal es carne-roja)
           (entonces mejor-color es rojo))

    (regla (si plato-principal es pescado)
           (entonces mejor-color es blanco))

    (regla (si plato-principal es ave y
               carne-es-pavo es si)
           (entonces mejor-color es rojo))

    (regla (si plato-principal es ave y
               carne-es-pavo es no)
            (entonces mejor-color es blanco)))
```



© José Angel Bañares

Backward Chaining

- La representación de reglas como hechos permite su manipulación

Si el plato principal es ave **podemos eliminar las reglas**

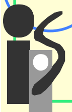
```
(regla (si plato-principal es carne-roja)
       (entonces mejor-color es rojo)) y
(regla (si plato-principal es pescado)
       (entonces mejor-color es blanco))
```

y **modificar las reglas**

```
(regla (si plato-principal es ave y
       carne-es-pavo es si)
      (entonces mejor-color es rojo))
(regla (si plato-principal es ave y
       carne-es-pavo es no)
      (entonces mejor-color es blanco)))
```

quedando las reglas

```
(regla (si carne-es-pavo es si)
      (entonces mejor-color es rojo))
(regla (si carne-es-pavo es no)
      (entonces mejor-color es blanco))
```



© José Angel Bañares

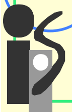
Backward Chaining

- **Representación del conocimiento**

```
;;; Representación de los objetivos
(deftemplate BC::objetivo
   (slot atributo))

(deffacts MAIN::objetivo-inicial
   (objetivo (atributo mejor-color)))

;;; Representación de atributos cuyo valor se ha obtenido
(deftemplate BC::atributo
   (slot nombre)
   (slot valor))
```

- **Módulos**

```
(defmodule BC
   (export deftemplate regla objetivo atributo))

(defmodule MAIN (import BC deftemplate regla objetivo))
```

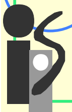


© José Angel Bañares

Backward Chaining

- **1. Reglas que generan subobjetivos y piden valores**

```
(defrule BC::intenta-regla
   (objetivo (atributo ?nombre-objetivo))
   (regla (si ?un-nombre $?)
          (entonces ?nombre-objetivo $?))
   (not (atributo (nombre ?un-nombre)))
   (not (objetivo (atributo ?un-nombre)))
   =>
   (assert (objetivo (atributo ?un-nombre))))

(defrule BC::pide-valor-atributo
   ?objetivo <- (objetivo (atributo ?nombre-objetivo))
   (not (atributo (nombre ?nombre-objetivo)))
   (not (regla (entonces ?nombre-objetivo $?)))
   =>
   (retract ?objetivo)
   (printout t "Cual es el valor de " ?nombre-objetivo "? ")
   (assert (atributo (nombre ?nombre-objetivo)
                     (valor (read)))))
```

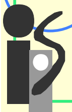


© José Angel Bañares

Backward Chaining

- **2. Reglas que actualizan reglas y objetivos**

```
(defrule BC::objetivo-satisfecho
   (declare (salience 100))
   ?objetivo <- (objetivo (atributo ?nombre-objetivo))
   (atributo (nombre ?nombre-objetivo))
   =>
   (retract ?objetivo))

(defrule BC::regla-satisfecha
   (declare (salience 100))
   (objetivo (atributo ?nombre-objetivo))
   (atributo (nombre ?un-nombre)
             (valor ?un-valor))
   ?regla <- (regla (si ?un-nombre es ?un-valor)
          (entonces ?nombre-objetivo es ?valor-objetivo))
   =>
   (retract ?regla)
   (assert (atributo (nombre ?nombre-objetivo)
                     (valor ?valor-objetivo))))
```

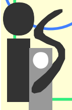


© José Angel Bañares

Backward Chaining

- **Reglas que actualizan valores (continua)**

```
(defrule BC::elimina-regla-no-reconocida
   (declare (salience 100))
   (objetivo (atributo ?nombre-objetivo))
   (atributo (nombre ?un-nombre) (valor ?un-valor))
   ?regla <- (regla (si ?un-nombre es ~?un-valor)
         (entonces ?nombre-objetivo es ?valor-objetivo))
   =>
   (retract ?regla))

(defrule BC::modifica-regla-reconocida
   (declare (salience 100))
   (objetivo (atributo ?nombre-objetivo))
   (atributo (nombre ?un-nombre) (valor ?un-valor))
   ?regla <- (regla (si ?un-nombre es ?un-valor y
                        $?resto-si)
          (entonces ?nombre-objetivo es ?valor-objetivo))
   =>
   (retract ?regla)
   (modify ?regla (si $?resto-si)))
```



© José Angel Bañares

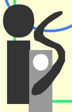
Backward Chaining

- **Comienzo del proceso**

