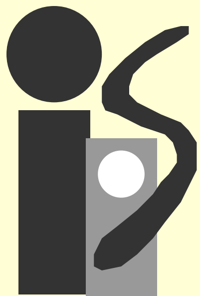
Programación en Lenguajes basados en reglas. Aspectos metodológicos II.



J.A. Bañares Bañares

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

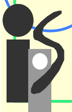
Copyright © 1997 José Ángel Bañares

Última revisión: Octubre., 2002

© José Angel Bañares

Objetivo

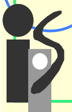
- **Objetivo:**
  - Presentación de técnicas útiles en el diseño, escritura y depurado de programas en sistemas de producción tipo OPS5 o CLIPS
- **Método:**
  - Como y dónde representar el conocimiento
    - Tipos de Conocimiento:
      Resolución del problema, control y datos
  - Desarrollo de programas con Sistemas de producción
    - Elaboración, Refinamiento, Generalización
  - Control
    - Interacción entre reglas, Cambios en la memoria de trabajo
    - *Forward* y *backward chaining*
    - Fases y hechos de control.
    - Módulos



© José Angel Bañares

1. El lugar del conocimiento

- **Los primeros trabajos en SP se inspiran en un modelo de la memoria humana:**
  - Memoria a largo plazo => Representada con de reglas estáticas
  - Memoria a corto plazo => Representada con la memoria de trabajo
- **Regla general:**
  - Cada principio del dominio de conocimiento en una regla
  - Resultados intermedios, datos y *flags* de control en la MT



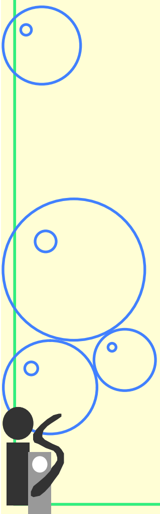
© José Angel Bañares

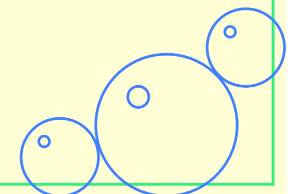
Conocimiento sobre la solución en MP

1 **Conocimiento sobre la solución del problema**

- Lo más natural es que aparezca en la **MP**
  - Ejemplo: Los sistemas expertos de diagnóstico constan de grandes conjuntos de reglas similares a la que sigue, aunque más complejas:

```
;;; Problemas al cocer pan
(deftemplate problema
   (slot manifestacion))          ; ¿Qué fué mal?
(deftemplate hipotesis
   (slot causa))                  ; ¿Cual es la causa?


(defrule Pan_no_sube "Causas de una mala coccion"
  (problem (manifestacion no_sube))
  =>
  (assert (hipotesis (causa levadura_inactiva))
  (assert (hipotesis (causa temperatura_agua_incorrecta))
  (assert (hipotesis (causa azucar_olvidada)))
```



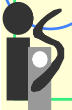


© José Angel Bañares

Conocimiento sobre la solución en MT

- Pero a veces el conocimiento de la solución del problema, pueden estar en la **MT**.
  - Una manifestación puede llevar a muchas hipótesis. Podemos tener una tabla que enlaza una manifestación con una hipótesis

```
(deftemplate problema  (slot manifestacion))
(deftemplate hipotesis (slot causa))
(deftemplate tabla_diagnosticos  (slot manifestación)
                                 (slot causa_posible))
(defrule Sugiere_hipotesis "Causas de la manifestación"
  (problema (manifestacion ?manifestacion))
  (tabla_diagnosticos (manifestacion ?manifestacion)
       (causa_posible ?hipotesis))
 =>
  (assert (hipotesis (causa ?hipotesis))))
(assert (tabla_diagnosticos  (manifestacion no_sube)
       (causa_posible (levadura_inactiva))
(assert (tabla_diagnosticos (manifestacion no_sube)
       (causa_posible temperatura_agua_incorrecta))
(assert (tabla_diagnosticos (manifestacion no_sube)
       (causa_posible (azucar_olvidada))
```



© José Angel Bañares

Conocimiento de control en MP

2 Conocimiento de control

- Se utiliza para dirigir la secuencia en que deben realizarse los pasos de la solución del problema. Razones:
  - Para considerar primero soluciones más prometedoras
  - Hacer el proceso de solución más comprensible para un observador
  - Forzar una secuencia de pasos
- Ejemplo de conocimiento de control en la **MP**:

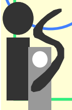
```
(deftemplate fase (slot descripcion) (slot estado))
(defrule PasoDe_desarrolloA_Evaluando
  (fase (descripcion Desarrollo) (estado terminado))
  (not (fase (descripcion Evaluando)))
  =>
  (assert (fase (descripcion Evaluando)(estado activo))))
(defrule PasoDe_EvaluandoA_Venta
  (fase (descripcion Evaluando) (estado terminado))
  (not (fase (descripcion Venta)))
  =>
  (assert (fase (descripcion Venta) (estado activo))))
```

© José Angel Bañares

Conocimiento de control en MT

- En un sentido amplio, los datos y resultados intermedios sirven como conocimiento de control, pero además en la **MT** puede haber elementos cuyo fin exclusivo es el control.
- Ejemplo control en la **MT**:

```
(deftemplate secuencia_fases   (slot fase_1)
                               (slot fase_2))
(deftemplate fase (slot descripcion) (slot estado))

(defrule cambia_fase
   (fase (desrcripcion ?fase1) (estado terminado))
   (secuencia_fases (fase_1 ?fase1) (fase_2 ?fase2))
   (not (fase (descripcion ?fase2)))
  =>
   (assert (fase (descripcion ?fase2) (estado activo)))

(assert (secuencia_fases  (fase_1 Desarrollo)
                           (fase_2 Evaluacion)))
(assert (secuencia_fases  (fase_1 Evaluacion)
                          (fase_2 Venta)))
```



© José Angel Bañares

Conocimiento de datos en MP

3 **Conocimiento sobre datos**

- Lo normal es guardar esta información en MT. Se puede guardar en MP si la actualización es poco frecuente:
  - Ejemplo de conocimiento de datos en la **MP**:

```
(deftemplate avion (slot modelo) (slot numero)
        (slot mision) (slot capitan))

(defrule Numero_Airbus
   ?avion <- (avion (modelo Airbus_20)
                    (numero nil))
  =>
  (modify ?avion (numero AB-20))
```

  - Si la compañía aérea tiene cientos de modelos debe haber cientos de reglas.

© José Angel Bañares

Conocimiento de datos en MT

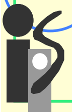
- Lo normal es guardar esta información en MT.
  - Ejemplo de conocimiento de datos en la **MT**:

```
(deftemplate avion (slot modelo) (slot numero)
        (slot mision) (slot capitan))

(deftemplate tabla_numeros
         (slot modelo) (slot numero))

(defrule asigna_numero
?avion <- (avion (modelo ?modelo) (numero nil))
?tabla <- (tabla_numeros (modelo ?modelo)
                         (numero ?numero))
  =>
  (modify ?avion (numero ?numero))

(assert (tabla_numeros (modelo Airbus_20)
                       (numero AB-20))
(assert (tabla_numeros (modelo Boeing)
                       (numero B-7))
```

© José Angel Bañares

Localización del Conocimiento. Resumen

- **Conocimiento de resolución del problema:**
  - En la MP.
  - Si se coloca en la MT
    - Desventaja: Se sacrifica la velocidad al emular otro SP.
    - Ventajas: El conocimiento de resolución del problema puede ser fácilmente modificado. Además el sistema puede razonar sobre sobre el conocimiento de resolución (metaconocimiento).
- **Conocimiento de Datos**
  - En la MT.
- **Conocimiento del Control**
  - Más difícil dar una norma
  - Control a largo plazo en MP
  - Control efímero en MT.
    - La estrategia MEA facilita el uso de la MT para control



