Razonamiento en sistemas de conocimiento basados en reglas

José A. Alonso y Francisco J. Martín

Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Hechos y variables

- **Constantes:**
  - **Juan, María, 123, hombre, mujer**
- **Tipos:**
  - **{sí, no}, Entero**
- **Variables simples:**
  - **edad, fuma**
- **Variables múltiples:**
  - **trastorno, síntomas**
- **Declaración de variables:**
  - **Tipadas:** $x^s : \tau$, $x^m : 2^\tau$
  - **edad:Entero, síntomas:**$2^{\{\textbf{tos, fiebre}\}}$
  - **No tipadas:** x^s, x^m

Hechos y variables

- **Declaración de dominio:**

  D = { edad:Entero
  fuma:{sí, no}
  síntoma:$2^{\{}$tos, fiebre$\}$
  enfermedad:{resfriado, gripe} }

- **Hechos**
  - **Hecho simple:** $x^s = c$
  - **edad=27, fuma=si**
  - **Hecho múltiple:** $X^m = C$
  - **sintoma={tos, fiebre}**
  - x^s **= desconocido**
  - x^m **= desconocido**

- **Conjunto de hechos (o memoria de trabajo):**
  - $\{x_1^s = c_1, \ldots, x_p^s = c_p, x_1^m = C_1, \ldots, x_q^m = C_q\}$

Sintaxis de reglas de producción

```
<regla-de-producción>
  ::= SI <antecedente> ENTONCES <consecuente> FIN
<antecedente>
  ::= <disyunción> [Y <disyunción>]*
<disyunción>
  ::= <condición> [O <condición>]*
<condición>
  ::= <predicado-2>(<variable>,<constante>) |
      <predicado-1>(<variable>)
<predicado-2>
  ::= igual | distinto | mayor-que | menor-que | ...
<predicado-1>
  ::= conocido | desconocido | ...
<consecuente>
  ::= <conclusión> [Y <conclusión>]*
<conclusión>
  ::= <acción>(<variable>,<constante>)
<acción>
  ::= añadir | eliminar | ...
```

Reglas de producción

- **Si el paciente tiene arterioesclerosis y calambre en las piernas cuando anda, entonces tiene estenosis arterial.**

```
SI
   igual(trastorno, arterioesclerosis) Y
   igual(sintomas, calambres-en-las-piernas) Y
   igual(presencia, andando)
ENTONCES
   añadir(diagnóstico, estenosis-arterial)
FIN
```

- **Si el paciente fuma y tiene más de 60 años, entonces tiene arterioesclerosis.**

```
SI
   mayor(edad, 60) Y
   igual(fuma, si)
ENTONCES
   añadir(trastorno, arterioesclerosis)
FIN
```

Reglas de producción

- **Significado de predicados:**

Predicado	Variable simple	Variable múltiple
$igual(x, c)$	$x = c$	$c \in x$
$distinto(x, c)$	$x \neq c$ **y** $x \neq desconocido$	$c \notin x$ **y** $x \neq desconocido$
$menor(x, c)$	$x < c$	$--$
$mayor(x, c)$	$x > c$	$--$
$conocido(x)$	$x \neq desconocido$	$x \neq desconocido$
$desconocido(x)$	$x = desconocido$	$x = desconocido$

- **Significado de acciones:**

Acción	Variable simple	Variable múltiple
$anadir(x, c)$	$x := c$	$x := x \cup \{c\}$
$eliminar(x, c)$	$x := desconocido$	**si** $x = \{c\}$
		entonces $x := desconocido$
		si no $x := x - \{c\}$

- **Base de conocimiento:**
  - **Declaración de dominio**
  - **Base de reglas**

Objeto–atributo–valor

- **Terna objeto–atributo–valor (OAV)**

```
<paciente, edad, 37>
<paciente, síntomas, {tos, fiebre}>
```

- **Sintaxis de reglas de producción con OAV**

```
<condición>
  ::= <predicado>(<objeto>,<atributo>,<valor>)
<conclusión>
  ::= <acción>(<objeto>,<atributo>,<valor>)
```

- **Ejemplos de reglas de producción con OAV**

```
igual(paciente, edad, 37)
igual(paciente, sintomas, {tos, fiebre})
```

- **Significado de predicados y acciones**

	Variable simple	Variable múltiple
$igual(o, a, v)$	$a(o) = v$	$a(o, v_i)$
$distinto(x, c)$	$a(o) \neq v$	$\neg a(o, v)$
$menor(x, c)$	$a(o) < v$	$--$
$mayor(x, c)$	$a(o) > v$	$--$
$anadir(x, c)$	$a(o) := v$	$a(o, v_i)$

Razonamientos progresivo y regresivo

- **Ejemplo**

```
Base de reglas
  R1: SI b ENTONCES g FIN
  R2: SI g ENTONCES c FIN
  R3: SI a ENTONCES b FIN
Hechos
  {a}
```

- **Razonamiento progresivo**

```
BH = {a} ->
   -> {a, b}       por R3
   -> {a, b, g}    por R1
   -> {a, b, g, c} por R2
```

- **Razonamiento regresivo**

```
Objetivo = {g}
         -> {b}  por R1
         -> {a}  por R3
         -> {}   Por Hechos
```

Tipos de variables

- **Variables objetivos**
  - x_o^s, x_o^m
  - $enfermedad_o^m : 2^{\{resfriado, gripe\}}$
- **Variables preguntables**
  - x_p^s, x_p^m
  - **temperatura, edad**
- **Variables deducibles**
  - **fiebre**

Ejemplo de B.C.: diagnostico.bc

- **Declaración del dominio**

```
(defvariable sintomas
   (pregunta "Cuáles son los sintomas del paciente?:")
   (es-objetivo nil))

(defvariable presencia
   (pregunta "Cuándo aparecen los sintomas?:")
   (es-objetivo nil))

(defvariable edad
   (pregunta "Cual es la edad el paciente?:")
   (es-objetivo nil))

(defvariable fuma
   (pregunta "El paciente fuma?:")
   (es-objetivo nil))

(defvariable diagnostico
   (pregunta nil)
   (es-objetivo t))

(defvariable trastorno
   (pregunta nil)
   (es-objetivo nil))
```