```
(defrule MAIN::comienza-BC
   =>
   (focus BC))
```

- **Traza. Initicialización**

```
CLIPS> (reset)
CLIPS> (facts)
f-0      (initial-fact)
f-1      (objetivo (atributo mejor-color))
f-2      (regla (si plato-principal es carne-roja)
                (entonces mejor-color es rojo))
f-3      (regla (si plato-principal es pescado)
                (entonces mejor-color es blanco))
f-4      (regla (si plato-principal es ave y carne-es-pavo
          es si) (entonces mejor-color es rojo))
f-5      (regla (si plato-principal es ave y carne-es-pavo
          es no) (entonces mejor-color es blanco))
For a total of 6 facts.
CLIPS> (agenda)
0       comienza-BC: f-0
For a total of 1 activation.
```



© José Angel Bañares

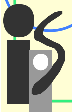
Backward Chaining

- **Reglas que pueden determinar el valor del** `mejor-color`

```
CLIPS> (agenda)
0      intenta-regla: f-1,f-5,,
0      intenta-regla: f-1,f-4,,
0      intenta-regla: f-1,f-3,,
0      intenta-regla: f-1,f-2,,
For a total of 4 activations.
```

- **Intenta regla**

```
; f5 SI plato-principal es ave y
;       carne-es-pavo es no
;    ENTONCES el mejor-color es blanco

CLIPS> (watch rules)
CLIPS> (watch facts)
CLIPS> (run 1)
FIRE    1 intenta-regla: f-1,f-5,,
==> f-6     (objetivo (atributo plato-principal))
CLIPS> (agenda)
0      pide-valor-atributo: f-6,,
For a total of 1 activation.
```

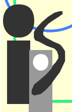


© José Angel Bañares

Backward Chaining

- **Ejecución de la regla pide-valor-atributo**

```
CLIPS> (run 1)
FIRE    1 pide-valor-atributo: f-6,,
<== f-6 (objetivo (atributo plato-principal))
Cual es el valor de plato-principal? ave
==> f-7 (atributo (nombre plato-principal) (valor ave))
CLIPS> (agenda)
100    modifica-regla-reconocida: f-1,f-7,f-5
100    modifica-regla-reconocida: f-1,f-7,f-4
100    elimina-regla-no-reconocida: f-1,f-7,f-3
100    elimina-regla-no-reconocida: f-1,f-7,f-2
For a total of 4 activations.
```

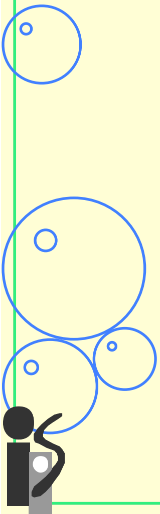


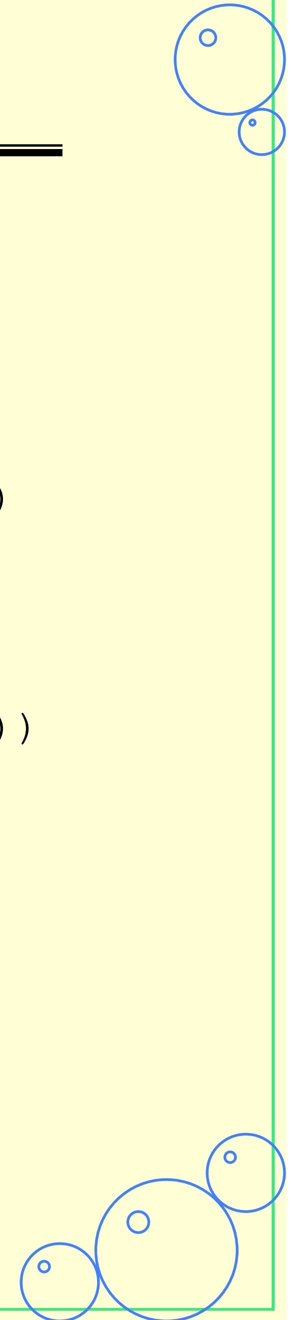
© José Angel Bañares

Backward Chaining

- **Modifica reglas reconocidas**