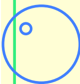
© José Angel Bañares

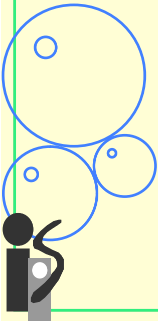
Sistemas híbridos

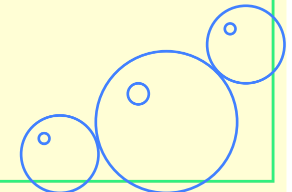
- **Con frecuencia, un único modelo de representación no es adecuado para un dominio**
  - Solución:Sistemas híbridos (CLIPS, KEE, Loops) que integran
    - Programación procedural/funcional

      Cálculos, acceso a bases de datos externas, etc.
    - Programación basada en reglas
    - Programación basada en *frames*/objetos

      Cuando se requiere un conocimiento más profundo de los principios del modelo



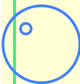


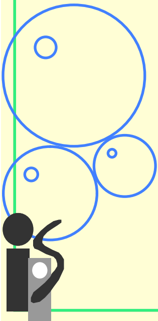


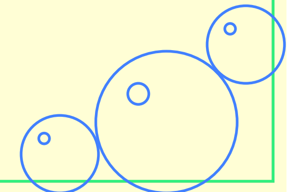
© José Angel Bañares

2. Desarrollo de SP

- **La posibilidad de crear especificaciones ejecutables como resultado del análisis sugiere un desarrollo iterativo**
  - El analista comienza con una definición del problema e interaccionando con el experto construye un conjunto de escenarios del problema.
  - Según se desarrolla el sistema debe aprender a percibir diferencias y donde prestar atención.
  - Los sistemas de producción crecen mediante procesos de diferenciación y expansión
    - **Expansión**: reconoce nuevos patrones de datos
    - **Diferenciación**: Distinción entre patrones existentes





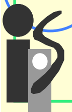


© José Angel Bañares

2.1 Desarrollo por elaboración

- **La nueva regla detecta en su LHS patrones que son ignorados completamente por las reglas anteriores.**
  - Ejemplo: Consejero ocupacional automático. A partir de un cuestionario aconseja profesión.

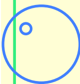
```
(deftemplate interes (slot area) (slot grado))

(defrule biologo
  (interes (area academico) (grado ?grado1&:(> ?grado1 60)))
  (interes (area naturaleza) (grado ?grado2&:(> ?grado2 70)))
  =>
  (assert (profesion biologo)))

(defrule graduado-social-enfermero-psicologo
  (interes (area academico)
          (grado ?grado1&:(and (> ?grado1 20) (< ?grado1 60))))
  (interes (area gente) (grado ?grado2&:(> ?grado2 70)))
  (interes (area asistente) (grado ?grado3&:(> ?grado3 80)))
  =>
  (assert (profesion graduado-social) (profesion enfermero)
          (profesion psicologo)))
```

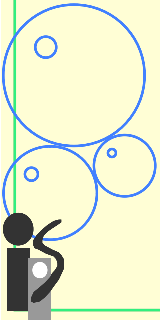


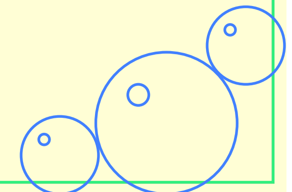
© José Angel Bañares

2.2 Desarrollo por refinamiento

- **Se incluyen distinciones no consideradas anteriormente**
  - Dos situaciones:
    - 1.**Casos excepcionales**: Una regla que considera un caso general y reglas que consideran excepciones.
    - 2. **Fisión**: Subdividir casos considerados anteriormente.
  - Los **Sistema de Producción** que se desarrollan de esta forma van ganando resolución más que ampliar el ámbito del problema:
    - 1. Parten de un conjunto inicial de reglas que cubren todos los casos en unas pocas categorías generales
    - 2. Según se refina el sistema las reglas son cada vez más discriminantes







© José Angel Bañares

Casos especiales y Excepciones

- 1. Casos excepcionales:
  - Ejemplo: Diagnóstico psiquiátrico

    Síntomas: insomnio, ansiedad, migrañas Diagnostico: Neurosis

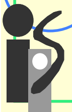
    Excepción: Adicción a la cafeína presenta los mismo síntomas
- CLIPS:
  - Reglas más específicas tienen preferencia
  - La regla que identifica un caso excepcional debe deshabilitar la regla general
  - Ejemplo:

    Año no es divisible por 4 => Febrero tiene 28 días

    Año es divisible por 4 pero no por 100=> Febrero tiene 29 días

    Año divisible por 100 pero no por 400 => Febrero tiene 28 días

    Año es divisible por 400 => Febrero tiene 29 días



© José Angel Bañares

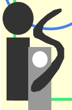
Ejemplo casos especiales

```
(deftemplate agno (slot numero) (slot dias)
                  (slot mod-4) (slot mod-100) (slot mod-400))

(defrule toma-modulo
  ?agno <- (agno (numero ?numero) (mod-4 nil))
  =>
   (modify ?agno (mod-4 =(mod ?numero 4))
   (mod-100 =(mod ?numero 100)) (mod-400 =(mod ?numero 400))))

(defrule agno-divisible-4
  ?agno <- (agno (dias nil) (mod-4 0))
  => (modify ?agno (dias 366)))

(defrule agno-divisible-100
  ?agno <- (agno (dias nil) (mod-4 0) (mod-100 0))
  => (modify ?agno (dias 365)))

(defrule agno-divisible-400
  ?agno <- (agno (dias nil) (mod-4 0) (mod-100 0) (mod-400 0))
  => (modify ?agno (dias 366)))

(assert (agno (numero 1996)))
```

© José Angel Bañares

Ejemplo casos especiales

```
CLIPS> (assert (agno (numero 2000)))
==> f-1      (agno (numero 2000) (dias nil) (mod-4 nil)
(mod-100 nil) (mod-400 nil))
==> Activation 0      toma-modulo: f-1
<Fact-1>
CLIPS> (run)
FIRE    1 toma-modulo: f-1
<== f-1      (agno (numero 2000) (dias nil)
                (mod-4 nil) (mod-100 nil) (mod-400 nil))
==> f-2      (agno (numero 2000) (dias nil)
                (mod-4 0) (mod-100 0) (mod-400 0))
==> Activation 0      agno-divisible-4: f-2
==> Activation 0      agno-divisible-100: f-2
==> Activation 0      agno-divisible-400: f-2
FIRE    2 agno-divisible-400: f-2
<== f-2      (agno (numero 2000) (dias nil)
                 (mod-4 0) (mod-100 0) (mod-400 0))
<== Activation 0      agno-divisible-100: f-2
<== Activation 0      agno-divisible-4: f-2
==> f-3      (agno (numero 2000) (dias 366)
                (mod-4 0) (mod-100 0) (mod-400 0))
```

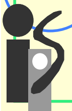


© José Angel Bañares

Fisión

- 2. Fisión: Una regla es reemplazada (no aumentada) por reglas que son casos especiales de la original.

```
(defrule graduado-social-psicologo
  (interes (area academico)
         (grado ?grado1&:(and (> ?grado1 20) (< ?grado1 60))))
  (interes (area gente) (grado ?grado2&:(> ?grado2 70)))
  (interes (area asistente) (grado ?grado3&:(> ?grado3 80)))
  (interes (area tecnica) (grado ?grado4&:(< ?grado4 70)))
  =>
  (assert (profesion graduado-social) (profesion psicologo)))

(defrule enfermero
  (interes (area academico)
         (grado ?grado1&:(and (> ?grado1 20) (< ?grado1 60))))
  (interes (area gente) (grado ?grado2&:(> ?grado2 70)))
  (interes (area asistente) (grado ?grado3&:(> ?grado3 80)))
  (interes (area tecnica) (grado ?grado4&:(> ?grado4 40)))
  =>
  (assert (profesion graduado-social) (profesion enfermero)
          (profesion psicologo)))
```



