Programación en Lenguajes basados en reglas. Aspectos metodológicos I.

J.A. Bañares Bañares

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

Última revisión: Septiembre, 2009

Programación LBR

- **Aspectos metodológicos I**
  - Documentación.
  - Derivación sistemática de reglas con una estrategia MEA.
- **Aspectos metodológicos II**
  - 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**

Objetivo

- **Objetivo:**
  - Presentación de una metodología para especificar, implementar y modificar sistemas de producción.
- **Método:**
  - Generación de soluciones parciales del problema y comprobación rigurosa antes de extenderlas.

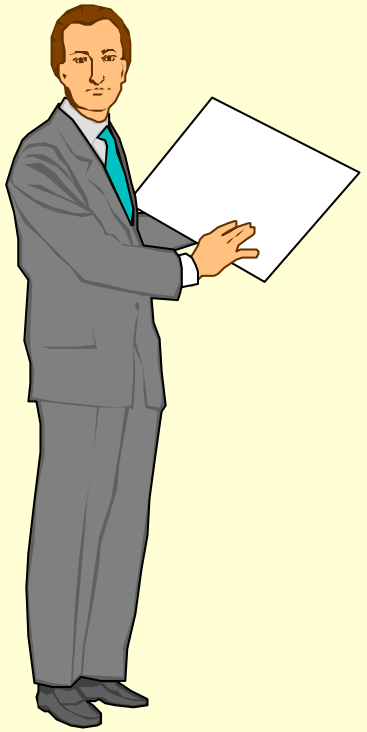
1. Problema Ejemplo

- **Definición del problema:**
  - Versión CLIPS del típico problema del mono y las bananas.
    - En una habitación de 10 x 10 x 10 hay
      - Un sofá pesado
      - Una escalera ligera
      - Un manojo de plátanos (suspendidos del techo, o sobre el sofá, sobre la escalera o sobre el suelo).
      - Un mono desesperadamente hambriento capaz de mover objetos ligeros, y
      - La manta del mono.
    - Escribe un programa que a partir de una descripción de los objetos y su localización de las instrucciones al mono para coger el manojo de plátanos.

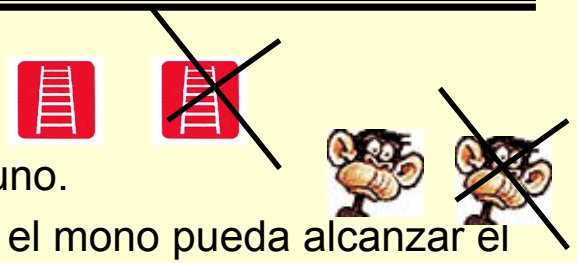
2. Análisis de requisitos

1 **Determinar la generalidad de la solución**

- Generalizamos a la representación de una clase de problemas de **planificación** con:
  - Un **agente** (el mono) capaz de desplazarse y manipular (mover) objetos
  - **Objetos** físicos (sofá, escalera, banana y manta) situados en un espacio tridimensional.
  - Un objeto particular será el **objetivo** (Coger bananas)
- Construimos un **Lista de Decisiones** para
  - Ayudar a restringir el problema
  - Determinar una representación adecuada
  - Si hubiera un experto (cliente) del problema nos ayudaría a definir la lista de decisiones.



Lista de decisiones



1. Hay exactamente un escalera en la habitación.
2. El mono es el agente del problema, y sólo hay uno.
3. La escalera es lo suficientemente alta para que el mono pueda alcanzar el techo, y lo suficientemente ligera para ser movida por el mono.
4. El **tamaño** de los objetos físicos será **ignorado**.
5. El peso de los objetos físicos se indicará puesto que el mono no puede mover objetos pesados.
6. El mono puede ver todos los objetos de la habitación.
7. El mono puede llevar a cabo ciertas acciones sin tener que indicar en detalle como se realizan: **Caminar** a una localización, **mover** un objeto a cualquier localización, **subir** y **bajar** de un objeto, y **coger** y **dejar** objetos.
8. El programa final permitirá en la planificación sólo las acciones descritas.

2. Objetos y Atributos

- **Completar con los objetos del dominio:**
  - Objetos físicos:

```
(deftemplate cosa
   (slot nombre          ; Nombre único del objeto, tal
      (type SYMBOL)      ; como "bananas", "manta", "sofa".
      (default ?NONE))   ; Debe haber una "escalera".

   (slot localizacion    ; Coordenadas cartesianas de la
      (type SYMBOL)      ; forma X-Y donde X e Y son enteros
      (default ?NONE))   ; entre 1 y 10.

   (slot sobre           ; Pueden ser objetos como "suelo",
      (type SYMBOL)      ; "techo", cualquier nombre de
      (default nada))    ; objeto, o "nada" para indicar que
                         ; que esta cogido por el mono.
   (slot peso            ; Pesado o ligero
      (type SYMBOL)
      (allowed-symbols ligero pesado)
      (default ligero)))
```