Ejemplo de B.C.: diagnostico.bc

- **Base de reglas**

```
;;; REGLA 1:
;;; SI el paciente tiene arterioesclerosis Y
;;;    calambre en las piernas cuando anda,
;;; ENTONCES tiene estenosis arterial.
(defregla regla-1
   (and
      (igual trastorno arterioesclerosis)
      (igual sintomas calambres-en-las-piernas)
      (igual presencia andando))
   (añade diagnostico estenosis-arterial))

;;; REGLA 2:
;;; SI el paciente fuma Y
;;;    tiene más de 60 años,
;;; ENTONCES tiene arterioesclerosis.
(defregla regla-2
   (and
      (mayor edad 60)
      (igual fuma si))
   (añade trastorno arterioesclerosis))
```

Ejemplos de B.C.: diagnostico.bc

- **Sesión**

```
> (load "mir-1")
T
> (consulta)

Nombre de la base de conocimientos: "diagnostico.bc"

Cual es la edad el paciente?:
-> (70)

El paciente fuma?:
-> (si)

Cuales son los sintomas del paciente?:
-> (calambres-en-las-piernas)

Cuando aparecen los sintomas?:
-> (andando)

DIAGNOSTICO = (ESTENOSIS-ARTERIAL)
NIL
```

Ejemplo de B.C.: `teatro.bc`

- **Referencia**
  - **P. Harmon y D. King "Expert systems, artificial intelligence in business." (John Wiley & Sons, 1985) p. 77**
- **Reglas**
  - **R1: Si la distancia al teatro es mayor de 5 Km, entonces ir en coche**
  - **R2: Si la distancia al teatro es mayor de 1 Km y queda menos de 15 minutos, entonces ir en coche**
  - **R3: Si la distancia al teatro es menor de 1 Km y queda más de 15 minutos, entonces ir andando**
  - **R4: Si hay que ir en coche y el teatro está en el centro de la ciudad, entonces coger un taxi**
  - **R5: Si hay que ir en coche y el teatro no está en el centro de la ciudad, entonces coger el coche propio**
  - **R6: Si hay que ir andando y el tiempo es malo, entonces coger un abrigo y andar**
  - **R7: Si hay que ir andando y el tiempo es bueno, entonces andar**

Ejemplo de B.C.: teatro.bc

- **Declaración del dominio**

```
(defvariable distancia
   (pregunta "Dime la distancia al teatro (en metros)")
   (es-objetivo nil))

(defvariable premura
   (pregunta "Dime cuantos minutos quedan para el comienzo"
   (es-objetivo nil))

(defvariable tiempo
   (pregunta "Dime si el tiempo es bueno o malo")
   (es-objetivo nil))

(defvariable es-el-teatro-centrico
   (pregunta "Dime si el teatro esta en el centro o no")
   (es-objetivo nil))

(defvariable medio
   (pregunta nil)
   (es-objetivo objetivo))

(defvariable tipo-de-medio
   (pregunta nil)
   (es-objetivo nil))
```

Ejemplo de B.C.: teatro.bc

- **Base de reglas**

```
;;; Si la distancia al teatro es mayor de 5 Km,
;;; entonces ir en coche
(defregla ir-en-coche-1
   (mayor distancia 5000)
   (annade tipo-de-medio coche))

;;; Si la distancia al teatro es mayor de 1 Km
;;;    y queda menos de 15 minutos,
;;; entonces ir en coche
(defregla ir-en-coche-2
   (and (mayor distancia 1000)
        (menor premura 15))
   (annade tipo-de-medio coche))

;;; Si la distancia al teatro es menor de 1 Km
;;;    y queda más de 15 minutos,
;;; entonces ir andando
(defregla ir-a-pie
   (and (menor distancia 1000)
        (mayor premura 15))
   (annade tipo-de-medio pie))
```

Ejemplo de B.C.: teatro.bc

```
;;; Si hay que ir en coche
;;;    y el teatro está en el centro de la ciudad,
;;; entonces coger un taxi
(defregla ir-en-taxi
   (and (igual tipo-de-medio coche)
        (igual es-el-teatro-centrico si))
   (annade medio en-taxi))

;;; Si hay que ir en coche
;;;    y el teatro no está en el centro de la ciudad,
;;; entonces coger el coche propio
(defregla ir-en-coche-propio
   (and (igual tipo-de-medio coche)
        (igual es-el-teatro-centrico no))
   (annade medio en-coche-propio))

;;; Si hay que ir andando
;;;    y el tiempo es malo,
;;; entonces coger un abrigo y andar
(defregla ir-andando-con-abrigo
   (and (igual tipo-de-medio pie)
        (igual tiempo malo))
   (annade medio andando-con-abrigo))

;;; Si hay que ir andando
;;;    y el tiempo es bueno,
;;; entonces andar
(defregla ir-andando
   (and (igual tipo-de-medio pie)
        (igual tiempo bueno))
   (annade medio andando))
```

Ejemplo de B.C.: teatro.bc

- **Sesión**

```
>  (load "mir-1")
T
> (consulta)

Nombre de la base de conocimientos: "teatro.bc"

Dime la distancia al teatro (en metros)
-> (700)

Dime cuantos minutos quedan para que comience la obra
-> (20)

Dime si el tiempo es bueno o malo
-> (malo)

MEDIO = (ANDANDO-CON-ABRIGO)
NIL
```

Ejemplo de B.C.: proposicional.bc

- **Declaración del dominio**

```
(defvariable x
   (pregunta "Cuanto vale x?:")
   (es-objetivo nil))

(defvariable y
   (pregunta nil)
   (es-objetivo nil))

(defvariable z
   (pregunta nil)
   (es-objetivo t))

(defvariable u
   (pregunta "Cuanto vale u?:")
   (es-objetivo nil))

(defvariable v
   (pregunta nil)
   (es-objetivo nil))

(defvariable w
   (pregunta "Cuanto vale w?:")
   (es-objetivo nil))
```

Ejemplo de B.C.: proposicional.bc

- **Base de reglas**

```
(defregla regla-1
 (and (igual w a)
      (igual x b))
 (añade v c))

(defregla regla-2
 (and (igual w d)
      (igual v c))
 (añade y e))

(defregla regla-3
  (igual v c)
  (añade z k))

(defregla regla-4
 (and (igual x j)
      (igual y e))
 (añade z h))

(defregla regla-5
 (and (igual u f)
      (igual x g))
 (añade z i))
```