```
CLIPS> (run 2)
FIRE    1 modifica-regla-reconocida: f-1,f-7,f-5
<== f-5 (regla (si plato-principal es ave y
  carne-es-pavo es no) (entonces mejor-color es blanco))
==> f-8 (regla (si carne-es-pavo es no)
               (entonces mejor-color es blanco))
FIRE    2 modifica-regla-reconocida: f-1,f-7,f-4
<== f-4 (regla (si plato-principal es ave y
     carne-es-pavo es si) (entonces mejor-color es rojo))
==> f-9 (regla (si carne-es-pavo es si)
        (entonces mejor-color es rojo))
CLIPS> (agenda)
100    elimina-regla-no-reconocida: f-1,f-7,f-3
100    elimina-regla-no-reconocida: f-1,f-7,f-2
0      intenta-regla: f-1,f-9,,
0      intenta-regla: f-1,f-8,,
For a total of 4 activations.
```



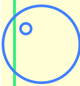
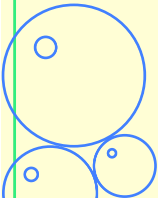
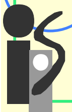
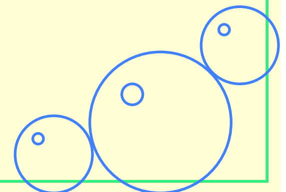


© José Angel Bañares

Backward Chaining

- **Elimina reglas no reconocidas**

```
CLIPS> (run 2)
FIRE    1 elimina-regla-no-reconocida: f-1,f-7,f-3
<== f-3  (regla (si plato-principal es pescado)
                (entonces mejor-color es blanco))
FIRE    2 elimina-regla-no-reconocida: f-1,f-7,f-2
<== f-2 (regla (si plato-principal es carne-roja)
               (entonces mejor-color es rojo))
CLIPS> (agenda)
0      intenta-regla: f-1,f-9,,
0      intenta-regla: f-1,f-8,,
For a total of 2 activations.
```

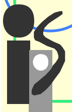





© José Angel Bañares

Backward Chaining

- **Intenta reglas modificadas**

```
CLIPS> (agenda)
0      intenta-regla: f-1,f-9,,
0      intenta-regla: f-1,f-8,,
For a total of 2 activations.
CLIPS> (run 1)
FIRE    1 intenta-regla: f-1,f-9,,
==> f-10 (objetivo (atributo carne-es-pavo))
CLIPS> (agenda)
0      pide-valor-atributo: f-10,,
For a total of 1 activation.
CLIPS> (run 1)
FIRE    1 pide-valor-atributo: f-10,,
<== f-10 (objetivo (atributo carne-es-pavo))
Cual es el valor de carne-es-pavo? si
==> f-11 (atributo (nombre carne-es-pavo) (valor si))
CLIPS> (agenda)
100    regla-satisfecha: f-1,f-11,f-9
100    elimina-regla-no-reconocida: f-1,f-11,f-8
For a total of 2 activations.
```

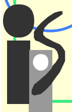


© José Angel Bañares

Backward Chaining

- **Fin**

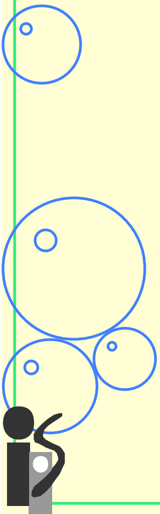
```
CLIPS> (run)
FIRE    1 regla-satisfecha: f-1,f-11,f-9
<== f-9 (regla (si carne-es-pavo es si)
               (entonces mejor-color es rojo))
==> f-12 (atributo (nombre mejor-color) (valor rojo))
FIRE    2 objetivo-satisfecho: f-1,f-12
<== f-1     (objetivo (atributo mejor-color))
CLIPS> (agenda)
CLIPS> (facts *)
f-0     (initial-fact)
f-7     (atributo (nombre plato-principal) (valor ave))
f-8     (regla (si carne-es-pavo es no)
               (entonces mejor-color es blanco))
f-11    (atributo (nombre carne-es-pavo) (valor si))
f-12    (atributo (nombre mejor-color) (valor rojo))
For a total of 5 facts.
```

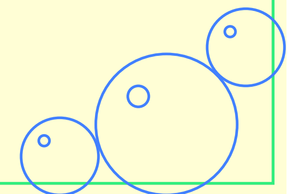


© José Angel Bañares

4. Problema de Monitorización

- **Definición del problema**
  - Una planta de procesamiento contiene varios dispositivos que deben ser monitorizados.
  - Cada dispositivo tiene uno o más sensores que ofrecen valores numéricos que indican la salud del dispositivo
  - El programa de monitorización debe
    - 1. Leer los valores de los sensores
    - 2. Evaluar las lecturas,
    - 3. Emitir alarmas o parar dispositivos

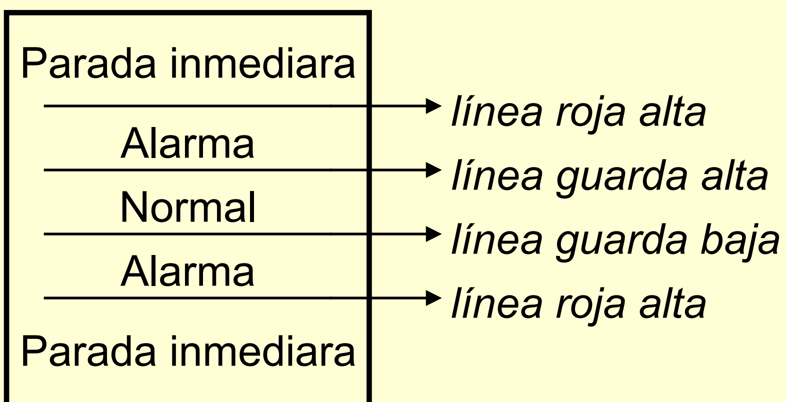




© José Angel Bañares

Acciones a realizar por el monitor

Valor del Sensor	***Acción***
valor <= línea roja baja	PARADA dispositivo
línea roja baja < valor <= línea guarda baja	Da ALARMA o PARADA dispositivo
línea guarda baja > valor < línea guarda alta	Ninguna
línea guarda alta >= valor < línea roja alta	Da ALARMA o PARADA dispositivo
valor >= línea guarda alta	PARADA dispositivo

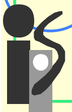