Objeto Agente

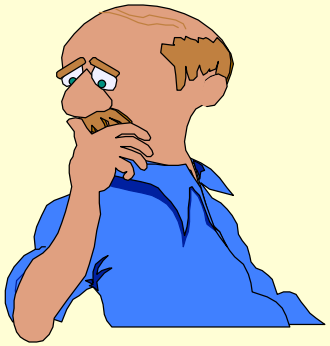
- Atributos del mono

```
(deftemplate mono

   (slot localizacion  ; Coorrdenadas cartesianas en la.
      (type SYMBOL)    ; forma X-Y donde X e Y son enteros
      (default nada))  ; entre 1 y 10

   (slot sobre         ; Localización vertical del mono,
      (type SYMBOL)    ; "suelo" o nombre de un objeto. En
      (default suelo)) ; este caso deben tener misma loc.

   (slot sostiene      ; "nada" o el nombre e un objeto.
      (type SYMBOL)    ; Los objetos deben tener la misma
      (default nada))) ; localización y ser ligeros.
```

3. Acciones

- **Acciones realizables por el mono:**
  - **Coger** y **dejar** un objeto
  - **Mover** un objeto a una localización horizontal
  - **Caminar** hasta una localización específica
  - **Subir** y **bajar** de un objeto.
- **Para cada acción se deben especificar**
  - precondiciones que deben cumplir los objetos en la habitación para realizar la acción.
  - poscondiciones que se deben verificar por los objetos tras realizar la acción.

Decisiones sobre acciones



- Completamos la **lista de decisiones** para determinar las precondiciones adecuadas:
  - El mono debe estar en la misma localización exacta que el objeto para poder cogerlo
  - El mono debe estar sobre la escalera para coger algo del techo
  - Los objetos dejados por el mono estarán en la posición del mono y caerán al suelo.
  - El mono puede descender de un objeto mientras sostiene algo.
  - El mono no puede subir a un objeto sosteniendo a otro objeto.
  - El mono sólo puede sostener un objeto cada vez.
- Los problemas que vayan surgiendo los anotaremos en una **lista de problemas**
  - Dos objetos pueden estar en la misma localización al mismo tiempo.

Acciones y sus pre/post

Coge Obj
pre:mono no sostiene nada,
Obj un objeto ligero,
mono está donde Obj,
Y o bien
mono en suelo y Obj no en techo
o
mono en escalera y O en techo
post:mono sostiene Obj
Obj está sobre "nada"

Deja Obj
pre:mono sostiene Obj
Obj es un objeto
post:mono no sostiene nada
Obj está sobre suelo

Mueve Obj a X-Y
pre:mono está sobre el suelo
mono sostiene Obj
Obj es objeto y no está en X-Y
post:mono está en X-Y
Obj está en X-Y

Camina a X-Y
pre:mono en suelo y no en X-Y
post:mono está en X-Y

Salta al suelo
pre:mono no está en el suelo
post:mono está en el suelo

Sube a Obj
pre:mono está en el suelo
mono no sostiene nada
Obj es un objeto y está en suelo
El mono y Obj en el mismo sitio
post:mono está en el suelo

4. Estrategia de resolución

- **Estrategia más adecuada:**
  - Dirigida por objetivos
    - Si todas se satisfacen las precondiciones se ejecuta la acción para lograr el objetivo
    - Si no se cumple alguna de las precondiciones se establece como subobjetivo que se satisfaga dicha precondición.
- **Utilizamos la estrategia MEA: (Means-End Analysis)**
  - Puesto que queremos que se satisfagan los subobjetivos antes de tratar de nuevo el objetivo original.

```
(set-strategy MEA)
```

- **La memoria de trabajo registra los objetivos pendientes y satisfechos**
  - Debemos representar los objetivos

Objetivos

- **Clasificamos los objetivos**
  - El objetivo es realizar acciones => clasificamos las acciones
    - COGE y DEJA: modifican atributo **sostiene**.
    - SALTA y SUBE: modifican atributo **sobre**.
    - MUEVE y CAMINA: modifican atributo **localización**.

```
(deftemplate objetivo
   (slot estado          ; Permite seguir el proceso de resolución
      (type SYMBOL)      ; "activo" o "satisfecho"
      (allowed-symbols activo satisfecho)
      (default ?NONE))
   (slot tipo            ;"SOSTIENE", "SOBRE", "LOCALIZACION"
      (type SYMBOL)
      (allowed-symbols sostiene sobre localizacion)
      (default ?NONE))
(multislot argumentos  ; Para sostiene el nombre del objeto o "nada"
      (type SYMBOL)    ; Para sobre "suelo" o nombre de un objeto
      (default ?NONE)));Para localizacion "nada" para indicar
                              ;  camina o un objeto para indicar mueve. A
                       ;  continuación van las coordenadas X-Y
```

5. Entradas

- **Entradas:**
  - Configuración inicial de la habitación
  - Objetivo a conseguir
- **Añadimos a la lista de decisiones**
  - Restricciones sobre configuraciones de la habitación
    - Debe haber sólo un mono, sólo una escalera y al menos un objetivo.
    - Se verifican las restricciones sobre los valores esperados en los atributos.
    - Si hay más de un objetivo en la memoria se supone que la intención es satisfacerlos secuencialmente.
  - Una buena extensión del programa sería definir las reglas que forzaran configuraciones legales.