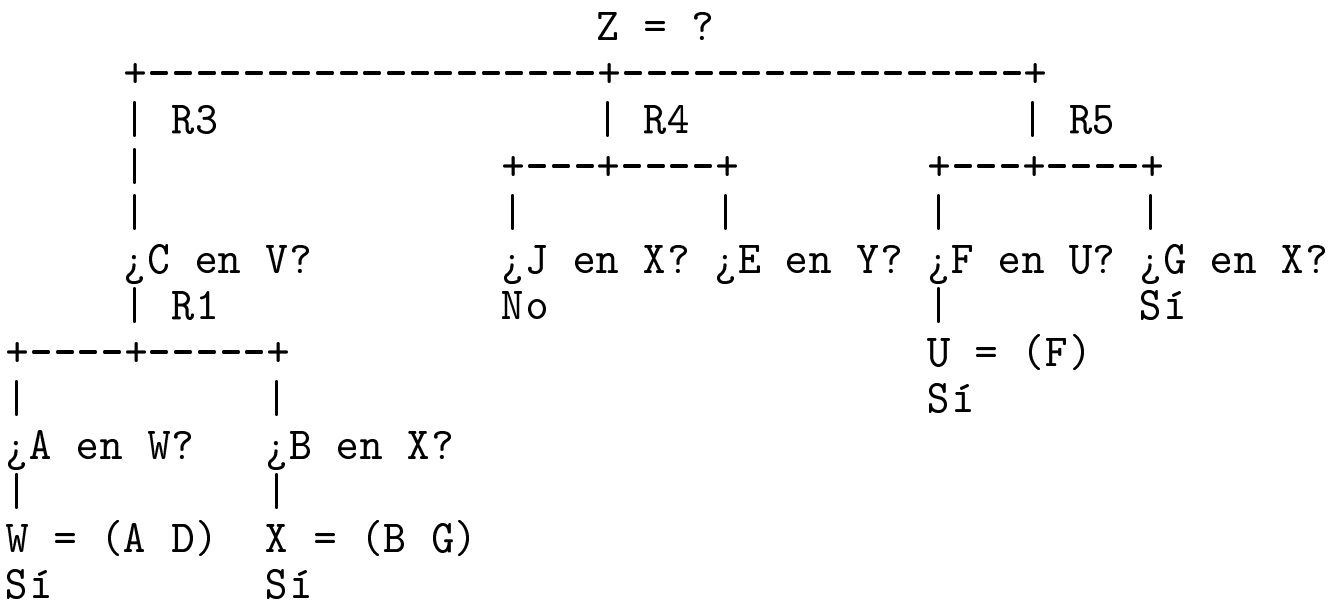
Ejemplo de B.C.: proposicional.bc

- **Sesión**

```
> (load "mir-1")
T
> (consulta)

Nombre de la base de conocimientos: "proposicional.bc"

Cuanto vale w?:
-> (a d)

Cuanto vale x?:
-> (b g)

Cuanto vale u?:
-> (f)

Z = (I K)
NIL
```

Ejemplo de B.C.: proposicional.bc

- **Arbol de inferencia**



Ejemplo de B.C.: `proposicional.bc`

Determina	Agenda	Aplica	Condiciones	Valores
Z	**R3, R4, R5**			
Z	**R4, R5**	**R3**	**C** $\in$ **V**	
V	**R1**			
V		**R1**	**A** $\in$ **W**, **B** $\in$ **X**	
W				**W** = **(A D)**
V		**R1**	**B** $\in$ **X**	
X				**X** = **(B G)**
V				**V** = **(C)**
Z	**R4, R5**			**Z** = **(K)**
Z	**R5**	**R4**	**J** $\in$ **X**, **E** $\in$ **Y**	
Z		**R5**	**F** $\in$ **U**, **G** $\in$ **X**	
U				**U** = **(F)**
Z				**Z** = **(I K)**

Motor de inferencia regresivo

- **Declaración de variables globales**

```
(defvar *variables-definidas*)
(defvar *base-de-reglas*)
```

- **Representación de variables**

```
(defstruct (n-variable (:constructor crea-variable)
                       (:conc-name nil)
                       (:print-function escribe-variable))
  nombre-de-variable
  valor
  pregunta
  es-objetivo
  ha-sido-determinada)

(defun escribe-variable (variable
                         &optional (canal t) profundidad)
  (format canal "~a*" (nombre-de-variable variable)))
```

- **Representación de reglas**

```
(defstruct (regla (:constructor crea-regla)
                  (:conc-name nil)
                  (:print-function escribe-regla))
  nombre-de-regla
  antecedente
  consecuente)

(defun escribe-regla (regla
                      &optional (canal t) profundidad)
  (format canal "~a*" (nombre-de-regla regla)))
```

Motor de inferencia regresivo

- **Consulta de la base de conocimiento**

```
;;; (CONSULTA)
;;; 1. Lee y analiza la base de conocimientos,
;;; 2. determina los valores de las variables objetivos y
;;; 3. escribe los valores de las variables objetivos.
(defun consulta ()
  (lee-base-de-conocimiento)   ;1
  (resuelve-objetivos)         ;2
  (escribe-objetivos))         ;3
```

Lisp: Manejo de ficheros

- **El procedimiento** `with-open-file`**:**

```
> (with-open-file (fich1 "fichero.dat" :direction :output)
    (format fich1 "uno ~%dos ~%tres"))
NIL
> (with-open-file (a "fichero.dat" :direction :input)
    (loop for n from 1 to 3 do (print (read a))))

UNO
DOS
TRES
NIL
> (with-open-file (a "fichero.dat" :direction :input)
    (loop for n from 1 to 5 do (print (read a))))

UNO
DOS
TRES
*** - READ: input stream #<STRING-CHAR-FILE-STREAM #"ficher
has reached its end
1. Break> abort

> (with-open-file (a "fichero.dat" :direction :input)
    (loop for n from 1 to 5 do (print (read a nil))))

UNO
DOS
TRES
#<END OF FILE>
#<END OF FILE>
NIL
```

Lisp: Manejo de ficheros

```
> (with-open-file (a "fichero.dat" :direction :input)
    (loop
      (let ((x (read a nil 'fdf)))
        (if (eq x 'fdf)
            (return)
            (format t "~&~a" x)))))
UNO
DOS
TRES
NIL
```