© José Angel Bañares

2.3 Desarrollo por Generalización

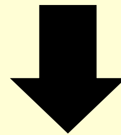
- **La generalización permite reducir la memoria de producción:**
  - Se **mantienen la cantidad de conocimiento constante**
  - La Generalización refina la representación del conocimiento por simplicidad, elegancia y eficiencia.
    - Si una regla que es reconoce un caso especial tiene la misma RHS que la regla general eliminarla
  - Ejemplo: Si dos reglas tienen RHS equivalentes y sus LHS pueden ser unificadas.
    - Las tres reglas siguientes que asignan Código postal sólo difieren en el nombre de la calle

```
(deftemplate direccion
         (slot numero)
         (multislot calle)
         (slot ciudad)
         (slot estado)
         (slot CP))
```

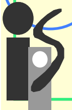
© José Angel Bañares

Ejemplo Generalización

```
(defrule AvdaCataluna-50004
  ?direccion <- (direccion (numero ?numero&: (> ?numero 300))
      (calle AvdaCataluna) (ciudad Zaragoza) (provincia Zaragoza))
  =>
  (modify ?direccion (CP 50007)))

(defrule PsoCuellar-50004
  ?direccion <- (direccion (numero ?numero&: (> ?numero 300))
      (calle PsoCuellar) (ciudad Zaragoza) (provincia Zaragoza))
  =>
  (modify ?direccion (CP 50004)))
```



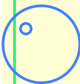
```
(defrule Gomez-Abellaneda-Zambrano-50004
  ?direccion <- (direccion (numero ?numero&: (> ?numero 300))
      (calle PsoCuellar|AvdaCataluna) (ciudad Zaragoza)
      (provincia Zaragoza))
  =>
  (modify ?direccion (CP 50004)))
```

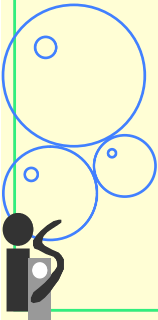


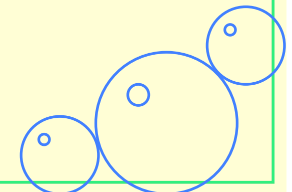
© José Angel Bañares

3. Introducción al Control en lenguajes tipo OPS5

- **Los sistemas de reconocimiento de patrones con encadenamiento hacia adelante permiten cierto control sobre la secuencia de disparos de reglas**
  - **Para explotar la potencia del lenguaje el control debería ser dirigido por los datos**
  - Razones para imponer control
    - OPS5 no permite partir la **MP** ni la **MT**
    - La solución directa es ineficiente
    - Ofrecer un interfaz humano comprensible
    - Emular otro SP de arquitectura diferente







© José Angel Bañares

3.1.1 Forward Chainig

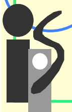
- **Ejemplo: Rellenar declaración de la renta**
  - Dos fases (que se pueden solapar)
    - Colocar datos en la MT

      Cadena de reglas que leen los datos
    - Combinar valores de acuerdo a las fórmulas

      Reglas que disparan en cuanto tienen los datos requeridos

      No afectan a la cadena de reglas que obtienen datos de entrada

```
;;; Ejemplo Declaración renta, encadenamiento progresivo

(deftemplate cantidad (slot nombre) ;Nombre de concepto fiscal
                      (slot valor)) ;Valor numerico
(deffacts tipos-impositivos    ; Tipos impositivos en funcion
 (tabla 1 0.3)                 ; del estado civil
 (tabla 2 0.22)
 (tabla 3 0.25)
 (tabla 4 0.22)
 (tabla 5 0.15))
```



© José Angel Bañares

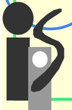
Fase entrada datos (forward chainig)

```
;;; Fase 1: Reglas para preguntar datos

; Pregunta estado civil del usuario

(defrule entrada-estado
   (initial-fact)
   (not (cantidad (nombre estado)))
  =>
   (printout t crlf "Introduce estado civil: ")
   (printout t crlf "1: Soltero ")
   (printout t crlf "2: Casado, declaracion conjunta ")
   (printout t crlf "3: Casado, declaracion separada ")
   (printout t crlf "4: Cabeza de familia ")
   (printout t crlf "5: Viudo(a) con hijos a su cargo " crlf)
   (assert (cantidad (nombre estado) (valor (read)))))
```

© José Angel Bañares

Entrada de datos (forward chainig)

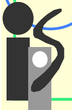
```
; Pregunta tipos de desgravaciones
(defrule desgravaciones
   (cantidad (nombre estado))
   (not (cantidad (nombre desgravacion)))
  =>
   (printout t crlf "Para las siguientes cuestiones si tu ")
   (printout t crlf "respuesta es SI introduce 1, y si es ")
   (printout t crlf "NO introduce 0")
   (printout t crlf "Tiene 65 agnos o mas: ")
   (bind ?eda-65 (read))
   (printout t crlf "Tiene alguna minusvalia: ")
   (bind ?minusvalia (read))
   (printout t crlf "Tiene su esposa 65 o mas agnos: ")
   (bind ?edad-esposa-65 (read))
   (printout t crlf "Tienen su esposa o hijos alguna minusvalia: ")
   (bind ?minusvalia-familiares (read))
   (printout t crlf "Para las siguientes preguntas introduzca ")
   (printout t crlf "el numero que corresponda.")
   (printout t crlf "Cuantos hijos viven con usted: ")
   (bind ?hijos (read))
   (printout t crlf "Cantidades aportadas a Vivienda:")
   (bind ?vivienda (read))
   (assert (cantidad (nombre desgravacion)
       (valor =(+ 1 ?eda-65 ?minusvalia ?edad-esposa-65
                    ?minusvalia-familiares ?hijos))))
   (assert (cantidad (nombre vivienda) (valor ?vivienda))))
```



© José Angel Bañares

Entrada de datos (forward chainig)

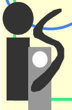
```
(defrule salario
    (cantidad (nombre desgravacion))
    (not (cantidad (nombre salario)))
  =>
    (printout t crlf "Introduce salario: ")
    (assert (cantidad (nombre salario) (valor (read)))))

(defrule retenciones
    (cantidad (nombre salario))
    (not (cantidad (nombre retencion)))
     =>
    (printout t crlf "Introduce retencion: ")
    (assert (cantidad (nombre retencion) (valor (read)))))
```



© José Angel Bañares

Cálculos (fordward chainig)

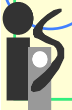
```
;;; Fase 2: Reglas para calculos
(defrule tipo-impositivo
   (cantidad (nombre estado) (valor ?estado))
   (tabla ?estado ?tipo)
   (not (cantidad (nombre tipo)))
  =>
   (assert (cantidad (nombre tipo) (valor ?tipo))))

(defrule calcula-desgravaciones
   (cantidad (nombre desgravacion) (valor ?num))
   (cantidad (nombre vivienda) (valor ?aportado-vivienda))
  =>
   (assert (cantidad (nombre total-desgravaciones)
       (valor =(+ (* 0.15 ?aportado-vivienda) (* 25000 ?num))))))

(defrule calcula-neto
  (cantidad (nombre total-desgravaciones) (valor ?desgravaciones))
  (cantidad (nombre salario) (valor ?salario))
 =>
 (assert (cantidad (nombre neto)
         (valor  =(- ?salario ?desgravaciones)))))
```



© José Angel Bañares

Cálculos (forward chainig)

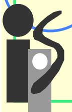
```
(defrule deshabilita-neto-negativo
  ?c <- (cantidad (nombre neto)
                  (valor  ?valor&:(< ?valor 0)))
  =>
  (modify ?c (valor 0)))

(defrule calcula-balance
  (cantidad (nombre neto) (valor ?neto&:(> ?neto 0)))
  (cantidad (nombre retencion) (valor ?retencion))
  (cantidad (nombre tipo) (valor ?tipo))
 =>
 (assert (cantidad (nombre balance)
         (valor =(- (* ?neto ?tipo) ?retencion)))))

(defrule escribe-impuesto
  (cantidad (nombre balance) (valor ?valor))
 =>
    (printout t crlf "El resultado es:" ?valor crlf))
```



© José Angel Bañares

3.1.2 Backward Chaining

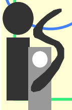
- Tres grupos de reglas
  - Reglas de subobjetivos
    - Si el objetivo no se ha alcanzado el objetivo divide este en subobjetivos mas simples
  - Reglas para manejar los objetivos inmediatamente resolubles
  - Reglas de control
    - Implementación del encadenamiento hacia atrás.
    - Una vez alcanzado los subobjetivos se soluciona el objetivo fusionando la solución de los subojetivos
- Puede tener reglas que hacen el sistema más sofisticado
  - Por ejemplo, el usuario quiere saber porqué se hace una pregunta y que responda en base a los objetivos en la memoria de trabajo.