Salidas

- **Salidas**
  - Instrucciones que el mono es capaz de realizar
    - Saltar sobre el suelo
    - camina a LOCALIZACION
    - Sube sobre OBJETO, Coge OBJETO, Deja OBJETO
    - Mueve OBJETO a LOCALIZACION

    Donde LOCALIZACION es una coordenada cartesiana y

    OBJETO es el atributo nombre de una instancia de cosa.
  - Mensajes de ayuda
    - Mono ya está sobre OBJETO, OBJETO ya esta cogido
    - El OBJETO ya está en LOCALIZACION
    - Enhorabuena, no hay objetivos activos
    - Imposible, el OBJETIVO no puede ser resuelto.
- **Añadimos a la lista de decisiones**
  - no se comprueba que la memoria de trabajo tiene una configuración legal.

Inicialización de la MT

- **Para chequear el programa es necesario producir muchas configuraciones iniciales:**
  - Cada configuración inicial chequea algún aspecto.
  - Definiremos reglas que inicializan la MT para chequear un aspecto.

```
(deftemplate casoTest
   (slot tipo           ; Los mismos que tipos de objetivos para
      (type SYMBOL)     ;  chequear cada tipo de acción y "general"
      (allowed-symbols ;  para chequear todas las reglas juntas.
                        ; sostiene sobre localizacion general)
      (default ?NONE))
   (multislot nombre    ; Nombre del caso. Una símbolo que
      (type SYMBOL)     ; describe el tipo de objetivo chequeado
      (default ?NONE)))

(defrule Test_General_Principio ""
  (casoTest (tipo general) (nombre principio))
  =>
  (assert (cosa (nombre bananas) (sobre techo)
                (localizacion 9-9))) ... )))
```

6. Diseño de las reglas

- **Estrategia para el diseño de reglas**
  - Reglas agrupadas por objetivos:
    - Se diseña un **conjunto de reglas** por cada acción relacionada con **un objetivo**.
  - Después de crear cada conjunto de reglas se examinan las **interacciones entre los conjuntos de reglas**.
    - Comprobación de cada conjunto de reglas mediante el desarrollo exhaustivo de configuraciones iniciales.
  - Para mantener una configuración legal de la memoria de trabajo
    - Se parte de una configuración inicial legal, y
    - se mantienen la legalidad con cada regla

6.1 Escritura de reglas

- **Procedimiento para desarrollar un conjunto de reglas para cada objetivo**
  - Asumiendo un modelo de solución general del problema (objetos y acciones) se produce **un primer prototipo**.
  - El procedimiento permite trasladar las acciones en reglas.
    - Las precondiciones pasan a la parte izquierda de la regla (LHS)
    - Las poscondiciones a la parte acción.
  - A partir de cada una de las **reglas base**, generamos otras reglas en las que no se cumplen todas las precondiciones
    - Esto implicará crear nuevas reglas que establecen nuevos subobjetivos

Pasos

- **Pasos en el diseño de reglas para un objetivo**
  1. Elegir una acción relacionada directamente con el objetivo
  2. Escribir las precondiciones de la acción incluyendo
     - Una condición relacionada con el objetivo
     - Una condición por cada una de las precondiciones
  3. Escribir las poscondiciones de la acción incluyendo
     - Un mensaje indicando la acción realizada
     - El objetivo cumplido es eliminado o marcado como satisfecho
  4. Comprueba que la regla deja la MT e una configuración legal
  5. Comprueba interacciones de la regla con otras del objetivo
  6. Traslada la regla a CLIPS

Pasos (cont.)

7 Las condiciones en la **LHS** de la regla cumplen 3 propósitos:

a Especificar el objetivo

b Asegurar que la acción se puede realizar

c Asegurarse de que los resultados de la acción no existen ya

- Elige una condición (b) y crea una nueva regla en la que dicha condición no se debe satisfacer para que la regla este habilitada.
- Comprueba si alguna otra condición debe ser modificada (por resultar superflua o incorrecta).
- En la nueva regla la **LRH** establece el subobjetivo de satisfacer la condición.

Pasos (cont.)

8 **Si** el paso 7 da una nueva regla **entonces**
vuelve al paso 4
sino crea una nueva regla modificando la condición (c)
La regla comprobará que todas las condiciones que se deben verificar al ejecutar la acción ya se cumplen marcará el objetivo como satisfecho,.e imprimirá el mensaje "**objetivo ya satisfecho**",

- **Buena práctica de ingeniería del software**
  - Colocar todas las posibles reglas relacionadas con una acción en un lugar.