Ejemplo de uso de ficheros

- **Contenidos de** `notas.dat`**:**

```
((AGUADO CASAS) (JUAN JOSE) 5)
((AGUILA PUENTE) (RAFAEL) 2)
((ALBA ADAME) (CARLOS) 9)
((ALCALDE PEREZ) (MARIA LUISA) 3)
```

- **Sesión**

```
> (shell "ls aprobados*")
aprobados.lsp
> (load "aprobados.lsp")
T
> (aprobados)
NIL
> (shell "ls aprobados*")
aprobados.dat  aprobados.lsp
> (shell "cat aprobados.dat")
((AGUADO CASAS) (JUAN JOSE) 5)
((ALBA ADAME) (CARLOS) 9)0
>
```

Ejemplo de uso de ficheros

- **Procedimiento** aprobados

```
(defun aprobados ()
  (with-open-file (notas "notas.dat" :direction :input)
     (with-open-file
       (aprobados "aprobados.dat" :direction :output)
        (loop
         (let ((alumno (read notas nil 'fdf)))
           (cond ((eq alumno 'fdf)
                  (return))
                 ((esta-aprobado alumno)
                  (format aprobados "~&~a" alumno)))))))))

(defun esta-aprobado (alumno)
  (>= (third alumno) 5))
```

Motor de inferencia regresivo

- **Lectura de la base de conocimiento**

```
(LEE-BASE-DE-CONOCIMIENTO)
1. Pregunta el nombre del fichero que contiene la base
   de conocimientos;
2. inicia las variables globales, variables definidas
   y base de reglas, con la lista vacía;
3. analiza sintácticamente la base de conocimientos
   actualizando los valores de las variables definidas
   y la base de reglas;
4. ordena las variables y las reglas como se encuentran
   en la base de conocimiento;
5. devuelve T.
(defun lee-base-de-conocimiento ()
  (format t "~&~%Nombre de la base de conocimientos: ")
  (setf *variables-definidas* nil
        *base-de-reglas* nil)
  (with-open-file
    (base-de-conocimientos (read) :direction :input)
    (analiza base-de-conocimientos))
  (setf *variables-definidas*
        (nreverse *variables-definidas*)
        *base-de-reglas*
        (nreverse *base-de-reglas*))
  t)
```

Motor de inferencia regresivo

- **Ejemplo de lectura de la base de conocimiento**

```
> (load "mir-1")
T
> (lee-base-de-conocimiento)
Nombre de la base de conocimientos: "proposicional.bc"
T
> *variables-definidas*
(X* Y* Z* U* V* W*)
> x
X*
> (nombre-de-variable x)
X
> (valor x)
NIL
> (pregunta x)
"Cuanto vale x?:"
> (es-objetivo x)
NIL
> (ha-sido-determinada x)
NIL
> regla-1
REGLA-1*
> (nombre-de-regla regla-1)
REGLA-1
> (antecedente regla-1)
(AND (IGUAL W A) (IGUAL X B))
> (consecuente regla-1)
((AÑADE V C))
```

Motor de inferencia regresivo

- **Análisis de la base de conocimiento**

```
(ANALIZA BASE-DE-CONOCIMIENTO)
Valor: - NIL, si la BASE-DE-CONOCIMIENTO es
         sintácticamente correcta;
       - ERROR, en caso contrario.
Efecto: Actualiza los valores de las *VARIABLES-DEFINIDAS*
  y la *BASE-DE-REGLAS*.
Procedimiento:
1. Va leyendo las expresiones de la base de conocimiento y
2. si ha alcanzado el final la base de conocimiento,
   devuelve NIL y termina;
3. si la expresión es una definición de variable
   (i.e. empieza por defvariable), analiza la definición
   y continúa con las restantes expresiones de la base de
   conocimiento;
4. si la expresión es una definición de regla
   (i.e. empieza por defregla), analiza la definición
   y continúa con las restantes expresiones de la base de
   conocimiento;
5. en otro caso, devuelve error (indicando la palabra que
   lo origina) y termina.
(defun analiza (base-de-conocimiento)
  (let ((expr (lee-expresion base-de-conocimiento)))    ;1
    (cond
     ((eq expr 'fdf) nil)                               ;2
      (t (case (first expr)
           (defvariable (analiza-def-variable expr))    ;3
           (defregla    (analiza-def-regla expr))       ;4
           (otherwise   (error "Palabra desconocida: ~a" ;5
                               (first expr))))
         (analiza base-de-conocimiento)))))
```

Motor de inferencia regresivo

- **Lectura de expresiones de la base de conocimiento**

```
(LEE-EXPRESION CANAL)
Valor:
- una expresión del CANAL y pasa a la siguiente, si no se
  encuentra al final del fichero;
- FDF, si se encuentra al final del fichero.
(defun lee-expresion (canal)
  (read canal nil 'fdf))
```

Motor de inferencia regresivo

- **Sintaxis de definición de variables**

```
<defvariable>
  ::= (defvariable <nombre de la variable>
        (pregunta <pregunta sobre la variable>)
        (es-objetivo <booleano>))
<nombre de la variable>
  ::= <simbolo>
<pregunta sobre la variable>
  ::= <cadena>
```

- **Ejemplos de definiciones de variables**

```
(defvariable sintomas
   (pregunta "Cuáles son los sintomas del paciente?:")
   (es-objetivo nil))

(defvariable diagnostico
   (pregunta nil)
   (es-objetivo t))

(defvariable trastorno
   (pregunta nil)
   (es-objetivo nil))
```

Motor de inferencia regresivo

- **Análisis de definiciones de variables**

```
(ANALIZA-DEF-VARIABLE DEF-VARIABLE)
Valor: La lista obtenida añadiendo el nombre de la
 variable de la DEF-VARIABLE a la lista de las
 *VARIABLES-DEFINIDAS*.
Efectos:
 1. Asigna como valor del nombre de la DEF-VARIABLE la
     correspondiente variable
 2. Añade la variable a la lista de las
    *VARIABLES-DEFINIDAS*.
(defun analiza-def-variable (def-variable)
  (push (setf (symbol-value (second def-variable))
              (crea-variable
               :nombre-de-variable (second def-variable)
               :pregunta (second (third def-variable))
               :es-objetivo (second (fourth def-variable)))
        *variables-definidas*))
```

