Tema 12: Aplicaciones de SBC: Decisión y metaintérpretes

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

Dpto. de Ciencias de la Computación e Inteligencia Artificial

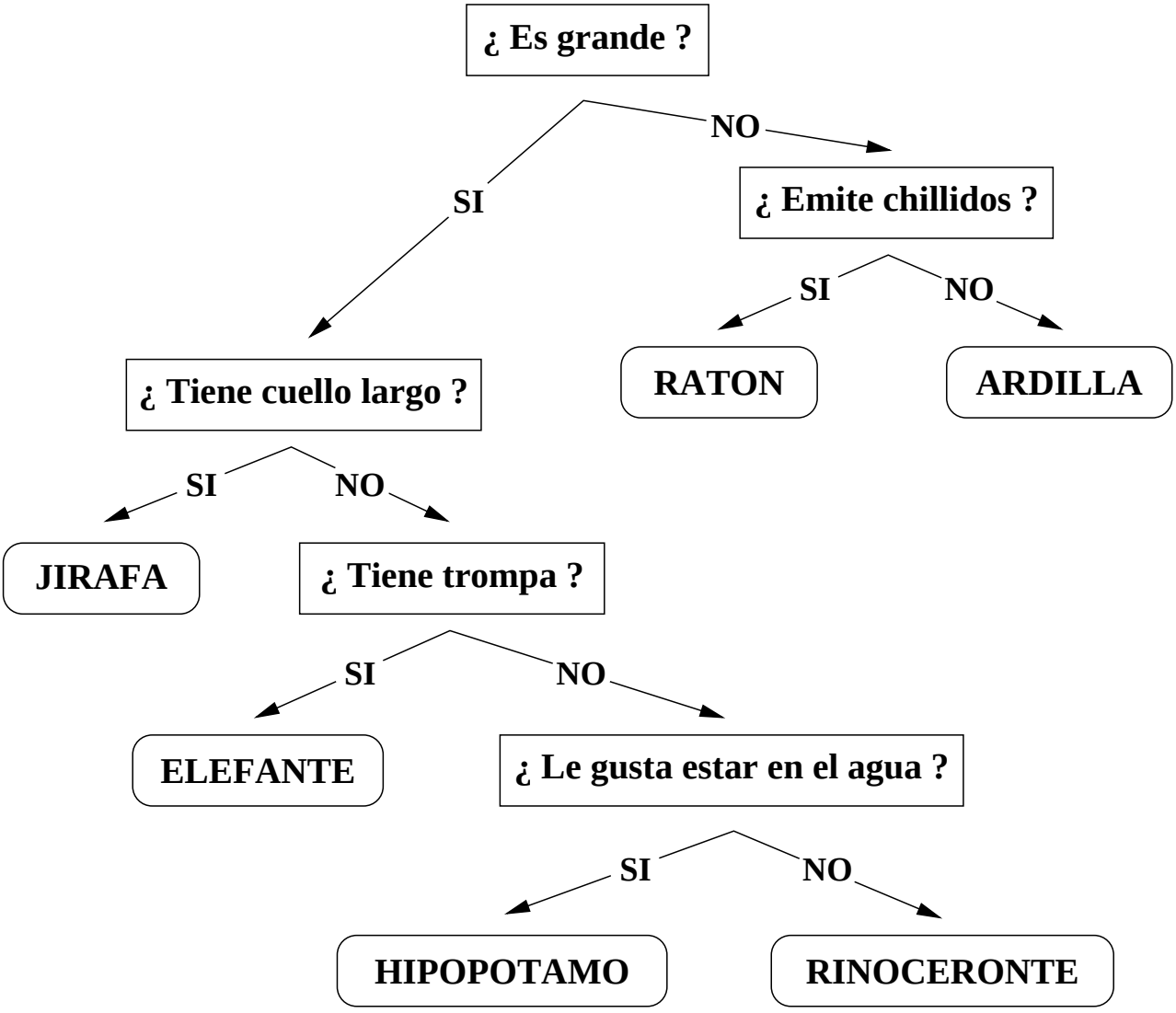
Universidad de Sevilla

Decisión y metaintérpretes

- Árboles de decisión
- Razonamiento regresivo
- Bibliografía:
  - Giarratano, J.C. y Riley, G. "Expert Systems Principles and Programming (2nd ed.)" (PWS Pub. Co., 1994)
    * Cap. 12.3: "Decision Trees"
    * Cap. 12.4: "Backward Chaining"
  - Lucas, P. y Gaag L.v.d. "Principles of expert systems" (Addison Wesley, 1991)
    * Cap. 3: "Production rules and inference"
  - Harmon, P. y King D. "Sistemas expertos" (Díaz de Santos, 1988)

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**

```
;;; REGLA: inicializacion
;;; SI
;;;   no hay nodo raíz
;;; ENTONCES
;;;   cargamos los datos del fichero "animales.dat"
;;;   e indica que el nodo actual es el nodo raíz.

(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**

```
;;; REGLA: nodo-decision
;;; SI
;;;   el nodo actual es un nodo decisión y
;;;   no tenemos ninguna respuesta del usuario
;;; ENTONCES
;;;   requerimos del usuario una respuesta a la
;;;     pregunta que tiene asociada el nodo decisión
;;;     actual y
;;;   almacenamos la respuesta.

(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))))
```

Árboles de decisión

- **Respuesta incorrecta**

```
;;; REGLA: respuesta-incorrecta
;;; SI
;;;   la respuesta proporcionada por el usuario no es
;;;      "si" ni "no"
;;; ENTONCES
;;;   eliminamos la respuesta incorrecta y
;;;   le indicamos al usuario que se ha equivocado.

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

Árboles de decisión

- **Respuesta afirmativa en los nodos internos**

```
;;; REGLA: procesa-si-en-nodo-decision
;;; SI
;;;   el nodo actual es un nodo decisión y
;;;   tenemos una respuesta afirmativa del usuario
;;; ENTONCES
;;;   cambiamos el nodo actual de manera adecuada.

(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**

```