```
Ciclo 20 - Sensor 4 en
   línea guarda alta
Ciclo 25 - Sensor 4 en
   linea roja alta
    PARADA dispositivo 4
Ciclo 32 - Sensor 3 en
   linea guarda baja
Ciclo 38 - Sensor 1 en
   linea  guarda  alta
   durante 6 ciclos
   PARADA dispositivo 1
```

© José Angel Bañares

Proceso de desarrollo del programa

- **Proceso iterativo:**
  - Se parte de la descripción general del problema y se van incorporando detalles específicos
- **Técnica**
  - Gracias a las técnicas de desarrollo iterativas que soportan los SEs, es posible construir prototipos a partir de problemas con especificaciones pobres.
    - Se realiza un prototipo
    - Se apuntan los detalles que faltan en la especificación
    - Se consulta con el experto los detalles y se elabora otro prototipo



© José Angel Bañares

Asunciones iniciales

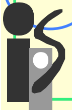
- **Detalles necesarios para empezar**
  - Decisiones sobre la implementación:
    - Reglas generales que permitan incorporar nuevos dispositivos y sensores fácilmente, en lugar de reglas específicas.
  - Detalles sobre el flujo de control
    - 3 Fases por ciclo: Lectura sensores, Análisis valores, acciones
  - Lectura sensores:
    - Posibilidades:

      -¿Lectura directa del sensor?

      - ¿Simulación de los valores de los sensores?

      - ¿El valor está disponible cuando se requiere?

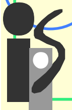
      - ¿Puede dar una lectura incorrecta el sensor?
- **Durante el desarrollo se debe mantener**
  - Una lista de asunciones
  - Lista de preguntas
  - Posibles inconsistencias relacionadas con la especificación



© José Angel Bañares

Lista de asunciones del problema

- Lista inicial de asunciones del problema de monitorización
  - Los datos de los sensores siempre están disponibles y son fiables
  - Los valores de los sensores se podrán leer directamente de éstos. Se debería soportar el uso de valores de sensores simulados.
  - Los valores de sensores de dispositivos parados no se monitorizarán.
  - El programa de monitorización realiza las acciones especificadas: ALARMA o PARADA de dispositivos.
  - El problema de divide en tres fases:
    - Lectura de los valores de los sensores
    - Análisis de los valores
    - Acciones
- También se debe mantener una lista de detalles sobre decisiones de implementación
  - Como representar la información, el control y la depuración del programa

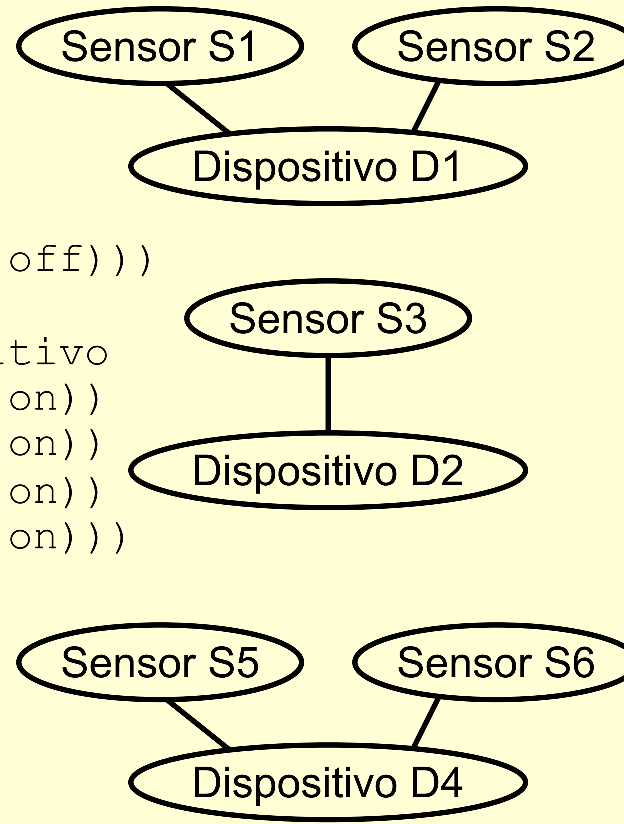


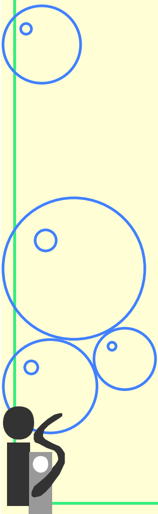
© José Angel Bañares

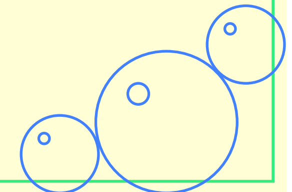
Implementación - > Representación

- **Representación dispositivos**

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

(deftemplate MAIN::dispositivo
   (slot nombre (type SYMBOL))
   (slot estado (allowed-values on off)))

(deffacts MAIN::informacion-dispositivo
  (dispositivo (nombre D1) (estado on))
  (dispositivo (nombre D2) (estado on))
  (dispositivo (nombre D3) (estado on))
  (dispositivo (nombre D4) (estado on)))
```