© José Angel Bañares

Objetivo (backward chainig)

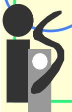
```
;;; Ejemplo Declaración renta, encadenamiento regresivo

(deftemplate objetivo
   (slot nombre)    ; El objetivo que se persigue
   (slot padre)     ; El objetivo cuyas permisas dieron este
   (slot valor))    ; Valor entero cuando esta satisfecho,
                  ; sino nil

(deffacts tipos-impositivos
 (tabla 1 0.3)
 (tabla 2 0.22)
 (tabla 3 0.25)
 (tabla 4 0.22)
 (tabla 5 0.15)
 (objetivo (nombre declaracion-hecha)))

(defrule escribe-resultado
   (objetivo (nombre declaracion-hecha) (valor ?valor&~nil))
  =>
  (printout t crlf "Resultado: " ?valor crlf))
```



© José Angel Bañares

Subobjetivos y control

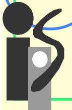
```
;;; Reglas de  división en subojetivos y control
;;; Objetivo se divide en subobjetivos, y subobjetivos se fusionan

(defrule divide:declaracion-hecha
   (objetivo (nombre declaracion-hecha) (valor nil))
  =>
   (assert (objetivo (nombre retencion)
                     (padre declaracion-hecha)))
   (assert (objetivo (nombre ingresos-netos)
                     (padre declaracion-hecha)))
   (assert (objetivo (nombre estado) (padre declaracion-hecha))))

(defrule fusiona:declaracion-hecha
   ?objetivo <- (objetivo (nombre declaracion-hecha) (valor nil))
   (objetivo (nombre ingresos-netos)
             (valor ?neto&:(numberp ?neto)))
   (test (> ?neto 0))
   (objetivo (nombre estado) (valor ?estado))
   (tabla ?estado ?tipo)
   (objetivo (nombre retencion)
             (valor ?retencion&:(numberp ?retencion)))
  =>
   (modify ?objetivo (valor =(- (* ?neto ?tipo) ?retencion))))
```



© José Angel Bañares

Subobjetivos y control

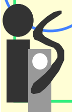
```
(defrule divide:ingresos-netos
   (objetivo (nombre ingresos-netos) (valor nil))
  =>
   (assert (objetivo (nombre salario) (padre ingresos-netos)))
   (assert (objetivo (nombre desgravacion) (padre ingresos-netos))))

(defrule fusiona:ingresos-netos
   ?objetivo <- (objetivo (nombre ingresos-netos) (valor nil))
   (objetivo (nombre desgravacion) (valor ?desgravacion&:(numberp ?
  desgravacion)))
   (objetivo (nombre salario) (valor ?salario&:(numberp ?salario)))
  =>
   (modify ?objetivo (nombre ingresos-netos)
           (valor =(- ?salario ?desgravacion))))

(defrule deshabilita-neto-negativo ; Demon
  ?obj <- (objetivo (nombre ingresos-netos) (valor  ?valor&~nil))
  (test (< ?valor 0))
  =>
  (modify ?obj (valor 0)))
```



© José Angel Bañares

Subobjetivos y control

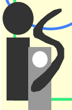
```
(defrule divide:desgravaciones
   (objetivo (nombre desgravacion) (valor nil))
  =>
   (assert (objetivo (nombre vivienda)
                     (padre desgravacion)))
   (assert (objetivo (nombre exenciones)
                     (padre desgravacion))))

(defrule fusiona:desgravaciones
   ?objetivo <- (objetivo (nombre desgravacion)
                          (valor nil))
   (objetivo (nombre exenciones) (valor ?num&~nil))
   (objetivo (nombre vivienda)
             (valor ?aportado-vivienda&~nil))
  =>
   (modify ?objetivo
           (valor =(+ (* 0.15 ?aportado-vivienda)
           (* 25000 ?num)))))
```



© José Angel Bañares

Objetivos satisfacibles

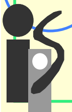
```
;;; Objetivos que se pueden satisfacer inmediatamente

(defrule entrada-estado
   ?objetivo <- (objetivo (nombre estado) (valor nil))
  =>
   (printout t crlf "Introduce estado civil: ")
   (printout t crlf "1: Soltero ")
   (printout t crlf "2: Casado, declaracion conjunta ")
   (printout t crlf "3: Casado, declaracion separada ")
   (printout t crlf "4: Cabeza de familia ")
   (printout t crlf "5: Viudo(a) con hijos a su cargo" crlf)
   (modify ?objetivo (valor (read))))

 (defrule entrada-retencion
   ?objetivo <- (objetivo (nombre retencion) (valor nil))
  =>
   (printout t crlf "Introduce retencion: ")
   (modify ?objetivo (valor (read))))
```



© José Angel Bañares

Objetivos satisfacibles

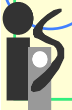
```
(defrule entrada-exenciones
    ?objetivo <- (objetivo (nombre exenciones) (valor nil))
  =>
   (printout t crlf "Para las siguientes cuestiones si tu ")
   (printout t crlf "respuesta es SI introduce 1, y si es ")
   (printout t crlf "NO introduce 0")
   (printout t crlf "Tiene 65 agnos o mas: ")
   (bind ?eda-65 (read))
   (printout t crlf "Tiene alguna minusvalia: ")
   (bind ?minusvalia (read))
   (printout t crlf "Tiene su esposa 65 o mas agnos: ")
   (bind ?edad-esposa-65 (read))
   (printout t crlf "Tienen su esposa o hijos alguna minusvalia: ")
   (bind ?minusvalia-familiares (read))
   (printout t crlf "Para las siguientes preguntas introduzca ")
   (printout t crlf "el numero que corresponda.")
   (printout t crlf "Cuantos hijos viven con usted: ")
   (bind ?hijos (read))
   (modify ?objetivo (valor =(+ 1 ?eda-65 ?minusvalia
                       ?edad-esposa-65 ?minusvalia-familiares
                       ?hijos))))
```



© José Angel Bañares

Objetivos satisfacibles y Explicación

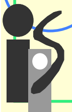
```
(defrule entrada-vivienda
   ?objetivo <- (objetivo (nombre vivienda) (valor nil))
  =>
   (printout t crlf "Cantidades aportadas a Vivienda:")
   (modify ?objetivo (valor (read))))

(defrule entrada-salario
   ?objetivo <- (objetivo (nombre salario) (valor nil))
  =>
   (printout t crlf "Salario:")
   (modify ?objetivo (valor (read))))

;;; Módulo explicación

(defrule explica
  (porque ?objetivo)
  (objetivo (nombre ?objetivo) (padre ?padre))
 =>
  (printout t crlf "Estoy intentando determinar "
                ?objetivo " porque"  crlf
            "es necesario para obtener " ?padre crlf))
```