;;; REGLA: procesa-no-en-nodo-decision
;;; SI
;;;   el nodo actual es un nodo decisión y
;;;   tenemos una respuesta negativa del usuario
;;; ENTONCES
;;;   cambiamos el nodo actual de manera adecuada.

(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)))
```

Árboles de decisión

- **Tratamiento de los nodos hoja**

```
;;; REGLA: nodo-respuesta
;;; SI
;;;   el nodo actual es un nodo respuesta y
;;;   no tenemos ninguna respuesta del usuario
;;; ENTONCES
;;;   el sistema indica al usuario la posible
;;;     respuesta y
;;;   se pregunta al usuario si la respuesta es
;;;     correcta (según su criterio) y
;;;   almacenamos la respuesta.

(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))))
```

Árboles de decisión

- **Clasificación satisfactoria**

```
;;; REGLA: procesa-si-en-nodo-respuesta
;;; SI
;;;   el nodo actual es un nodo respuesta y
;;;   tenemos una respuesta afirmativa del usuario
;;; ENTONCES
;;;   pasamos a la fase en la que se pregunta al
;;;     usuario si quiere continuar.

(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**

```
;;; REGLA: procesa-no-en-nodo-respuesta
;;; SI
;;;   el nodo actual es un nodo respuesta y
;;;   tenemos una respuesta negativa del usuario
;;; ENTONCES
;;;   el sistema pasa a la fase de aprendizaje.

(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)))
```

Árboles de decisión

- **¿Quieres continuar?**

```
;;; REGLA: pregunta-de-nuevo
;;; SI
;;;   estamos en la fase en la que se pregunta al
;;;     usuario si quiere continuar y
;;;   no tenemos ninguna respuesta del usuario
;;; ENTONCES
;;;   se pregunta al usuario si quiere realizar otra
;;;     consulta y
;;;   almacenamos la respuesta.

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

Árboles de decisión

- **Se vuelve a hacer una consulta**

```
;;; REGLA: una-vez-mas
;;; SI
;;;   estamos en la fase en la que se pregunta al
;;;     usuario si quiere continuar y
;;;   tenemos una respuesta afirmativa del usuario
;;; ENTONCES
;;;   se comienza de nuevo desde el nodo raíz.