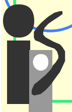


© José Angel Bañares

Representación

- **Representación sensores**

```
(deftemplate MAIN::sensor
   (slot nombre (type SYMBOL))
   (slot dispositivo (type SYMBOL))
   (slot valor-numerico (type SYMBOL NUMBER)
                        (allowed-symbols none)
                        (default none))
   (slot estado (allowed-values bajo-linea-roja
                                bajo-linea-guarda
                                normal
                                sobre-linea-roja
                                sobre-linea-guarda)
                (default normal))
   (slot bajo-linea-roja (type NUMBER))
   (slot bajo-linea-guarda (type NUMBER))
   (slot sobre-linea-guarda (type NUMBER))
   (slot sobre-linea-roja (type NUMBER)))
```



© José Angel Bañares

Representación

- **Atributos de los sensores**

```
(deffacts MAIN::sensor-information
  (sensor (nombre S1) (dispositivo D1)
          (bajo-linea-roja 60) (bajo-linea-guarda 70)
          (sobre-linea-guarda 120) (sobre-linea-roja 130))
  (sensor (nombre S2) (dispositivo D1)
          (bajo-linea-roja 20) (bajo-linea-guarda 40)
          (sobre-linea-guarda 160) (sobre-linea-roja 180))
  (sensor (nombre S3) (dispositivo D2)
          (bajo-linea-roja 60) (bajo-linea-guarda 70)
          (sobre-linea-guarda 120) (sobre-linea-roja 130))
  (sensor (nombre S4) (dispositivo D3)
          (bajo-linea-roja 60) (bajo-linea-guarda 70)
          (sobre-linea-guarda 120) (sobre-linea-roja 130))
  (sensor (nombre S5) (dispositivo D4)
          (bajo-linea-roja 65) (bajo-linea-guarda 70)
          (sobre-linea-guarda 120) (sobre-linea-roja 125))
  (sensor (nombre S6) (dispositivo D4)
          (bajo-linea-roja 110) (bajo-linea-guarda 115)
          (sobre-linea-guarda 125) (sobre-linea-roja 130))))
```

© José Angel Bañares

Inicio y control ejecución

- **Inicio-ciclo**

```
;;; Hecho ordenado (ciclo <numero>) indica ciclo
;;; Hecho ordenado (fuente-datos <fuentes> indica de
;;; de donde provienen los datos

(deffacts MAIN::comienza-ciclo
  ;(fuente-datos usuario)
   (fuente-datos hechos)
  (ciclo 0))
```