Motor de inferencia regresivo

- **Ejemplo de análisis de definiciones de variables**

```
> (load "mir-1")
T
> (setf *variables-definidas* ())
NIL
> (analiza-def-variable
   '(defvariable x
      (pregunta "Cuanto vale x?:")
      (es-objetivo nil)))
(X*)
> *variables-definidas*
(X*)
> x
X*
> (nombre-de-variable x)
X
> (valor x)
NIL
> (pregunta x)
"Cuanto vale x?:"
> (es-objetivo x)
NIL
> (ha-sido-determinada x)
NIL
```

Motor de inferencia regresivo

- **Sintaxis de reglas**

```
<definicion-de-regla>
  ::= (regla <nombre> <antecedente> <consecuente>)
<nombre>
  ::= <simbolo>
<antecedente>
  ::= (and <condicion>*)
<condicion>
  ::= (<predicado> <variable> <valor>)
<predicado>
  ::= igual | mayor | menor | igual-numero
       | conocido | distinto
<consecuente>
  ::= <conclusion>*
<conclusion>
  ::= (<accion> <variable> <valor>)
<accion>
  ::= añade
<regla>
  ::= (<antecedente> <consecuente>)
```

- **Ejemplos de reglas**

```
(defregla regla-1
   (and
      (igual trastorno arterioesclerosis)
      (igual sintomas calambres-en-las-piernas)
      (igual presencia andando))
   (añade diagnóstico estenosis-arterial))
```

Motor de inferencia regresivo

- **Análisis de definiciones de reglas**

```
(ANALIZA-DEF-REGLA DEF-REGLA)
Valor: La lista obtenida añadiendo el nombre de la
  regla a la *BASE-DE-REGLAS*.
Efectos:
  1. Asigna como valor del nombre de la regla la
     correspondiente regla.
  2. Añade la regla a la *BASE-DE-REGLAS*.
(defun analiza-def-regla (def-regla)
  (push (setf (symbol-value (second def-regla))
              (crea-regla
               :nombre-de-regla (second def-regla)
               :antecedente (third def-regla)
               :consecuente (nthcdr 3 def-regla)))
        *base-de-reglas*))
```

Motor de inferencia regresivo

- **Ejemplo de análisis de definiciones de reglas**

```
> (load "mir-1")
T
> (setf *base-de-reglas* ())
NIL
> (analiza-def-regla
   '(defregla regla-1
      (and (igual w a)
           (igual x b))
      (añade v c)))
(REGLA-1*)
> *base-de-reglas*
(REGLA-1*)
> regla-1
REGLA-1*
> (antecedente regla-1)
(AND (IGUAL W A) (IGUAL X B))
> (consecuente regla-1)
((AÑADE V C))
```

Motor de inferencia regresivo

- **Determinación de valores de los objetivos**

```
(RESUELVE-OBJETIVOS)
Valor:  NIL
Efecto: Determina los valores de las variables objetivos.
(defun resuelve-objetivos ()
  (loop for x in *variables-definidas* do
        (when (es-objetivo x)
              (determina-valores x))))
```

- **Determinación de valores de variables**

```
(DETERMINA-VALORES VARIABLE)
Efecto: Señala la VARIABLE como determinada.
Procedimiento:
 1. Intenta deducir los valores de la variable,
    y si no lo consigue
 2. pregunta por dichos valores.
 3. Señala la variable como determinada.
(defun determina-valores (variable)
  (if (not (deduce-valores variable))     ;1
      (pregunta-valores variable))        ;2
  (hace-determinada variable))            ;3
```

Motor de inferencia regresivo

- **Ejemplo de determinación de variables**

```
> (load "mir-1")
T
> (lee-base-de-conocimiento)
Nombre de la base de conocimientos: "proposicional.bc"
T
> (determina-valores w)
Cuanto vale w?:
-> (a b c)
T
> (valor w)
(A B C)
> (ha-sido-determinada w)
T
> (determina-valores v)
Cuanto vale x?:
-> (b d)
T
> (valor v)
(C)
> (determina-valores y)
T
> (valor y)
NIL
> (ha-sido-determinada y)
T
```

Motor de inferencia regresivo

- **Deducción de valores de las variables**

```
(DEDUCE-VALORES VARIABLE)
Valor: La lista de los valores deducidos de la VARIABLE.

Procedimiento:
 1. Forma el conjunto conflicto, seleccionando las
    reglas que contiene la variable en su consecuente.
 2. Para cada una de dichas reglas cuyo antecedente se
    verifica, ejecuta su consecuente.
 3. Devuelve el valor de la variable.

(defun deduce-valores (variable)
  (let ((conjunto-de-conflicto (selecciona variable))) ;1
    (loop for regla in conjunto-de-conflicto do        ;2
      (aplica-regla regla)))
  (valor variable))                                    ;3
```

- **Formación del conjunto de conflicto**

```
(SELECCIONA VARIABLE)
Valor: La lista de las reglas de la *BASE-DE-REGLAS* que
 tienen la VARIABLE en su consecuente.
Ejemplo:
    (selecciona z)  =>  (REGLA-3* REGLA-4* REGLA-5*)
(defun selecciona (variable)
  (loop for regla in *base-de-reglas*
    when (ocurre? variable (consecuente regla))
    collect regla))
```

Motor de inferencia regresivo

- **Ocurrencia de una variable**

```
(OCURRE? VARIABLE CONSECUENTE)
Valor: T, si la VARIABLE ocurre en el CONSECUENTE de la reg
       NIL, en caso contrario.
Ejemplo:
    (consecuente regla-3)              =>  ((AÑADE Z K))
    (ocurre? z (consecuente regla-3))  =>  T
    (ocurre? x (consecuente regla-3))  =>  NIL
(defun ocurre? (variable consecuente)
  (loop for conclusion in consecuente
    thereis (eq variable (second conclusion))))
```

- **Aplicación de una regla**

```
(APLICA-REGLA REGLA)
Valor: El resultado de ejecutar el consecuente de la REGLA,
 cuando se verifica su antecedente.
Ejemplo:
    (antecedente regla-3)              =>  (IGUAL V C)
    (consecuente regla-3)              =>  ((AÑADE Z K))
    (valor z)                          =>  NIL
    (valor v)                          =>  NIL
    (setf (valor v) '(c d))            =>  (C D)
    (ha-sido-determinada v)            =>  NIL
    (setf (ha-sido-determinada v) t)   =>  T
    (aplica-regla regla-3)             =>  NIL
    (valor v)                          =>  (C D)
(defun aplica-regla (regla)
  (when (verifica-antecedente? (antecedente regla))
        (ejecuta-consecuente (consecuente regla))))
```