(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**

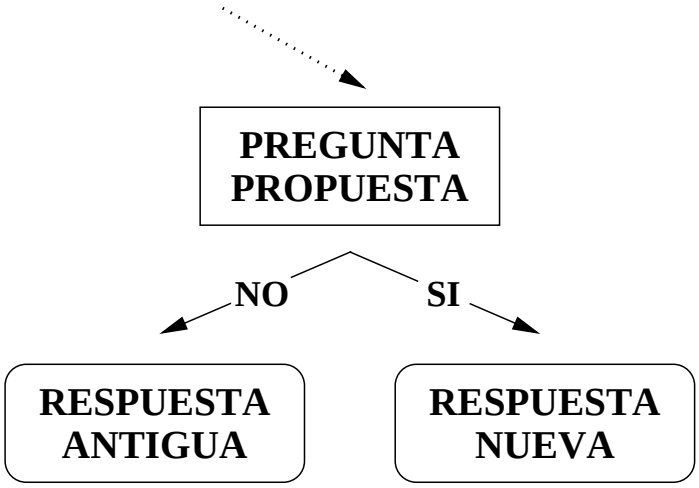
```
;;; REGLA: no-mas-por-favor
;;; SI
;;;   estamos en la fase en la que se pregunta al
;;;     usuario si quiere continuar y
;;;   tenemos una respuesta negativa del usuario
;;; ENTONCES
;;;   se salvan los hechos que componen el árbol de
;;;     decisión en el fichero "animales.dat"

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

Árboles de decisión

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

  - **Árbol modificado:**



Árboles de decisión

```
;;; REGLA: aprende
;;; SI
;;;   estamos en la fase de aprendizaje
;;; ENTONCES
;;;   el sistema pregunta al usuario la respuesta
;;;     correcta y la pregunta que tiene asociada y
;;;   modifica de manera conveniente el árbol de
;;;     decisión.

(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)))
```

Razonamiento regresivo

- **Forma de las reglas**

```
(regla (si <variable> es|mayor|menor <valor>
           { y <variable> es|mayor|menor <valor> }*)
       (entonces <variable> es <valor>))
```

- **Objetivo inicial**

```
(objetivo-inicial <variable>)
```

- **Ejemplo (teatro.dat)**

```
(regla (si distancia mayor 5000)
       (entonces tipo-de-medio es coche))
(regla (si distancia mayor 1000 y
           premura menor 15)
       (entonces tipo-de-medio es coche))
(regla (si distancia menor 1000 y
           premura mayor 15)
       (entonces tipo-de-medio es andando))
(regla (si tipo-de-medio es coche y
           teatro-centrico es si)
       (entonces medio es taxi))
(regla (si tipo-de-medio es coche y
           teatro-centrico es no)
       (entonces medio es coche-propio))
(regla (si tipo-de-medio es andando y
           tiempo es malo)
       (entonces medio es andando-con-abrigo))
(regla (si tipo-de-medio es andando y
           tiempo es bueno)
       (entonces medio es andando))

(objetivo-inicial medio)
```

Razonamiento regresivo

- **Ejemplo (teatro.dat)**

```
(pregunta (atributo distancia)
          (texto "Dime la distancia al teatro (en metros)"))
(pregunta (atributo premura)
          (texto "Dime cuantos minutos quedan para el comienzo"))
(pregunta (atributo tiempo)
          (texto "Dime si el tiempo es bueno o malo"))
(pregunta (atributo teatro-centrico)
          (texto "Dime si el teatro esta en el centro o no"))
```

Razonamiento regresivo

- **Sesión:**