6.2 Chequeo de las reglas

- **Las reglas se comprueban según se escriben**
  - Pasos 4 y 5
- **El grupo de reglas está completo si considera todas las configuraciones de la MT relacionadas con las precondiciones de las acciones para conseguir el objetivo.**
  - Paso 7
- **Cuando el grupo esté terminado se definen un conjunto de configuraciones iniciales de la MT y se comprueban.**
- **La verificación completa incluirá**
  - Ejecución del programa
  - Identificación de configuraciones de la MT no alcanzables
  - Configuraciones alcanzables que no llevan a la solución
  - Cualquier otra anomalía

6.3.1 Ejemplo, Objetivo SOBRE

- **Objetivo SOBRE:**
  - Acciones BAJA (SALTA) al suelo y SUBE sobre <Objeto>
  - 1. Salta sobre el suelo

    Precondiciones:

    El mono no está sobre el suelo

    Poscondiciones

    El mono está sobre el suelo
  - 2. Condiciones para LHS

    SI hay un objetivo de estar en el suelo

    y el mono no está en el suelo

    ⇒ *Objetivo*
  - 3. Acciones en LRS

    ⇒ *Condición*

    ENTONCES **escribe** “Salta al suelo”

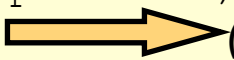
    y **modifica** el atributo `sobre` del mono con el valor “`suelo`”

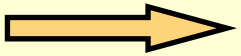
    y **modifica** el `estado` del objetivo con “`satisfecho`”

Ejemplo (cont.)

- 4. La regla deja la MT en una configuración legal
- 5. No hay otras reglas para comprobrar la consistencia
- 6. Codifico la regla

```
(defrule Sobre_Suelo "Salta sobre el suelo"
  ?objetivo <- (objetivo (estado activo) (tipo sobre)
                         (argumentos suelo))
  ?mono <- (mono (sobre ?sobre&~suelo))
  =>
  (printout t "El mono salta desde " ?sobre " al suelo." crlf)
  (modify ?mono (sobre suelo))
  (modify ?objetivo (estado satisfecho)))
```





- 7.Modificar cualquier precondición cambiaría el tipo de acción o crearía una situación en la que la acción ya estaría hecha.

Ejemplo (cont.)

- 8. Consideramos la regla que se dispara cuando el objetivo de estar en el suelo ya se cumple.

```
(defrule Sobre_Suelo_Satisfecho
          "Salta sobre el suelo ya esta satisfecho"
  ?objetivo <- (objetivo (estado activo) (tipo sobre)
                         (argumentos suelo))
  ?mono <- (mono (sobre suelo))
  =>
  (printout t "El mono ya esta ya sobre el suelo." crlf)
  (modify ?objetivo (estado satisfecho)))
```

Ejemplo (cont.)

- **Chequeo regla Sobre_Suelo:**

```
(defrule Test_Sobre_Suelo ""
  (casoTest (tipo sobre) (nombre suelo))
  =>
  (assert (cosa (nombre escalera) (sobre suelo)
                (localizacion t5-7) (peso ligero)))
  (assert (mono (sobre escalera) (localizacion t5-7)))
  (assert (objetivo (estado activo) (tipo sobre)
                    (argumentos suelo))))
```

- **Ejemplo Chequeo**

```
CLIPS> (load "myb.dec")
TRUE
CLIPS> (load "myb1.rul")
Defining defrule: Sobre_Suelo +j+j
Defining defrule: Sobre_Suelo_Satisfecho =j+j
TRUE
CLIPS> (load "myb1.tst")
Defining defrule: Test_Sobre_Suelo +j
TRUE
```

Proceso chequeo

```
CLIPS> (assert  (casoTest (tipo sobre) (nombre suelo)))
<Fact-0>
CLIPS> (facts)
f-0     (casoTest (tipo sobre) (nombre suelo))
For a total of 1 fact.
CLIPS> (agenda)
0      Test_Sobre_Suelo: f-0
For a total of 1 activation.
CLIPS> (run 1)
CLIPS> (facts)
f-0     (casoTest (tipo sobre) (nombre suelo))
f-1     (cosa (nombre escalera) (localizacion t5-7) (sobre suelo)
              (peso ligero))
f-2     (mono (localizacion t5-7) (sobre escalera) (sostiene nada))
f-3     (objetivo (estado activo) (tipo sobre) (argumentos suelo))
For a total of 4 facts.
CLIPS> (agenda)
0      Sobre_Suelo: f-3,f-2
For a total of 1 activation.
```

Proceso de chequeo (cont.)

```
CLIPS> (run 1)
El mono salta desde escalera al suelo.
CLIPS> (facts)
f-0     (casoTest (tipo sobre) (nombre suelo))
f-1     (cosa (nombre escalera) (localizacion t5-7) (sobre suelo)
              (peso ligero))
f-4     (mono (localizacion t5-7) (sobre suelo) (sostiene nada))
f-5     (objetivo (estado satisfecho) (tipo sobre)
                  (argumentos suelo))
For a total of 4 facts.
CLIPS> (agenda)
```

Ejemplo, sube sobre objeto

- 1.Sube al objeto

  Precondiciones:

  El mono está en el suelo

  El mono no sostiene nada

  El objeto es una cosa

  El objeto está en el suelo → *Añadimos a la lista de decisiones*

  El mono y el objeto están en la misma localización

  Poscondiciones

  El mono está sobre el objeto

- 2 y 3 SI **hay un objetivo de estar sobre el objeto**

  y el objeto está en la localización L sobre el suelo

  y el mono está en L, no sostiene nada,

  y **no está sobre el objeto**

  ENTONCES **escribe** "Sube sobre el objeto"

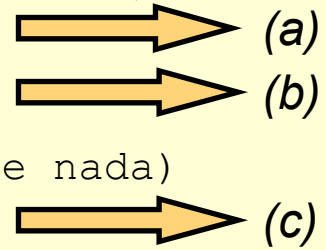
  y **modifica** el atributo `sobre` del mono con el objeto

  y **modifica** el `estado` del objetivo con "`satisfecho`"

Sube sobre obj. (cont.)

- 4. La regla deja la MT en una configuración legal
- 5. No interacciona con la regla Sobre_Suelo porque tienen objetivos distintos.
- 6. Codifico la regla

```
(defrule Sobre_Objeto "Sube directamente"
  ?objetivo <- (objetivo (estado activo) (tipo sobre)
                         (argumentos ?obj))
  (cosa (nombre ?obj) (localizacion ?lugar)
        (sobre suelo))
  ?mono <- (mono (localizacion ?lugar) (sostiene nada)
           (sobre ~?obj))
  =>
  (printout t "mono sube sobre " ?obj "." crlf)
  (modify ?mono (sobre ?obj))
  (modify ?objetivo (estado satisfecho)))
```



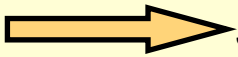
Deja lo que sostiene

- 7. Otras reglas para esta acción
  - Considerar la secuencia en la que que las precondiciones deberían satisfacerse.
    - Si el mono sostiene algo debe dejarlo antes de subir

      Podría ser que el objetivo fuera subir sobre el objeto que sostiene el mono => relajamos condición de que el objeto esté sobre el suelo

      No estará en conflicto con Sobre_Suelo porque suelo no es una cosa

```
(defrule Sobre_Objeto_Sostiene "Deja Objeto para subir"
  (objetivo (estado activo) (tipo sobre) (argumentos ?obj))
  (cosa (nombre ?obj) (localizacion ?lugar))
  (mono (localizacion ?lugar)
        (sostiene ~nada))
  (not (objetivo (estado activo) (tipo sostiene)
                 (argumentos nada)))
  =>
  (assert (objetivo (estado activo) (tipo sostiene)
                    (argumentos nada))))
```



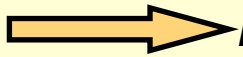
Localización diferente

- El mono debe estar en la misma localización que el objeto al que tiene que subir

  No hay razón para especificar otras características del mono salvo la localización.

  No estará en conflicto con Sobre_Objeto ni con Sobre_Objeto_Sostiene porque la localización del mono difiere.

```
(defrule Sobre_Objeto_Localizacion
        "Camina a nueva localizacion para subir sobre objeto"
  (objetivo (estado activo) (tipo sobre) (argumentos ?obj))
  (cosa (nombre ?obj) (localizacion ?lugar) (sobre suelo))
  (mono (localizacion ~?lugar))
  (not (objetivo (estado activo)
                 (tipo localizacion) (argumentos ?lugar)))
  =>
  (assert (objetivo (estado activo) (tipo localizacion)
                    (argumentos ?lugar))))
```



Objetivo satisfecho

- 8. Consideramos la regla que se dispara cuando el objetivo de estar en el suelo ya se cumple.

```
(defrule Sobre_Objeto_Satisfecho "Ya esta sobre el objeto"
  ?objetivo <- (objetivo (estado activo) (tipo sobre)
                         (argumentos ?obj))
  (cosa (nombre ?obj) (localizacion ?lugar) (sobre suelo))
  (mono (sobre ?obj) (localizacion ?lugar))
  =>
  (printout t "El mono ya est'a sobre el objeto "
              ?obj crlf)
  (modify ?objetivo (estado satisfecho)))
```

  - Se pueden sustituir Sobre_Suelo_Satisfecho y Sobre_Objeto_Satisfecho por

```
(defrule Sobre_Satisfecho "Ya esta sobre el objeto"
  ?objetivo <- (objetivo (estado activo) (tipo sobre)
                         (argumentos ?obj))
  (mono (sobre ?obj) (localizacion ?lugar))
  =>
  (printout t "El mono ya est'a sobre el objeto " ?obj crlf)
  (modify ?objetivo (estado satisfecho)))
```