Motor de inferencia regresivo

- **Lisp: El procedimiento** `eval`

```
> (list '+ 2 3)
(+ 2 3)
> (eval (list '+ 2 3))
5
> '(+ 2 3)
(+ 2 3)
> (eval '(+ 2 3))
5
```

- **Verificación del antecedente**

```
(VERIFICA-ANTECEDENTE? ANTECEDENTE)
Valor: T, si se verifican las condiciones del antecedente;
       NIL, en caso contrario.
Ejemplo:
    > (lee-base-de-conocimiento)
    Nombre de la base de conocimientos: "proposicional.bc"
    T
    > (antecedente regla-1)
    (AND (IGUAL W A) (IGUAL X B))
    > (verifica-antecedente? (antecedente regla-1))
    Cuanto vale w?:
    -> (a c)
    Cuanto vale x?:
    -> (b d)
    T
(defun verifica-antecedente? (antecedente)
  (when (eval antecedente)
        t))
```

Lisp: Macros

- **El procedimiento** `defmacro`

```
(defmacro nombre (parametro-1 ... parametro-m)
  expresion-1
  ...
  expresion-n)
```

- **El procedimiento** `progn`

```
> (progn (setf a 2)
         (setf b (+ a 1))
         (+ a b))
5
```

- **Definición de la macro** `n-when`

```
(N-WHEN TEST EXP-1 ... EXP-N)
Ejemplos:
    (n-when (= 1 1) (setf a 2 b 3) (+ a b))  =>  5
    (n-when (= 1 2) (setf a 2 b 3) (+ a b))  =>  NIL
(defmacro n-when (test &rest consecuente)
  (list 'if test
            (cons 'progn consecuente)
            nil))
```

- **El procedimiento** `macroexpand`

```
> (macroexpand '(n-when (= 1 1) (setf a 2 b 3) (+ a b)))
(IF (= 1 1) (PROGN (SETF A 2 B 3) (+ A B)) NIL) ;
T
```

Lisp: Macros

- **Formas resumidas**

```
‘(a (+ 2 3) c)          =>  (A (+ 2 3) C)
‘(a ,(+ 2 3) c)         =>  (A 5 C)
‘(a ,(list ’x ’y) c)    =>  (A (X Y) C)
‘(a ,@(list ’x ’y) c)   =>  (A X Y C)
(setf b ’(x y))         =>  (X Y)
‘(a b c)                =>  (A B C)
‘(a ,b c)               =>  (A (X Y) C)
‘(a ,@b c)              =>  (A X Y C)
```

- **Redefinición de `n-when` con formas reducidas**

```
(N-WHEN TEST EXP-1 ... EXP-N)
Ejemplos:
 > (macroexpand ’(n-when-2 (= 1 1) (setf a 2 b 3) (+ a b)))
 (IF (= 1 1) (PROGN (SETF A 2 B 3) (+ A B)) NIL) ;
 T
 > (n-when (= 1 1) (setf a 2 b 3) (+ a b))
 5
(defmacro n-when-2 (test &rest consecuente)
  ‘(if ,test (progn ,@consecuente) nil))
```

Motor de inferencia regresivo

- **Predicado** igual

```
(IGUAL VARIABLE CONSTANTE)
Valor: no-NIL, si la CONSTANTE es uno de los valores de
        la VARIABLE
       NIL, en caso contrario.

(defmacro igual (variable constante)
  `(verifica-condicion? #'(lambda (x y) (member y x))
                        ',variable
                        ',constante))

Ejemplo:
 > (lee-base-de-conocimiento)
 Nombre de la base de conocimientos: "proposicional.bc"
 T
 > (setf (valor w) '(a b))
 (A B)
 > (setf (ha-sido-determinada w) t)
 T
 > (macroexpand '(igual w a))
 (VERIFICA-CONDICION? #'(LAMBDA (X Y) (MEMBER Y X)) W 'A) ;
 T
 > (igual w a)
 (A B)
```

Motor de inferencia regresivo

- **Verificación de una condición**

```
(VERIFICA-CONDICION? PREDICADO VARIABLE CONSTANTE)
Valor: no-NIL, si se verifica PREDICADO(VARIABLE,CONSTANTE)
Efecto: Si no se ha determinado el valor de la VARIABLE,
 lo determina.
Procedimiento:
 1. Si se ha determinado el valor de la variable,
 2. evalúa predicado(variable,constante);
 3. en caso contrario, determina el valor de la variable y
 4. verifica la condición.

(defun verifica-condicion? (predicado variable constante)
  (cond
   ((ha-sido-determinada variable)
    (funcall predicado (valor variable) constante))
   (t (determina-valores variable)
      (verifica-condicion? predicado variable constante))))

Ejemplo:
 > (lee-base-de-conocimiento)
 Nombre de la base de conocimientos: "proposicional.bc"
 T
 > (setf (valor w) '(a b))
 (A B)
 > (setf (ha-sido-determinada w) t)
 T
 > (verifica-condicion? #'(lambda (x y) (member y x)) w 'a)
 (A B)
```

Motor de inferencia regresivo

- **Predicado** mayor

```
(MAYOR VARIABLE CONSTANTE)
Valor: T, si el valor de la VARIABLE es mayor que la
        CONSTANTE
       NIL, en caso contrario.
Ejemplo:
    > (lee-base-de-conocimiento)
    Nombre de la base de conocimientos: "proposicional.bc"
    T
    > (setf (valor x) 7)
    7
    > x
    X*
    > (mayor x 5)
    T
(defun mayor (variable constante)
  (verifica-condicion? #'> variable constante))
```

- **Predicado** menor

```
(MENOR VARIABLE CONSTANTE)
Valor: T, si el valor de la VARIABLE es menor que la
        CONSTANTE
       NIL, en caso contrario.
(defun menor (variable constante)
  (verifica-condicion? #'< variable constante))
```