© José Angel Bañares

3.2 Fases y Hechos de control

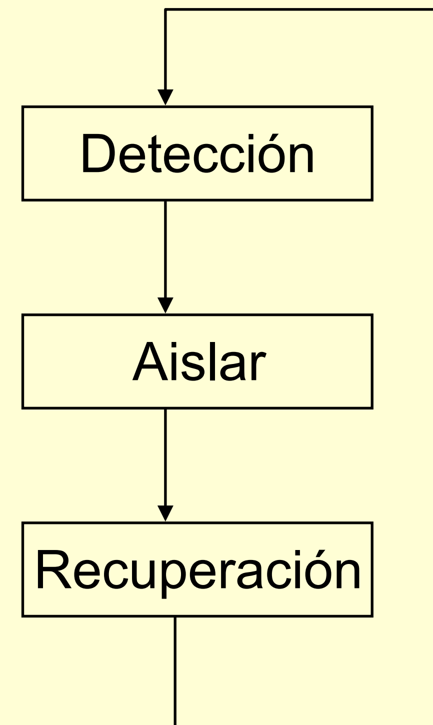
- **Control empotrado en las reglas**
  - Ejemplo Juego del Nim: Hecho que indica el turno en un juego entre el computador y un usuario
    - Elección del jugador que empieza
    - Elección del número de piezas
    - Turno del humano
    - Turno de la computadora
  - Hechos de control:
    - `(fase elige-jugador)`
    - `(fase -elige-numero-de-piezas)`
    - `(turno h)`
    - `(turno c)`
  - **Desventaja**:
    - Se mezcla el conocimiento del dominio con la estructura de control => Mantenimiento y desarrollo más costoso.
    - Dificultad para precisar la conclusión de una fase

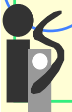


© José Angel Bañares

Técnicas de control

- **Ejemplo:**
  - Problema de un SE que en un dispositivo electrónico
    - Detecta Fallos
    - Aísla causas
    - Recupera si es posible



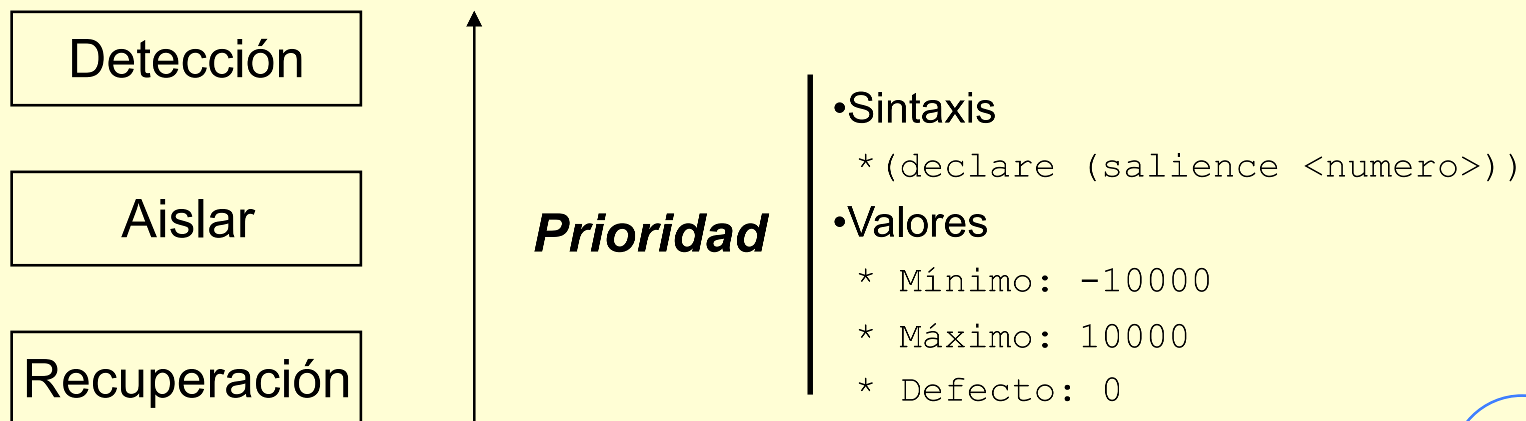


© José Angel Bañares

Aproximaciones 1 y 2

- **Aproximaciones al control usando fases y prioridades**
  - 1. **Insertar control en las reglas**
    - Cada grupo de reglas tienen un patrón que indican en que fase se aplican
    - Una regla de cada fase se aplica cuando no quedan más reglas aplicables.
  - 2. **Utilizar prioridades**
    - No se garantiza el orden de ejecución correcto

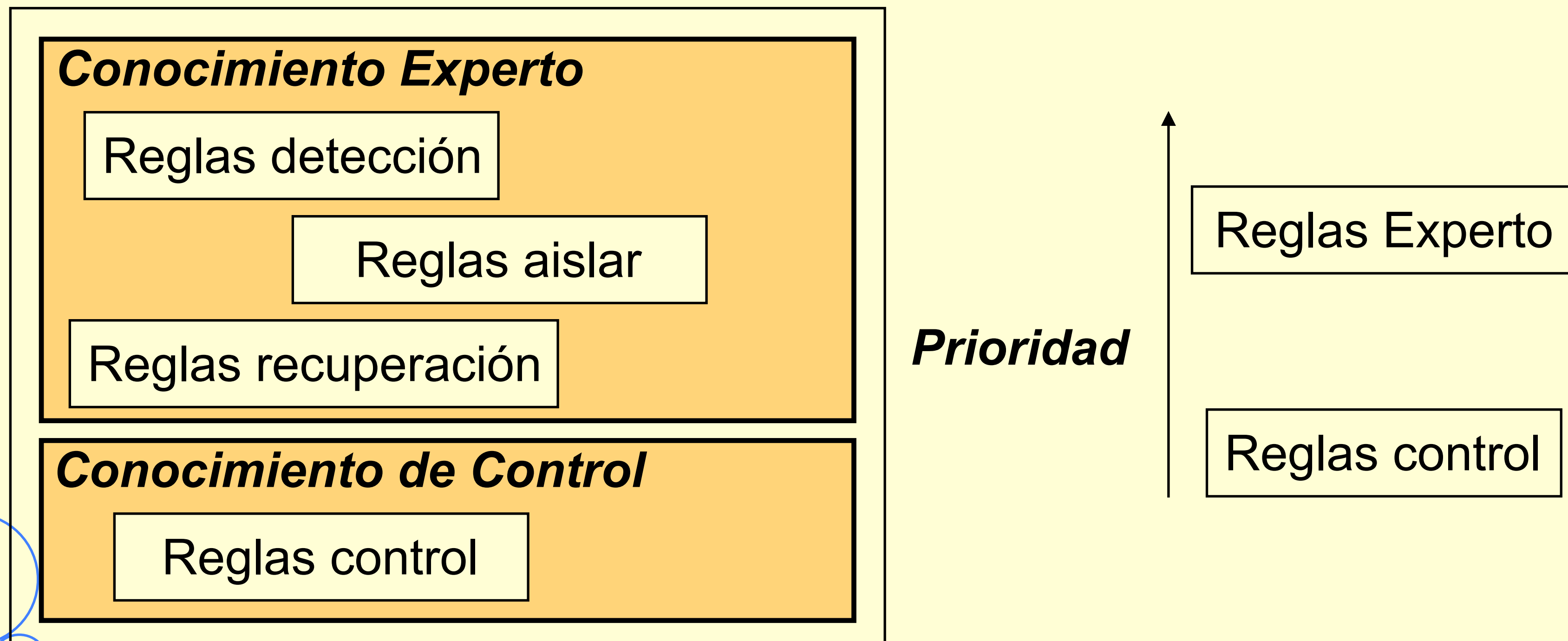
Las reglas de detección siempre tienen mayor prioridad

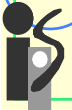


© José Angel Bañares

Aproximación 3 (cont.)

- 3. Insertar patrón de fase en reglas de conocimiento y separar reglas de control que cambian de fase
  - **Reglas de control con menor prioridad**





© José Angel Bañares

Aproximación 3 (cont.)