Chequeo Objetivo Sobre

- **Una regla de test por cada regla**

```
(defrule Test_Sobre_Objeto ""
 (casoTest (tipo sobre) (nombre Objeto))
  =>
  (assert (cosa (nombre escalera) (sobre suelo)
                (localizacion t5-5) (peso ligero)))
  (assert (mono (sobre suelo) (localizacion t5-5)))
  (assert (objetivo (estado activo) (tipo sobre) (argumentos escalera))))

(defrule Test_Sobre_Objeto_Sostiene ""
 (casoTest (tipo sobre) (nombre Sostiene_Objeto))
  =>
  (assert (cosa (nombre escalera) (sobre suelo)
                (localizacion t5-5) (peso ligero)))
  (assert (mono (sobre suelo) (localizacion t5-5) (sostiene escalera)))
  (assert (objetivo (estado activo) (tipo sobre) (argumentos escalera))))

(defrule Test_Sobre_Objeto_Localizacion ""
 (casoTest (tipo sobre) (nombre Localizacion_Objeto))
  =>
 ... )))

(defrule Test_Sobre_Objeto_Satisfecho ""
 ... )))
```

6.3.2 Objetivo Sostiene

- **Objetivo SOSTIENE:**
  - Acciones DEJA <Objeto> y COGE <Objeto>
    - DEJA <Objeto>

```
(defrule Sostiene_Nada "Deja el objeto que lleve"
  ?objetivo <- (objetivo (estado activo) (tipo sostiene)
                         (argumentos nada))
  ?mono <- (mono (localizacion ?lugar)
                 (sostiene ?obj&~nada))
  ?cosa <- (cosa (nombre ?obj))
  =>
  (printout t "mono deja " ?obj "." crlf)
  (modify ?objetivo (estado satisfecho))
  (modify ?mono (sostiene nada))
  (modify ?cosa (sobre suelo)))

(defrule Sostiene_Nada_Satisfecho
        "Deja el objeto que lleve ya satisfecho"  ...))
```

COGE

- COGE <Objeto>

```
(defrule Sostiene_Objeto "Coge objeto desde la escalera"
  ?objetivo <- (objetivo (estado activo) (tipo sostiene)
                         (argumentos ?nombre))
  ?cosa <- (cosa (nombre ?nombre) (localizacion ?lugarCosa)
                 (sobre ?donde) (peso ligero))
  ?escalera <- (cosa (nombre escalera) (sobre suelo)
                     (localizacion ?lugarEscalera))
  ?mono <- (mono (sobre ?sobre) (sostiene nada))
  (test (or (and (eq ?sobre suelo) (neq ?donde techo))
            (and (eq ?sobre escalera) (eq ?donde techo)
                 (eq ?lugarCosa ?lugarEscalera))))
  (not (cosa (sobre ?nombre)))
  =>
  (printout t "mono coge " ?nombre "." crlf)
  (modify ?cosa (sobre nada))
  (modify ?mono (sostiene ?nombre))
  (modify ?objetivo (estado satisfecho)))
```

OR

- Lenguajes como OPS5 no permiten el OR

```
(defrule Sostiene_Objeto_NoTecho "Objeto no est'a en el techo"
  ?objetivo <- (objetivo (estado activo) (tipo sostiene)
                         (argumentos ?nombre))
  ?cosa <- (cosa (nombre ?nombre) (localizacion ?lugar)
                 (sobre ~techo) (peso ligero))
  ?mono <- (mono (sobre suelo) (sostiene nada)
                 (localizacion ?lugar))
  (not (cosa (sobre ?nombre)))
  => ... )
(defrule Sostiene_Objeto_Techo "Objeto en el techo"
  ?objetivo <- (objetivo (estado activo) (tipo sostiene)
                         (argumentos ?nombre))
  ?cosa <- (cosa (nombre ?nombre) (localizacion ?lugar)
                 (sobre techo) (peso ligero))
  ?escalera <- (cosa (nombre escalera) (localizacion ?lugar)
                (sobre suelo))
  ?mono <- (mono (sobre escalera) (sostiene nada))
  (not (cosa (sobre ?nombre)))
  => ...)
```

Nuevas reglas derivadas

- **El mono no debe sostener nada**
  - Independiente de que el objeto esté o no en el techo
- **El mono debe estar en la localización correcta**
  - Si el objeto está en el techo y la escalera bajo el objeto, se establece el objetivo de estar sobre la escalera
    - Las reglas del grupo SOBRE establecerán el objetivo de ir a la escalera.
    - Encadenando las acciones de los dos grupos SOBRE y SOSTIENE disminuye el número de reglas y se asegura que las acciones se llevan a cabo de forma adecuada.
  - Si el objeto no está en el techo se establece objetivo ir a la localización del objeto.
- **La escalera debe estar bajo el objeto si éste está en el techo**
- **El mono debe estar en el suelo si el objeto no está en el techo.**

6.3.3 Objetivo Localización

- **Diseño: Hay dos acciones:**
  - Mueve un objeto, denominado `Localizacion_Objeto`
  - Mueve mono a un lugar, denominado `Localizacion_Mono`

- **Expresión de subojetivos:**

  `Localizacion_Objeto` objetivo tipo "`localizacion`" y en argumentos un valor distinto de "`nada`" seguido de las coordenadas.

  `Localizacion_Mono` objetivo tipo "`localizacion`" y en argumentos el valor "`nada`" seguido de las coordenadas.