Motor de inferencia regresivo

- **Predicado** igual-numero

```
(IGUAL-NUMERO VARIABLE CONSTANTE)
Valor: T, si el valor de la VARIABLE es un número igual a
          la CONSTANTE;
       NIL, en caso contrario.
(defun igual-numero (variable constante)
  (verifica-condicion? #'= variable constante))
```

- **Predicado** conocido

```
(CONOCIDO VARIABLE)
Valor: T, si la VARIABLE tiene valor no nulo
          (i.e. distinto del inicial);
       NIL, en caso contrario.
Ejemplo:
    > (lee-base-de-conocimiento)
    Nombre de la base de conocimientos: "proposicional.bc"
    T
    > (conocido x)
    Cuanto vale x?:
    -> (desconocido)
    NIL
    > (valor x)
    NIL
(defmacro conocido (variable)
  `(not (verifica-condicion? #'eq ',variable nil)))
```

Motor de inferencia regresivo

- **Predicado** `distinto`

```
(DISTINTO VARIABLE CONSTANTE)
Valor: T, si los valores de la VARIABLE son conocidos y
          la CONSTANTE es distinta a todos los valores de
          la VARIABLE;
       NIL, en caso contrario.
Ejemplo:
    > (lee-base-de-conocimiento)
    Nombre de la base de conocimientos: "proposicional.bc"
    T
    > (conocido x)
    Cuanto vale x?:
    -> (desconocido)
    NIL
    > (distinto x a)
    NIL
    > (distinto w a)
    Cuanto vale w?:
    -> (b c)
    T
(defmacro distinto (variable constante)
  `(and (conocido ,variable)
        (not (igual ,variable ,constante))))
```

Motor de inferencia regresivo

- **Ejecución del consecuente**

```
(EJECUTA-CONSECUENTE CONSECUENTE)
Valor: NIL.
Efecto: Evalúa las acciones del CONSECUENTE.
Ejemplo:
    > (lee-base-de-conocimiento)
    Nombre de la base de conocimientos: "proposicional.bc"
    T
    > (consecuente regla-1)
    ((AÑADE V C))
    > (valor v)
    NIL
    > (ejecuta-consecuente (consecuente regla-1))
    NIL
    > (valor v)
    (C)
(defun ejecuta-consecuente (consecuente)
  (loop for accion in consecuente do
    (eval accion)))
```

Motor de inferencia regresivo

- **Acción** añade

```
(AñADE VARIABLE CONSTANTE)
Valor: La lista obtenida añadiendo la CONSTANTE a los
 valores de la VARIABLE.
Efecto: Añade la CONSTANTE a los valores de la VARIABLE.
Ejemplo:
    > (lee-base-de-conocimiento)
    Nombre de la base de conocimientos: "proposicional.bc"
    T
    > (valor v)
    NIL
    > (macroexpand-1 '(añade v c))
    (PUSH 'C (VALOR V)) ;
    T
    > (valor v)
    NIL
    > (añade v c)
    (C)
    > (valor v)
    (C)
(defmacro añade (variable constante)
  `(push ',constante (valor ,variable)))
```

Motor de inferencia regresivo

- **Pregunta de valores**

```
(PREGUNTA-VALORES VARIABLE)
Valor:
 - NIL, si no se puede preguntar los valores de la VARIABLE
 - NIL, si la respuesta es (desconocido);
 - N,   si la respuesta es (N) y N es un número;
 - L,   si la respuesta es una lista L que no empieza por
        desconocido ni por un número.
Procedimiento:
 1. Cuando la variable es preguntable,
 1.1. pregunta su valor,
 1.2. lee la respuesta y,
 1.3. salvo que empiece por desconocido,
 1.4. le asigna al valor de la variable
 1.5. el primer elemento de la respuesta, si dicho elemento
      un número o
 1.6. la respuesta, si no lo es.
 2. Cuando la variable no es preguntable, devuelve NIL.
(defun pregunta-valores (variable)
  (when (es-preguntable variable)                          ;
        (format t "~&~%~a ~%-> " (pregunta variable))      ;
        (let ((respuesta (read)))                          ;
          (unless (eq (first respuesta) 'desconocido)      ;
                  (setf (valor variable)                   ;
                        (if (numberp (first respuesta))    ;
                            (first respuesta)
                            respuesta))))))                ;
```

Motor de inferencia regresivo

- **Ejemplo de pregunta de valores**

```
> (lee-base-de-conocimiento)
Nombre de la base de conocimientos: "proposicional.bc"
T
> (pregunta-valores z)
NIL
> (pregunta-valores x)
Cuanto vale x?:
-> (desconocido)
NIL
> (valor x)
NIL
> (pregunta-valores x)
Cuanto vale x?:
-> (7)
7
> (valor x)
7
> (pregunta-valores w)
Cuanto vale w?:
-> (a b)
(A B)
> (valor w)
(A B)
```

Motor de inferencia regresivo

- **Variables preguntables**

```
(ES-PREGUNTABLE VARIABLE)
Valor: no-NIL, si se puede preguntar los valores de la VARI
       NIL, en caso contrario.
(defun es-preguntable (variable)
  (pregunta variable))
```

- **Marca de determinación**

```
(HACER-DETERMINADA VARIABLE)
Valor: T
Efecto: Marca la VARIABLE como determinada.
(defun hace-determinada (variable)
  (setf (ha-sido-determinada variable) t))
```

Motor de inferencia regresivo

- **Escritura de valores de objetivos**

```
(ESCRIBE-OBJETIVOS)
Valor: NIL
Efecto: Escribe los valores de las variables objetivos.
Ejemplo:
    > (lee-base-de-conocimiento)
    Nombre de la base de conocimientos: "proposicional.bc"
    T
    > (escribe-objetivos)
    Z = NIL
    NIL
(defun escribe-objetivos ()
  (loop for x in *variables-definidas* do
    (when (es-objetivo x)
          (format t "~&~%~a = ~a ~%"
                  (nombre-de-variable x)
                  (valor x)))))
```