- Reglas de control y del dominio

```
;;; Reglas de control
(defrule deteccion-a-aislar
    (declare (salience -10))
    ?fase <- (fase deteccion)
  =>
    (retract ?fase)
    (assert (fase aislar)))

(defrule aislar-a-recuperacion
    (declare (salience -10))
    ?fase <- (fase aislar)
  =>
    (retract ?fase)
    (assert (fase recuperacion)))

(defrule recuperar-a-deteccion
    (declare (salience -10))
    ?fase <- (fase recuperacion)
  =>
    (retract ?fase)
    (assert (fase deteccion)))
```

```
;;; Ejemplo regla recuperacion
(defrule recupera
  (fase recuperacion)
  ...)
```

```
CLIPS> (watch rules)
CLIPS> (reset)
CLIPS>  (assert (fase deteccion))
<Fact-1>
CLIPS> (run 5)
FIRE    1 deteccion-a-aislar: f-1
FIRE    2 aislar-a-recuperacion: f-2
FIRE    3 recuperar-a-deteccion: f-3
FIRE    4 deteccion-a-aislar: f-4
FIRE    5 aislar-a-recuperacion: f-5
```

© José Angel Bañares

Aproximación 3 (cont.)

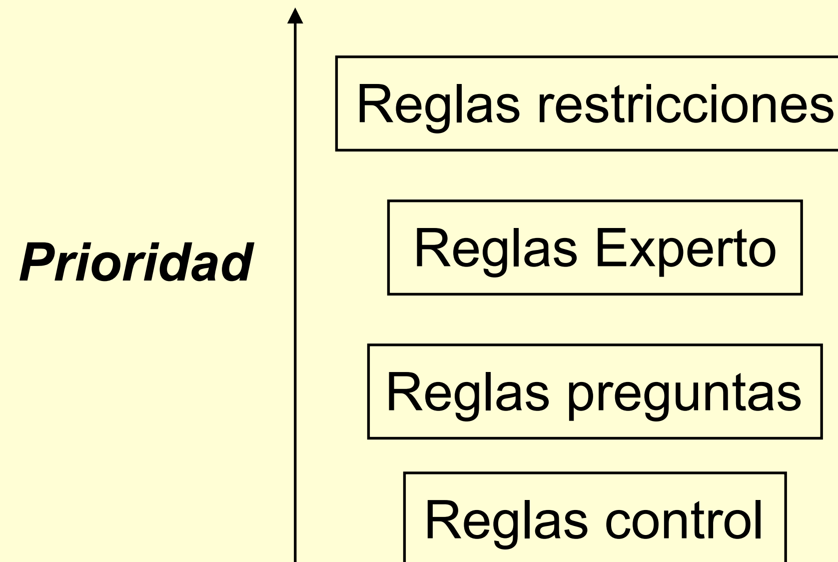
- Implementación más genéricas de las reglas de control

```
(deffacts informacion-control
  (fase deteccion)
  (secuencia-fases aislar recuperaracion deteccion))

(defrule cambia-fase
  (declase (salience -10))
  ?fase <- (fase ?fase-en-curso)
  ?lista <- (secuencia-fases ?siguiente $?resto)
=>
  (retract ?fase ?lista)
  (assert (fase ?siguiente))
  (assert (secuencia-fases ?resto ?siguiente)))
```

© José Angel Bañares

Aproximación 3 (cont.)

- Se pueden añadir niveles adicionales a la jerarquía de prioridades fácilmente
  - Ejemplo: SE que asigna gente a distintas tareas
    - Las reglas de restricción detectan estados improductivos o ilegales => Se sacarán inmediatamente las violaciones
    - Las reglas que preguntan al usuario sólo se utilizan cuando no se puede derivar la respuesta de las reglas del dominio



© José Angel Bañares

Evitar utilización Prioridades

```
(defrule A-Ganar
   (declare (salience 10))
   ?f <- (elige-movimiento)
   (cuadro-abierto gana)
=>
   (retract f)
   (assert (mueve gana)))
(defrule A-Bloquear
   (declare (salience 5))
   ?f <- (elige-movimiento)
   (cuadro-abierto bloquea)
=>
   (retract f)
   (assert (mueve bloquea)))
(defrule cualquiera
   ?f <- (elige-movimiento)
   (cuadro-abierto
            ?c&iz|der|medio)
=>
   (retract f)
   (assert (mueve-a ?c)))
```

```
(defrule A-Ganar
   ?f <- (elige-movimiento)
   (cuadro-abierto gana)
=>
   (retract f)
   (assert (mueve gana)))
(defrule A-Bloquear
   ?f <- (elige-movimiento)
   (cuadro-abierto bloquea)
   (not (cuadro-abierto gana))
=>
   (retract f)
   (assert (mueve bloquea)))
(defrule cualquiera
   ?f <- (elige-movimiento)
   (cuadro-abierto
            ?c&iz|der|medio)
   (not (cuadro-abierto gana))
   (not (cuadro-abierto bloquea))
=>
   (retract f)
   (assert (mueve-a ?c)))
```

© José Angel Bañares

3.4 Módulos en CLIPS

- **CLIPS permite partir una base de conocimiento en módulos**
  - Por defecto CLIPS define el módulo `MAIN`
  - Para definir módulos

```
(defmodule <nombre-modulo> [<comentario>])
```

```
CLIPS> (clear)
CLIPS> (deftemplate  sensor (slot nombre))
CLIPS> (ppdeftemplate sensor)
(deftemplate MAIN::sensor
   (slot nombre))
CLIPS> (defmodule DETECCION)
CLIPS> (defmodule AISLAR)
CLIPS> (defmodule RECUPERACION)
CLIPS> (defrule ejemplo1 =>)
CLIPS> (ppdefrule ejemplo1)
(defrule RECUPERACION::ejemplo1
   =>)
CLIPS>
```

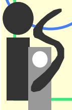
© José Angel Bañares

Módulos

- **El módulo en curso se cambia**
  - cuando se especifica el nombre de un módulo en una construcción,o mediante `set-current-module`
  - Las instrucciones CLIPS operan sobre las construcciones del módulo en curso o del módulo especificado

```
CLIPS> (deftemplate sensor
                   (slot nombre))
CLIPS> (ppdeftemplate sensor)
(deftemplate MAIN::sensor
   (slot nombre))
CLIPS> (defmodule DETECCION)
CLIPS> (defmodule AISLAR)
CLIPS> (defmodule RECUPERACION)
CLIPS> (defrule ejemplo1 =>)
CLIPS> (ppdefrule ejemplo1)
(defrule RECUPERACION::ejemplo1
   =>)
CLIPS> (defrule
          AISLAR::ejemplo2 =>)
```

```
CLIPS> (get-current-module)
AISLAR
CLIPS> (set-current-module DETECCION)
AISLAR
CLIPS> (list-defrules)
CLIPS>  (set-current-module AISLAR)
DETECCION
CLIPS> (list-defrules)
ejemplo2
For a total of 1 defrule.
CLIPS> (list-defrules RECUPERACION)
ejemplo1
For a total of 1 defrule.
CLIPS> (list-defrules *)
```



© José Angel Bañares

Importando y Exportando Hechos

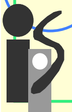
- **A diferencia de** `deffacts` **y** `defrule`**,** `deftemplate` **(y todos los hechos que utilizan deftemplate), pueden ser compartidos con otros módulos**
  - Exportar

    ```
    (export ?ALL) (export deftemplate ?ALL)
    (export ?NONE) (export deftemplate ?NONE)
    (export deftemplate <deftemplate>+)
    ```

  - Importar

    ```
    (import ?ALL) (import deftemplate ?ALL)
    (import ?NONE) (import deftemplate ?NONE)
    (import deftemplate <deftemplate>+)
    ```

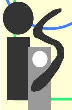
  - Una construcción debe definirse antes de importarse, pero no tiene que estar definida antes de exportarse.
  - Los módulos, salvo MAIN, no pueden redefinir lo que importan y exportan



© José Angel Bañares

Importar y exportar hechos

```
CLIPS>(defmodule DETECCION
        (export deftemplate fallo))
CLIPS>(deftemplate DETECCION::fallo
        (slot componente))
CLIPS>(defmodule AISLAR
        (export deftemplate posible-causa))
CLIPS>(deftemplate AISLAR::posible-causa
        (slot componente))
CLIPS>(defmodule RECUPERACION
        (import DETECCION deftemplate fallo)
        (import AISLAR deftemplate posible-causa))

CLIPS>(deffacts DETECCION::inicio
         (fallo (componente A)))

CLIPS>(deffacts AISLAR::inicia
        (posible-causa (componente B)))

CLIPS>(deffacts RECUPERACION::inicio
        (fallo (componente C))
        (posible-causa (componente D)))
```