- **Control de ejecución**

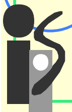
```
(defrule MAIN::comienza-siguiente-ciclo
   ?f <- (ciclo ?ciclo-actual)
   =>
   (retract ?f)
   (assert (ciclo (+ ?ciclo-actual 1)))
   (focus ENTRADA TENDENCIAS ALARMAS))
```

© José Angel Bañares

ENTRADA de valores

- **Lectura directa del sensor**
  - Se supone definida la función externa `lee-valor-sensor`
  - Para utilizar funciones externas implementadas en otros lenguajes es necesario recompilar el código CLIPS

```
(defmodule ENTRADA (import MAIN ?ALL))

(defrule ENTRADA::Lee-valor-sensores-De-Sensores
   (fuente-datos sensores)
   ?s <- (sensor (nombre ?nombre)
                 (valor-numerico none)
                 (dispositivo ?dispositivo))
   (dispositivo (nombre ?dispositivo) (estado on))
   =>
   ;(modify ?s (valor-numerico
                (lee-valor-sensor ?nombre))))
```

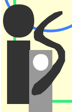


© José Angel Bañares

ENTRADA de valores

- **Pregunta al usuario (Durante etapa de prototipado)**

```
(defrule ENTRADA::Lee-Valor-Sensor-De-Usuario
  (fuente-datos usuario)
  ?s <- (sensor (nombre ?nombre)
                (valor-numerico none)
                (dispositivo ?dispositivo))
  (dispositivo (nombre ?dispositivo) (estado on))
  =>
  (printout t "Introduce el valor del sensor: "
                ?nombre ": ")
  (bind ?valor-numerico (read))
  (if (not (numberp ?valor-numerico))
      then (halt)
      else (modify ?s (valor-numerico ?valor-numerico))))
```



© José Angel Bañares

Entrada de valores

- **Lectura de un guión (Durante etapa de prototipado)**

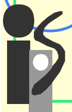
```
(deftemplate ENTRADA::Hecho-para-datos-sensor
    (slot nombre)
    (multislot datos))

(deffacts ENTRADA::sensor-fact-datos-values
   (Hecho-para-datos-sensor (nombre S1)
      (datos 100 100 110 110 115 120))
   (Hecho-para-datos-sensor (nombre S2)
      (datos 110 120 125 130 130 135))
   (Hecho-para-datos-sensor (nombre S3)
      (datos 100 120 125 130 130 125))
   (Hecho-para-datos-sensor (nombre S4)
      (datos 120 120 120 125 130 135))
   (Hecho-para-datos-sensor (nombre S5)
      (datos 110 120 125 130 135 135))
   (Hecho-para-datos-sensor (nombre S6)
      (datos 115 120 125 135 130 135)))
```

© José Angel Bañares

Entrada de valores

- **Lectura de un guión (Durante etapa de prototipado)**

```
(defrule ENTRADA::Lee-valores-sensor-de-Hechos
  (fuente-datos hechos)
  ?s <- (sensor (nombre ?nombre)(valor-numerico none))
  ?f <- (Hecho-para-datos-sensor (nombre ?nombre)
                   (datos ?valor-numerico $?resto))
  =>
  (modify ?s (valor-numerico ?valor-numerico))
  (modify ?f (datos ?resto)))

(defrule ENTRADA::No-quedan-valores-sensor-en-Hechos
  (fuente-datos hechos)
  (sensor (nombre ?nombre)(valor-numerico none))
  (Hecho-para-datos-sensor (nombre ?nombre) (datos))
  =>
   (printout t "Ningun hecho contiene datos para el sensor "
              ?nombre crlf)
  (printout t "DETENCION del sistema de monitorizacion."
              crlf)
  (halt))
```

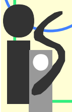


© José Angel Bañares

Entrada de valores

- **Lectura de un fichero (Durante etapa de prototipado)**
  - Asumiremos que los valores de los sensores pueden dejarse sin especificar indicando que el valor del sensor no ha cambiado.

```
(defrule ENTRADA::Abre-fichero-con-valores-sensores
  (fuente-datos fichero)
  (not (fichero-datos-abierto))
  =>
  (bind ?flag file-closed)
  (while (eq ?flag file-closed)
    (printout t "Cual es el nombre del fichero de datos: ")
    (bind ?nombre-fichero (readline))
    (if (open ?nombre-fichero fichero-datos "r")
       then (bind ?flag true)))
  (assert (fichero-datos-abierto)))
```