Sesión con traza

```
> (load "mir-1")
T
> (trace resuelve-objetivos
         determina-valores
         deduce-valores
         selecciona
         aplica-regla
         verifica-antecedente?
         verifica-condicion?
         ejecuta-consecuente
         pregunta-valores)
(RESUELVE-OBJETIVOS ... PREGUNTA-VALORES)
> (consulta)
Nombre de la base de conocimientos: "proposicional.bc"
 1. (RESUELVE-OBJETIVOS)
  2. (DETERMINA-VALORES 'Z*)
   3. (DEDUCE-VALORES 'Z*)
    4. (SELECCIONA 'Z*)
    4. SELECCIONA ==> (REGLA-3* REGLA-4* REGLA-5*)
    4. (APLICA-REGLA 'REGLA-3*)
     5. (VERIFICA-ANTECEDENTE? '(IGUAL V C))
      6. (VERIFICA-CONDICION?
          '#<CLOSURE :LAMBDA (X Y) (MEMBER Y X)> 'V* 'C)
       7. (DETERMINA-VALORES 'V*)
        8. (DEDUCE-VALORES 'V*)
         9. (SELECCIONA 'V*)
         9. SELECCIONA ==> (REGLA-1*)
         9. (APLICA-REGLA 'REGLA-1*)
          10. (VERIFICA-ANTECEDENTE? '(AND (IGUAL W A) (IGUAL X B)))
           11. (VERIFICA-CONDICION?
                 '#<CLOSURE :LAMBDA (X Y) (MEMBER Y X)> 'W* 'A)
            12. (DETERMINA-VALORES 'W*)
```

Sesión con traza

```
             13. (DEDUCE-VALORES 'W*)
              14. (SELECCIONA 'W*)
              14. SELECCIONA ==> NIL
             13. DEDUCE-VALORES ==> NIL
             13. (PREGUNTA-VALORES 'W*)

Cuanto vale w?:
-> (a d)

             13. PREGUNTA-VALORES ==> (A D)
            12. DETERMINA-VALORES ==> T
            12. (VERIFICA-CONDICION?
                 '#<CLOSURE :LAMBDA (X Y) (MEMBER Y X)> 'W* 'A)
            12. VERIFICA-CONDICION? ==> (A D)
           11. VERIFICA-CONDICION? ==> (A D)
           11. (VERIFICA-CONDICION?
                '#<CLOSURE :LAMBDA (X Y) (MEMBER Y X)> 'X* 'B)
            12. (DETERMINA-VALORES 'X*)
             13. (DEDUCE-VALORES 'X*)
              14. (SELECCIONA 'X*)
              14. SELECCIONA ==> NIL
             13. DEDUCE-VALORES ==> NIL
             13. (PREGUNTA-VALORES 'X*)

Cuanto vale x?:
-> (b g)

             13. PREGUNTA-VALORES ==> (B G)
            12. DETERMINA-VALORES ==> T
            12. (VERIFICA-CONDICION?
                 '#<CLOSURE :LAMBDA (X Y) (MEMBER Y X)> 'X* 'B)
            12. VERIFICA-CONDICION? ==> (B G)
```

Sesión con traza

```
        11. VERIFICA-CONDICION? ==> (B G)
       10. VERIFICA-ANTECEDENTE? ==> (B G)
       10. (EJECUTA-CONSECUENTE '((AÑADE V C)))
       10. EJECUTA-CONSECUENTE ==> NIL
      9. APLICA-REGLA ==> NIL
     8. DEDUCE-VALORES ==> (C)
    7. DETERMINA-VALORES ==> T
    7. (VERIFICA-CONDICION?
        '#<CLOSURE :LAMBDA (X Y) (MEMBER Y X)> 'V* 'C)
    7. VERIFICA-CONDICION? ==> (C)
   6. VERIFICA-CONDICION? ==> (C)
  5. VERIFICA-ANTECEDENTE? ==> (C)
  5. (EJECUTA-CONSECUENTE '((AÑADE Z K)))
  5. EJECUTA-CONSECUENTE ==> NIL
4. APLICA-REGLA ==> NIL
4. (APLICA-REGLA 'REGLA-4*)
 5. (VERIFICA-ANTECEDENTE? '(AND (IGUAL X J) (IGUAL Y E)))
  6. (VERIFICA-CONDICION?
      '#<CLOSURE :LAMBDA (X Y) (MEMBER Y X)> 'X* 'J)
  6. VERIFICA-CONDICION? ==> NIL
 5. VERIFICA-ANTECEDENTE? ==> NIL
4. APLICA-REGLA ==> NIL
4. (APLICA-REGLA 'REGLA-5*)
 5. (VERIFICA-ANTECEDENTE? '(AND (IGUAL U F) (IGUAL X G)))
  6. (VERIFICA-CONDICION?
      '#<CLOSURE :LAMBDA (X Y) (MEMBER Y X)> 'U* 'F)
   7. (DETERMINA-VALORES 'U*)
    8. (DEDUCE-VALORES 'U*)
     9. (SELECCIONA 'U*)
     9. SELECCIONA ==> NIL
    8. DEDUCE-VALORES ==> NIL
    8. (PREGUNTA-VALORES 'U*)
```

Sesión con traza

```
Cuanto vale u?:
-> (f)

        8. PREGUNTA-VALORES ==> (F)
       7. DETERMINA-VALORES ==> T
       7. (VERIFICA-CONDICION?
           '#<CLOSURE :LAMBDA (X Y) (MEMBER Y X)> 'U* 'F)
       7. VERIFICA-CONDICION? ==> (F)
      6. VERIFICA-CONDICION? ==> (F)
      6. (VERIFICA-CONDICION?
          '#<CLOSURE :LAMBDA (X Y) (MEMBER Y X)> 'X* 'G)
      6. VERIFICA-CONDICION? ==> (G)
     5. VERIFICA-ANTECEDENTE? ==> (G)
     5. (EJECUTA-CONSECUENTE '((AÑADE Z I)))
     5. EJECUTA-CONSECUENTE ==> NIL
    4. APLICA-REGLA ==> NIL
   3. DEDUCE-VALORES ==> (I K)
  2. DETERMINA-VALORES ==> T
 1. RESUELVE-OBJETIVOS ==> NIL

Z = (I K)
```

Indetermismos

- **Indeterminismo de primera especie**
  - **Orden de aplicación de reglas seleccionadas**
  - **Estrategia de resolución de conflictos**
  - `selecciona`
- **Ejemplos de estrategia de resolución de conflictos**
  - **Por el orden de especificación**
  - **Por el número de condiciones**
  - **Por el número de condiciones no determinadas**
- **Indeterminismo de segunda especie**
  - **Orden de evaluación de las acciones**
- **Posibles mejoras**
- **Encadenamiento hacia adelante**