© José Angel Bañares

Importar y exportar hechos

```
CLIPS> (reset)
CLIPS> (facts DETECCION)
f-1     (fallo (componente A))
f-3     (fallo (componente C))
For a total of 2 facts.
CLIPS> (facts AISLAR)
f-2     (posible-causa (componente B))
f-4     (posible-causa (componente D))
For a total of 2 facts.
CLIPS> (facts RECUPERACION)
f-1     (fallo (componente A))
f-2     (posible-causa (componente B))
f-3     (fallo (componente C))
f-4     (posible-causa (componente D))
For a total of 4 facts.
CLIPS>
```

© José Angel Bañares

Módulos y control

- **Cada módulo tiene su propia agenda**

```
(defrule DETECCION::regla-1
   (fallo (componente A | C))
  =>)

(defrule AISLAR::regla-2
   (posible-causa (componente B | D))
  =>)

(defrule RECUPERACION::regla-3
   (fallo (componente A | C))
   (posible-causa (componente B | D))
  =>)
```

```
CLIPS> (get-current-module)
RECUPERACION
CLIPS> (agenda)
0       regla-3: f-3,f-4
0       regla-3: f-3,f-2
0       regla-3: f-1,f-4
0       regla-3: f-1,f-2
For a total of 4 activations.
CLIPS> (agenda *)
MAIN:
DETECCION:
   0       regla-1: f-3
   0       regla-1: f-1
AISLAR:
   0       regla-2: f-4
   0       regla-2: f-2
RECUPERACION:
   0       regla-3: f-3,f-4
   0       regla-3: f-3,f-2
   0       regla-3: f-1,f-4
   0       regla-3: f-1,f-2
For a total of 8 activations.
```

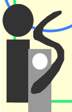
© José Angel Bañares

current focus

- **Clips mantiene un** current focus**, que determina la agenda que se usa**
  - La instrucción `(focus <nombre-modulo>+)`, designa el **current focus**.
  - Por defecto el current focus es `MAIN` (al ejecutar `clear` o `reset`), y no cambia al cambiarse de módulo
  - `focus` cambia el current focus y apila el anterior en una pila (**focus stack**). Cuando la agenda del current focus se vacia se obtiene el siguiente de la pila.
  - Funciones Utiles: `get-focus-stack`, `get-focus`, `pop-focus`

```
CLIPS> (watch rules)
CLIPS> (focus DETECCION)
TRUE
CLIPS> (run)
FIRE    1 regla-1: f-3
FIRE    2 regla-1: f-1
CLIPS> (focus AISLAR)
TRUE
CLIPS> (focus RECUPERACION)
TRUE
```

```
CLIPS> (list-focus-stack)
RECUPERACION
AISLAR
CLIPS> (run)
FIRE    1 regla-3: f-3,f-4
FIRE    2 regla-3: f-3,f-2
FIRE    3 regla-3: f-1,f-4
FIRE    4 regla-3: f-1,f-2
FIRE    5 regla-2: f-4
FIRE    6 regla-2: f-2
CLIPS>
```



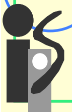
© José Angel Bañares

return

```
CLIPS> (clear)
CLIPS>
(defmodule MAIN
   (export deftemplate initial-fact))
CLIPS>
(defmodule DETECCION
  (import MAIN deftemplate initial-fact))
CLIPS> (defrule MAIN::principio
           =>
           (focus DETECCION))
CLIPS> (defrule DETECCION::ejemplo-1
          =>
          (return)
          (printout t "No printout" crlf))
CLIPS>  (defrule DETECCION::ejemplo-2
        =>
          (return)
          (printout t "No printout" crlf))
CLIPS> (reset)
<== Focus MAIN
==> Focus MAIN.
```

```
CLIPS> (run)
FIRE    1 principio: f-0
==> Focus DETECCION from MAIN
FIRE    2 ejemplo-1: f-0
<== Focus DETECCION to MAIN
<== Focus MAIN
CLIPS> (focus DETECCION)
==> Focus DETECCION
TRUE
CLIPS> (run)
FIRE    1 ejemplo-2: f-0
<== Focus DETECCION
==> Focus MAIN
<== Focus MAIN
CLIPS>
```

- return
  - Se deja de ejecutar la RHS
  - Saca el current focus stack
- pop-focus
  - Se acaba de ejecutar la RHS
  - Saca el current focus stack

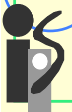


© José Angel Bañares

Auto-focus

- Si una regla tiene declarado auto-focus con el valor true, se activa su módulo automáticamente si la regla se sensibiliza
  - Útil spara detectar violación de restricciones

```
CLIPS> (clear)
CLIPS> (defmodule MAIN
          (export deftemplate initial-fact))
CLIPS> (defmodule DETECCION
          (import MAIN deftemplate initial-fact))
CLIPS> (defrule DETECCION::ejemplo
            (declare (auto-focus TRUE))
          =>)
CLIPS> (reset)
<== Focus MAIN
==> Focus MAIN
==> Focus DETECCION from MAIN
```

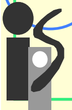


© José Angel Bañares

Reemplazar fases y hechos de control

```
(clear)
(defmodule DETECCION)
(defmodule AISLAR)
(defmodule RECUPERACION)
(deffacts MAIN::informacion-control
  (secuencia-fases DETECCION
           AISLAR RECUPERAR))
(defrule MAIN::cambia-fase
   ?lista <- (secuencia-fases
                 ?siguiente $?resto)
=>
   (focus ?siguiente)
   (retract ?lista)
   (assert (secuencia-fases
               ?resto ?siguiente)))
```

```
CLIPS> (reset)
CLIPS> (watch rules)
CLIPS> (watch focus)
CLIPS> (run 5)
FIRE    1 cambia-fase: f-1
==> Focus DETECCION from MAIN
<== Focus DETECCION to MAIN
FIRE    2 cambia-fase: f-2
==> Focus AISLAR from MAIN
<== Focus AISLAR to MAIN
FIRE    3 cambia-fase: f-3
==> Focus RECUPERACION from MAIN
<== Focus RECUPERACION to MAIN
FIRE    4 cambia-fase: f-4
==> Focus DETECCION from MAIN
<== Focus DETECCION to MAIN
FIRE    5 cambia-fase: f-5
==> Focus AISLAR from MAIN
<== Focus AISLAR to MAIN
```



© José Angel Bañares

Ventajas defmodule

- Permite Dividir la base de conocimiento
- Permite controlar que hechos son visibles en cada módulo
- Se puede controlar sin utilizar prioridades ni hechos de control
  - Es posible salir de una fase, y volver a la fase posteriormente para ejecutar las instancias que todavía quedan activas en la agenda.