Subobjetivos Localización

- **Subobjetivo Localizacion_Objeto**

`Localizacion_Objeto:` Acción de mover el objeto

`Localizacion_Objeto_Sobre_Suelo:` Objeto cogido y mono no está en el suelo, estable objetivo de saltar al suelo

`Localizacion_Objeto_Sostiene:` Establece objetivo de coger objeto si no está cogido

`Localización_Objeto_Satisfecho`

- **Subobjetivo Localizacion_Mono**

`Localizacion_Mono:` Acción de caminar el mono cuando no sostiene nada

`Localizacion_Mono_Objeto:` Acción de caminar el mono cuando éste sostiene algo

`Localizacion_Mono_Sobre:` Establece objetivo de saltar al suelo sino está en el suelo

`Localización_Mono_Satisfecho`

7 Lista de decisiones

- Hay una única escalera en la habitación
- El mono se puede mover por sí mismo, y es el único ser de estas características en la habitación
- La escalera es suficientemente alta para que el mono, cuando esté sobre la escalera alcance el techo. Pesa poco
- El tamaño de los objetos físicos se ignorará
- El peso de los objetos debe expresarse ya que el mono no puede mover objetos pesados
- El mono puede ver todos los objetos en la habitación
- El mono puede llevar a cabo algunas acciones como: caminar, mover objetos, subir y bajar de objetos, y coger y dejar objetos
- El programa final mostrará la planificación de acciones para conseguir el objetivo
- La localización será un atributo de los objetos
- La localización horizontal será un punto definido con X e Y, la localización vertical será el suelo, el techo o sobre un objeto
- La situación inicial debe definir el objetivo

Lista de decisiones

- “nada” en el atributo sobre indica que el objeto está cogido
- El mono debe estar en el mismo lugar que un objeto para cogerlo
- El mono debe estar sobre la escalera para coger algo del techo
- Un objeto dejado por el mono estára en la misma localización sobre el suelo
- El mono puede bajar de un objeto mientras sostiene otro
- El mono no puede subir a un objeto si lleva otro
- El mono sólo puede sostener un objeto a la vez
- La estrategia será dirigida por objetivos, con los tipos “localizacion”, “sobre” y “sostiene”
- Los objetivos tendrán el atributo estado “activo” cuando entren en la MT
- Los objetivos tendrán el atributo estado “satisfecho” cuando se alcancen
- No se van a chequear violaciones de configuraciones legales de la MT
- El programa no ayuda al usuario a establecer una configuración inicial de la MT

8. Lista de problemas

- Dos objetos pueden estar en la misma localización física a la vez.
- El grupo de reglas para “Subir sobre <Objeto>“ está afectado por el problema anterior.
- La reglas Sobre_Objeto permitirá subir al mono sólo sobre objetos que están sobre el suelo. Esto puede ser muy restrictivo.
- Las reglas de que implementan el objetivo localización tienen que cambiar la localización de cualquier objeto que lleve el mono.

8. Ejercicios / Prácticas

- **Implementa y comprueba el grupo de reglas para el objetivo localización**
- **Escribe las reglas que indican el final de la ejecución:**
  - Todos los objetivos están satisfechos
  - No quedan reglas para ejecutar y quedan objetivos no satisfechos.
- **Comprueba el correcto funcionamiento del programa**

Código

Myb.dec

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa  iaaa iaaa iaaa iaaa;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Fichero myb.dec
;;; Problema clasico del mono y las bananas.
;;; Objetivo: Ilustra una metodologia de  programacion y depuracion
;;;           de un programa basado en reglas.
;;;
;;; Autor:
;;; Modificado: Jose Angel Banares
;;; Fecha: 17-2-1998
;;; Ficheros: myb.dec, myb.rul, myb.tst.
;;;
;;; Declaraciones para el problema del mono y las bananas
;;;
;;; Mono representa el animal que puede moverse y llevar cosas.
;;;    Debe haber una 'unica instancia en la memoria de trabajo.
;;; Cosa representa los objetos f'isicos del dominio. Debe haber
;;;    por lo menos una escalera en la memoria de trabajo.
;;; Objetivo representa los objetivos creados durante la resolucion
;;;    problema.
;;; CasoTest representa los casos de test para comprobar el programa
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa  iaaa iaaa iaaa iaaa;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(deftemplate mono

   (slot localizacion
      (type SYMBOL)
      (default nada))

   (slot sobre
      (type SYMBOL)
      (default suelo))

   (slot sostiene
      (type SYMBOL)
      (default nada)))

(deftemplate cosa

   (slot nombre
      (type SYMBOL)
      (default ?NONE))

   (slot localizacion
      (type SYMBOL)
      (default ?NONE))

   (slot sobre
      (type SYMBOL)
      (default nada))

   (slot peso
      (type SYMBOL)
      (allowed-symbols ligero pesado)
      (default ligero)))