```
CLIPS> (load "regresivo.clp")
+%%****************
TRUE
CLIPS> (reset)
CLIPS> (run)
Dime la distancia al teatro (en metros)
700
Dime cuantos minutos quedan para el comienzo
40
Dime si el tiempo es bueno o malo
malo
Solución: medio : andando-con-abrigo
CLIPS> (facts)
f-0     (initial-fact)
...
f-12    (objetivo-inicial medio)
f-16    (atributo distancia 700)
f-19    (atributo premura 40)
f-20    (atributo tipo-de-medio andando)
f-24    (atributo tiempo malo)
f-25    (atributo medio andando-con-abrigo)
For a total of 11 facts.
CLIPS>
```

Razonamiento regresivo

- **Módulo MAIN**

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

(deftemplate MAIN::regla
  (multislot si)
  (multislot entonces))

(deftemplate MAIN::pregunta
  (slot atributo)
  (slot pregunta))
```

Razonamiento regresivo

```
;;; REGLA: inicial
;;;   cargamos los hechos del fichero "reglas.dat".

(defrule MAIN::inicial
  =>
  (load-facts "teatro.dat"))

;;; REGLA: objetivo-inicial
;;; SI
;;;   tenemos un atributo como objetivo inicial y
;;;   no se ha obtenido su valor
;;; ENTONCES
;;;   almacenamos el atributo como objetivo.

(defrule MAIN::objetivo-inicial
  (objetivo-inicial ?nombre)
  (not (atributo ?nombre ?))
  =>
  (assert (objetivo ?nombre)))

;;; REGLA: solucion
;;; SI
;;;   tenemos un atributo como objetivo inicial y
;;;   se ha obtenido su valor
;;; ENTONCES
;;;   presentamos el atributo y el valor al usuario.

(defrule MAIN::solucion
  (objetivo-inicial ?nombre)
  (atributo ?nombre ?valor)
  =>
  (printout t "Solución: " ?nombre ": " ?valor crlf))
```

Razonamiento regresivo

```
;;; REGLA: intenta-regla
;;; SI
;;;   tenemos cierto atributo como objetivo y
;;;   dicho atributo no tiene asignado valor y
;;;   existe una regla que le asigna cierto valor
;;;     a dicho atributo en sus acciones
;;; ENTONCES
;;;   incluimos como nuevo objetivo el primer
;;;     atributo que aparece en las condiciones de
;;;     la regla.

(defrule MAIN::intenta-regla
  (objetivo ?nombre)
  (not (atributo ?nombre ?))
  (regla (si ?variable $?)
         (entonces ?nombre es ?))
  =>
  (assert (objetivo ?variable)))
```

Razonamiento regresivo

```
;;; REGLA: pregunta-valor-atributo
;;; SI
;;;   tenemos cierto atributo como objetivo y
;;;   dicho atributo no tiene valor asociado y
;;;   no hay reglas que concluyan algún valor
;;;     para ese atributo y
;;;   existe una pregunta asociada al atributo
;;; ENTONCES
;;;   planteamos la pregunta al usuario y
;;;   almacenamos su respuesta como valor asociado
;;;     a dicho atributo.

(defrule MAIN::pregunta-valor-atributo
  ?objetivo <- (objetivo ?nombre)
  (not (atributo ?nombre ?))
  (not (regla (entonces ?nombre ? ?)))
  ?pregunta <- (pregunta (atributo ?nombre)
                         (texto ?texto))
  =>
  (retract ?objetivo ?pregunta)
  (printout t ?texto crlf)
  (assert (atributo ?nombre (read))))
```

Razonamiento regresivo

- **Módulo RAZONA**

```
(defmodule RAZONA
  (import MAIN deftemplate objetivo
                           atributo
                           regla))

;;; REGLA: objetivo-satisfecho
;;; SI
;;;   tenemos cierto atributo como objetivo
;;;   y dicho atributo tiene asignado valor
;;; ENTONCES
;;;   eliminamos el objetivo.

(defrule RAZONA::objetivo-satisfecho
  (declare (auto-focus TRUE))
  ?objetivo <- (objetivo ?nombre)
  (atributo ?nombre ?)
  =>
  (retract ?objetivo))
```