© José Angel Bañares

3.5 Ejemplo con módulos

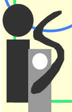
- **El problema del granjero, el lobo, la cabra y la col**
  - Utilización de módulos
  - Definición de funciones y esquemas algorítmicos clásicos para repeticiones y composiciones condicionales

- **Módulo principal**

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

- **Función auxiliar**

```
(deffunction MAIN::opuesta (?orilla)
   (if (eq ?orilla izquierda)
       then derecha
       else izquierda))
```



© José Angel Bañares

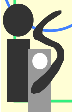
Módulo principal

- **Representación de estados**

```
(deftemplate MAIN::nodo
   (slot localizacion-granjero )
   (slot localizacion-lobo)
   (slot localizacion-cabra)
   (slot localizacion-col)
   (multislot camino))
```

- **Estado inicial**

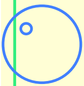
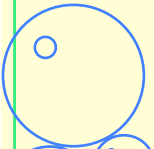
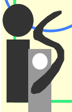
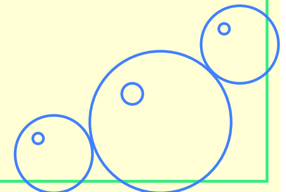
```
(deffacts MAIN::nodo-inicial
     (nodo
          (localizacion-granjero izquierda)
          (localizacion-lobo izquierda)
          (localizacion-cabra izquierda)
          (localizacion-col izquierda)
          (camino)))
```



© José Angel Bañares

Módulo principal

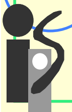
```
(defrule MAIN::movimiento-solo
  ?nodo <- (nodo (localizacion-granjero ?orilla-actual)
                 (camino $?movimientos))
=>
  (duplicate ?nodo
         (localizacion-granjero (opuesta ?orilla-actual))
          (camino $?movimientos solo)))

(defrule MAIN::movimiento-con-lobo
  ?nodo <- (nodo (localizacion-granjero ?orilla-actual)
                 (localizacion-lobo ?orilla-actual)
                 (camino $?movimientos))
 =>
  (duplicate ?nodo
         (localizacion-granjero (opuesta ?orilla-actual))
         (localizacion-lobo (opuesta ?orilla-actual))
         (camino $?movimientos lobo)))
```

© José Angel Bañares

Módulo principal

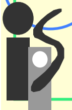
```
(defrule MAIN::movimiento-con-cabra
  ?nodo <- (nodo (localizacion-granjero ?orilla-actual)
                 (localizacion-cabra ?orilla-actual)
                 (camino $?movimientos))
  =>
  (duplicate ?nodo
        (localizacion-granjero (opuesta ?orilla-actual))
        (localizacion-cabra (opuesta ?orilla-actual))
        (camino $?movimientos cabra)))

(defrule MAIN::movimiento-con-col
  ?nodo <- (nodo (localizacion-granjero ?orilla-actual)
               (localizacion-col ?orilla-actual)
               (camino $?movimientos))
  =>
  (duplicate ?nodo
       (localizacion-granjero  (opuesta ?orilla-actual))
       (localizacion-col  (opuesta ?orilla-actual))
       (camino $?movimientos col)))
```



© José Angel Bañares

Módulo de restricciones

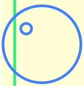
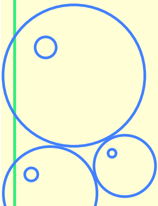
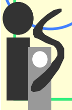
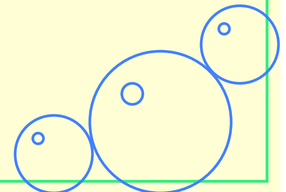
```
(defmodule RESTRICCIONES
  (import MAIN deftemplate nodo))

(defrule RESTRICCIONES::lobo-come-cabra
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (localizacion-granjero ?l1)
                 (localizacion-lobo ?l2&~?l1)
                 (localizacion-cabra ?l2))
  =>
  (retract ?nodo))

(defrule RESTRICCIONES::cabra-come-col
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (localizacion-granjero ?l1)
                 (localizacion-cabra ?l2&~?l1)
                 (localizacion-col ?l2))
  =>
  (retract ?nodo))
```



© José Angel Bañares

Módulo de restricciones

```
(defrule RESTRICCIONES::camino-circular
  (declare (auto-focus TRUE))
  (nodo  (localizacion-granjero ?granjero)
         (localizacion-lobo ?lobo)
         (localizacion-cabra ?cabra)
         (localizacion-col ?col)
         (camino $?movimientos-1))
  ?nodo <- (nodo (localizacion-granjero ?granjero)
                 (localizacion-lobo ?lobo)
                 (localizacion-cabra ?cabra)
                 (localizacion-col ?col)
                 (camino $?movimientos-1 ? $?movimientos-2))
  =>
  (retract ?nodo))
```

© José Angel Bañares

Módulo solución

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

(defrule SOLUCION::reconoce-solucion
  (declare (auto-focus TRUE))
  ?nodo <- (nodo  (localizacion-granjero derecha)
                  (localizacion-lobo derecha)
                  (localizacion-cabra derecha)
                  (localizacion-col derecha)
                  (camino $?movimientos))
  =>
  (retract ?nodo)
  (assert (solucion $?movimientos)))
```

© José Angel Bañares

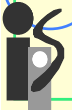
Módulo solución

```
(defrule SOLUCION::escribe-solucion
  ?mv <- (solucion $?m)
  =>
  (retract ?mv)
  (printout t crlf crlf  "Solucion encontrada " crlf)
  (bind ?orilla derecha)
  (loop-for-count (?i 1 (length $?m))
     (bind ?cosa (nth ?i $?m))
     (printout t "El granjero se mueve "
        (switch ?cosa
           (case solo then "solo ")
           (case lobo then "con el lobo")
           (case cabra then "con la cabra ")
           (case col then "con la col "))
           " a la " ?orilla "." crlf)
      (bind ?orilla (opuesta ?orilla))))
```

© José Angel Bañares

Traza del problema del granjero

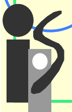
```
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (watch activations)
CLIPS> (watch focus)
CLIPS> (reset)
<== Focus MAIN
==> Focus MAIN
<== f-0     (initial-fact)
==> f-0     (initial-fact)
==> f-1     (nodo (localizacion-granjero izquierda)
                (localizacion-lobo izquierda)
                (localizacion-cabra izquierda)
                (localizacion-col izquierda)
                (camino))
==> Activation 0      movimiento-con-col: f-1
==> Activation 0      movimiento-con-cabra: f-1
==> Activation 0      movimiento-con-lobo: f-1
==> Activation 0      movimiento-solo: f-1
```



© José Angel Bañares

Traza del problema del granjero

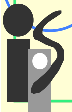
```
CLIPS> (run 1)
FIRE    1 movimiento-solo: f-1
==> f-2     (nodo (localizacion-granjero derecha)
                  (localizacion-lobo izquierda)
                  (localizacion-cabra izquierda)
                  (localizacion-col izquierda)
                  (camino solo))
==> Focus RESTRICCIONES from MAIN
==> Activation 0      cabra-come-col: f-2
==> Activation 0      lobo-come-cabra: f-2
==> Activation 0      movimiento-solo: f-2
CLIPS> (get-focus-stack)
(RESTRICCIONES MAIN)
CLIPS> (agenda)
0      lobo-come-cabra: f-2
0      cabra-come-col: f-2
For a total of 2 activations.
```



© José Angel Bañares

Traza del problema del granjero

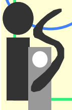
```
CLIPS> (run 1)
FIRE    1 lobo-come-cabra: f-2
<== f-2      (nodo
                (localizacion-granjero derecha)
                (localizacion-lobo izquierda)
                (localizacion-cabra izquierda)
                (localizacion-col izquierda)
                (camino solo))
<== Activation 0      movimiento-solo: f-2
<== Activation 0      cabra-come-col: f-2
<== Focus RESTRICCIONES to MAIN
CLIPS> (get-focus-stack)
(MAIN)
CLIPS> (agenda)
0      movimiento-con-lobo: f-1
0      movimiento-con-cabra: f-1
0      movimiento-con-col: f-1
For a total of 3 activations.
```



© José Angel Bañares

Traza del problema del granjero

```
CLIPS> (run)

Solucion encontrada
El granjero se mueve con la cabra  a la derecha.
El granjero se mueve solo  a la izquierda.
El granjero se mueve con el lobo a la derecha.
El granjero se mueve con la cabra  a la izquierda.
El granjero se mueve con la col  a la derecha.
El granjero se mueve solo  a la izquierda.
El granjero se mueve con la cabra  a la derecha.


Solucion encontrada
El granjero se mueve con la cabra  a la derecha.
El granjero se mueve solo  a la izquierda.
El granjero se mueve con la col  a la derecha.
El granjero se mueve con la cabra  a la izquierda.
El granjero se mueve con el lobo a la derecha.
El granjero se mueve solo  a la izquierda.
El granjero se mueve con la cabra  a la derecha.
```



© José Angel Bañares

Ejercicio. Mundo bloques sin estrategia MEA

- **Suponer que no se cuenta con estrategia MEA**
  - Opción 1: Uso de focus
  - Opción 2: Uso de focus y auto-focus