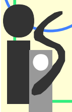
© José Angel Bañares

Entrada de valores

- **Lectura de un fichero (Durante etapa de prototipado)**

```
(defrule ENTRADA::Lee-valores-sensores-de-fichero
  (fuente-datos fichero)
  (fichero-datos-abierto)
  (ciclo ?tiempo)
  =>
  (bind ?nombre (read fichero-datos))
  (if (eq ?nombre EOF) then (halt))
  (while (and (neq ?nombre fin-de-ciclo)
              (neq ?nombre EOF))
     (bind ?valor-numerico (read fichero-datos))
     (if (eq ?valor-numerico EOF) then (halt))
     (assert (valor-numerico-sensor ?nombre
                                    ?valor-numerico))
     (bind ?nombre (read fichero-datos))
     (if (eq ?nombre EOF) then (halt))))
```

S1 100
S2 110
S3 100
S4 120
S5 110
S6 115
fin-de-ciclo
S2 120
S3 120
S5 120
S6 120
fin-de-ciclo
S1 110
S2 125
S3 125
S4 120
S5 125
S6 125
fin-de-ciclo
...

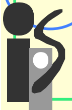


© José Angel Bañares

Entrada de valores

- **Lectura de un fichero**

```
(defrule ENTRADA::Elimina-Valores-de-Sensores-Inactivos
   (fuente-datos dichero)
   (fichero-datos-abierto)
   (ciclo ?tiempo)
   (sensor (nombre ?nombre) (dispositivo ?dispositivo))
   (dispositivo (nombre ?dispositivo) (estado off))
   ?datos <- (valor-numerico-sensor ?nombre ?valor-numerico)
   =>
   (retract ?datos))

; Sólo realiza acciones sobre sensores que han cambiado
(defrule ENTRADA::Transfiere-valores-numericos-a-Sensores
   (fuente-datos fichero)
   ?s <- (sensor (nombre ?nombre)
                 (valor-numerico none)
                 (dispositivo ?dispositivo))
   (dispositivo (nombre ?dispositivo) (estado on))
   ?f <- (valor-numerico-sensor ?nombre ?valor-numerico)
   =>
   (modify ?s (valor-numerico ?valor-numerico))
   (retract ?f))
```



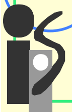
© José Angel Bañares

Detección de la tendencia

- **Fase detección de la tendencia**
  - 1. Se determina el estado (normal, bajo o sobre línea guarda o línea roja)

```
(defmodule TENDENCIAS (import MAIN ?ALL))

(defrule TENDENCIAS::estado-Normal
  ?s <- (sensor (valor-numerico ?valor-numerico&~none)
                (bajo-linea-guarda ?blg)
                (sobre-linea-guarda ?slg))
  (test (and (> ?valor-numerico ?blg)
             (< ?valor-numerico ?slg)))
  =>
  (modify ?s (estado normal) (valor-numerico none)))

(defrule TENDENCIAS::estado-sobre-linea-guarda
  ?s <- (sensor (valor-numerico ?valor-numerico&~none)
                (sobre-linea-guarda ?slg)
                (sobre-linea-roja ?slr))
  (test (and (>= ?valor-numerico ?slg)
             (< ?valor-numerico ?slr)))
  =>
  (modify ?s (estado sobre-linea-guarda)
             (valor-numerico none)))
```

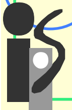
El valor numérico será leido en el siguiente ciclo



© José Angel Bañares

Detección de la tendencia

```
(defrule TENDENCIAS::estado-sobre-linea-roja
  ?s <- (sensor (valor-numerico ?valor-numerico&~none)
                (sobre-linea-roja ?slr))
  (test (>= ?valor-numerico ?slr))
  =>
  (modify ?s (estado sobre-linea-roja)(valor-numerico none)))

(defrule TENDENCIAS::estado-bajo-linea-guarda
  ?s <- (sensor (valor-numerico ?valor-numerico&~none)
                (bajo-linea-guarda ?blg)
                (bajo-linea-roja ?blr))
  (test (and (> ?valor-numerico ?blr)
             (<= ?valor-numerico ?blg)))
  =>
  (modify ?s (estado bajo-linea-guarda)
             (valor-numerico none)))

(defrule TENDENCIAS::estado-bajo-linea-roja
  ?s <- (sensor (valor-numerico ?valor-numerico&~none)
                (bajo-linea-roja ?blr))
  (test (<= ?valor-numerico ?blr))
  =>
  (modify ?s (estado bajo-linea-roja) (valor-numerico none)))
```