Razonamiento regresivo

```
;;; REGLA: regla-satisfecha
;;; SI
;;;   la condición de una regla menciona un atributo y
;;;   el atributo tiene asociado un valor y
;;;   dicho valor cumple la condición de la regla
;;; ENTONCES
;;;   eliminamos la regla e
;;;   indicamos que el atributo que aparece en las
;;;     acciones de la regla toma el valor indicado.

(defrule RAZONA::regla-satisfecha-es
  (declare (auto-focus TRUE))
  (atributo ?nombre ?v1)
  ?r <- (regla (si ?nombre es ?v1)
               (entonces ?objetivo es ?v2))
  => (retract ?r)
     (assert (atributo ?objetivo ?v2)))

(defrule RAZONA::regla-satisfecha-mayor
  (declare (auto-focus TRUE))
  (atributo ?nombre ?v1)
  ?r <- (regla (si ?nombre mayor ?v2&:(> ?v1 ?v2))
                   (entonces ?objetivo es ?v3))
  => (retract ?r)
     (assert (atributo ?objetivo ?v3)))

(defrule RAZONA::regla-satisfecha-menor
  (declare (auto-focus TRUE))
  (atributo ?nombre ?v1)
  ?r <- (regla (si ?nombre menor ?v2&:(< ?v1 ?v2))
                   (entonces ?objetivo es ?v3))
  => (retract ?r)
     (assert (atributo ?objetivo ?v3)))
```

Razonamiento regresivo

```
;;; REGLA: elimina-regla-no-util
;;; SI
;;;   en la condición de una regla aparece cierto
;;;     atributo y
;;;   el atributo tiene asociado un valor y
;;;   dicho valor no cumple la condición de la regla
;;; ENTONCES
;;;   dicha regla nunca se va a satisfacer y la
;;;     podemos eliminar.

(defrule RAZONA::elimina-regla-no-util-es
  (declare (auto-focus TRUE))
  (atributo ?nombre ?valor)
  ?r <- (regla (si ?nombre es ~?valor $?))
  =>
  (retract ?r))

(defrule RAZONA::elimina-regla-no-util-mayor
  (declare (auto-focus TRUE))
  (atributo ?nombre ?valor1)
  ?r <- (regla (si ?nombre mayor ?valor2
                   &:(not (> ?valor1 ?valor2)) $?))
  =>
  (retract ?r))

(defrule RAZONA::elimina-regla-no-util-menor
  (declare (auto-focus TRUE))
  (atributo ?nombre ?valor1)
  ?r <- (regla (si ?nombre menor ?valor2
                   &:(not (< ?valor1 ?valor2)) $?))
  =>
  (retract ?r))
```

Razonamiento regresivo

```
;;; REGLA: modifica-regla-util
;;; SI
;;;   en la condición de una regla aparece cierto
;;;     atributo en primer lugar y
;;;   la regla tiene más condiciones y
;;;   el atributo ya tiene asignado valor y
;;;   dicho valor cumple la condición de la regla
;;; ENTONCES
;;;   modificamos la regla eliminando la condición
;;;     asociada al atributo pues se satisface.

(defrule RAZONA::modifica-regla-util-es
  (declare (auto-focus TRUE))
  (atributo ?nombre ?valor)
  ?r <- (regla (si ?nombre es ?valor y $?resto))
  =>
  (modify ?r (si $?resto)))

(defrule RAZONA::modifica-regla-util-mayor
  (declare (auto-focus TRUE))
  (atributo ?nombre ?valor1)
  ?r <- (regla (si ?nombre mayor ?valor2
                   &:(> ?valor1 ?valor2) y $?resto))
  =>
  (modify ?r (si $?resto)))

(defrule RAZONA::modifica-regla-util-menor
  (declare (auto-focus TRUE))
  (atributo ?nombre ?valor1)
  ?r <- (regla (si ?nombre menor ?valor2
                   &:(< ?valor1 ?valor2) y $?resto))
  =>
  (modify ?r (si $?resto)))
```