© José Angel Bañares

Detección de la tendencia

- **Fase detección de la tendencia**
  - 2. Se mantiene información del valor pasado del sensor y se monitoriza la tendencia

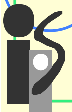
```
(deftemplate MAIN::tendencia-sensor
    (slot nombre)
    (slot estado (default normal))
    (slot principio (default 0))
    (slot fin (default 0))
    (slot alarma-antes-parada (default 3)))

(deffacts MAIN::Inicia-Tendencias
   (tendencia-sensor (nombre S1) (alarma-antes-parada 3))
   (tendencia-sensor (nombre S2) (alarma-antes-parada 5))
   (tendencia-sensor (nombre S3) (alarma-antes-parada 4))
   (tendencia-sensor (nombre S4) (alarma-antes-parada 4))
   (tendencia-sensor (nombre S5) (alarma-antes-parada 4))
   (tendencia-sensor (nombre S6) (alarma-antes-parada 2)))
```

© José Angel Bañares

Detección de la tendencia

- **Fase detección de la tendencia**

```
;;; Monitorización de la tendencia
(defrule TENDENCIAS::estado-No-Ha-Cambiado
  (ciclo ?tiempo)
  ?tendencia <- (tendencia-sensor (nombre ?sensor)
                                  (estado ?estado)
                                  (fin ?fin-ciclo&~?tiempo))
  (sensor (nombre ?sensor) (estado ?estado)
          (valor-numerico none))
  =>
  (modify ?tendencia (fin ?tiempo)))

(defrule TENDENCIAS::estado-Ha-Cambiado
  (ciclo ?tiempo)
  ?tendencia <- (tendencia-sensor (nombre ?sensor)
                           (estado ?estado)
                           (fin ?fin-ciclo&~?tiempo))
  (sensor (nombre ?sensor)(estado ?nuevo-estado&~?estado)
          (valor-numerico none))
  =>
  (modify ?tendencia (principio ?tiempo) (fin ?tiempo)
               (estado ?nuevo-estado)))
```

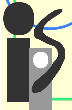


© José Angel Bañares

Alarmas

- **Fase de alamas:**
  - Sensores en la zona roja => PARADA Inmediata
  - Sensores en la zona de guarda durante un tiempo => PARADA Inmediata
  - Sensores en la zona de guarda => ALARMA

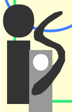
```
(defmodule ALARMAS (import MAIN ?ALL))

(defrule ALARMAS::Parada-en-Region-Roja
  (ciclo ?tiempo)
  (tendencia-sensor
     (nombre ?sensor)
     (estado ?estado&sobre-linea-roja | bajo-linea-roja))
  (sensor (nombre ?sensor) (dispositivo ?dispositivo))
  ?on <- (dispositivo (nombre ?dispositivo) (estado on))
  =>
  (printout t "ciclo " ?tiempo " - ")
  (printout t "Sensor " ?sensor " en " ?estado crlf)
  (printout t "   PARADA dispositivo " ?dispositivo
              crlf)
  (modify ?on (estado off)))
```



© José Angel Bañares

Alarmas

```
(defrule ALARMAS::Parada-en-Region-Alarma
  (ciclo ?tiempo)
  (tendencia-sensor
       (nombre ?sensor)
       (estado ?estado&sobre-linea-guarda |bajo-linea-guarda)
       (alarma-antes-parada ?duracion)
       (principio ?principio) (fin ?fin))
  (test (>= (+ (- ?fin ?principio) 1)  ?duracion))
  (sensor (nombre ?sensor) (dispositivo ?dispositivo))
  ?on <- (dispositivo (nombre ?dispositivo) (estado on))
  =>
  (printout t "ciclo " ?tiempo " - ")
  (printout t "Sensor " ?sensor " en " ?estado " ")
  (printout t "durante " ?duracion " ciclos " crlf)
  (printout t "  PARADA dispositivo " ?dispositivo crlf)
  (modify ?on (estado off)))
```



© José Angel Bañares

Alarmas

```
(defrule ALARMAS::Sensor-En-Region-Guarda
  (ciclo ?tiempo)
  (tendencia-sensor
       (nombre ?sensor)
       (estado ?estado&sobre-linea-guarda |bajo-linea-guarda)
       (alarma-antes-parada ?duracion)
       (principio ?principio) (fin ?fin))
  (test (< (+ (- ?fin ?principio) 1) ?duracion))
  =>
  (printout t "ciclo " ?tiempo " - ")
  (printout t "Sensor " ?sensor " en " ?estado crlf))
```