(deftemplate objetivo

   (slot estado
```

Código

Myb.rul

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa  iaaa iaaa iaaa iaaa;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Fichero myb.rul
;;; Problema clasico del mono y las bananas.
;;; Objetivo: Ilustra una metodologia de  programacion y depuracion
;;;           de un programa basado en reglas.
;;; Autor:
;;; Modificado: Jose Angel Banares
;;; Fecha: 17-2-1998
;;; Ficheros: myb.dec, myb.rul, myb.tst.
;;;
;;; El programa se organiza alrededor de los grupos de reglas
;;; desarrollados para cada tipo de objetivo
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa  iaaa iaaa iaaa iaaa;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; ****************************************************************
;
; Grupo para TERMINACION
;
; Dise~no: Hay dos acciones para terminar
;            Enhorabuena (Todos los objetivos satisfechos)
;            Imposible (Los objetivos no pueden satisfacerse)
;
; ****************************************************************

(defrule Enhorabuena "FIN conseguido"
  (casoTest)
  (objetivo (estado satisfecho))
  (not (objetivo (estado activo)))
  =>
  (printout t  crlf "ENHORABUENA. Los objetivos se han satisfecho." crlf)
  (halt))

(defrule Imposible "Imposible"
  (objetivo (estado activo))
  =>
  (printout t  crlf "IMPOSIBLE. El objetivo no puede ser alcanzado." crlf)
  (halt))
; ****************************************************************
;
; Grupo para el Objetivo de tipo "SOBRE"
;
; Dise~no: Hay dos acciones
```

Código

Myb.rul

```
; Grupo para el Objetivo de tipo "SOBRE"
;
; Dise~no: Hay dos acciones
;            Salta sobre el suelo
;            Sube sobre <cosa>
;
; ************************************************************************
;Grupo para la accion SALTA SOBRE EL SUELO
; Sobre_Suelo implementa la accion de saltar al suelo si el mono
;   no esta en el suelo
;   Reglas que estan en conflicto siempre con esta: Imposible
; Sobre_Suelo_Satisfecho indica que el mono esta ya en el suelo
;   Reglas que estan en conflicto siempre con esta: Imposible
;------------------------------------------------------------------------
(defrule Sobre_Suelo "Salta sobre el suelo"
  ?objetivo <- (objetivo (estado activo) (tipo sobre) (argumentos suelo))
  ?mono <- (mono (sobre ?sobre&~suelo))
  =>
  (printout t "El mono salta desde " ?sobre " al suelo." crlf)
  (modify ?mono (sobre suelo))
  (modify ?objetivo (estado satisfecho)))

(defrule Sobre_Suelo_Satisfecho "Salta sobre el suelo ya esta satisfecho"
  ?objetivo <- (objetivo (estado activo) (tipo sobre) (argumentos suelo))
  ?mono <- (mono (sobre suelo))
  =>
  (printout t "El mono ya est'a ya sobre el suelo." crlf)
  (modify ?objetivo (estado satisfecho)))

;------------------------------------------------------------------------
;Grupo para la accion SUBE SOBRE COSA
; Sobre_Objeto implementa la accion de saltar sobre el objeto
;   Reglas que estan en conflicto siempre con esta: Imposible
; Sobre_Objeto_Sostiene Si el mono sostiene algo, debe dejarlo
;   Reglas que estan en conflicto siempre con esta:
;     Sostiene_Objeto_NoTecho_Sobre
; Sobre_Objeto_Localizacion Si el mono no esta en la misma localizacion
;   que el objeto, establece el objetivo mover a "localizacion"
;   Reglas que estan en conflicto siempre con esta: Imposible
; Sobre_Objeto_Satisfecho indica que el mono esta ya sobre el objeto
;   Reglas que estan en conflicto siempre con esta: Imposible
;------------------------------------------------------------------------
(defrule Sobre_Objeto "Sube directamente"
  ?objetivo <- (objetivo (estado activo) (tipo sobre) (argumentos ?obj))
  (cosa (nombre ?obj) (localizacion ?lugar) (sobre suelo))
  ?mono <- (mono (localizacion ?lugar)  (sobre ~?obj) (sostiene nada))
  =>
  (printout t "mono sube sobre " ?obj "." crlf)
  (modify ?mono (sobre ?obj))
```

Código

Myb.rul


```
;   Reglas que estan en conflicto siempre con esta: Imposible
; Sobre_Objeto_Sostiene Si el mono sostiene algo, debe dejarlo
;   Reglas que estan en conflicto siempre con esta:
;     Sostiene_Objeto_NoTecho_Sobre
; Sobre_Objeto_Localizacion Si el mono no esta en la misma localizacion
;   que el objeto, establece el objetivo mover a "localizacion"
;   Reglas que estan en conflicto siempre con esta: Imposible
; Sobre_Objeto_Satisfecho indica que el mono esta ya sobre el objeto
;   Reglas que estan en conflicto siempre con esta: Imposible
;-----------------------------------------------------------------------
(defrule Sobre_Objeto "Sube directamente"
  ?objetivo <- (objetivo (estado activo) (tipo sobre) (argumentos ?obj))
  (cosa (nombre ?obj) (localizacion ?lugar) (sobre suelo))
  ?mono <- (mono (localizacion ?lugar)  (sobre ~?obj) (sostiene nada))
  =>
  (printout t "mono sube sobre " ?obj "." crlf)
  (modify ?mono (sobre ?obj))
  (modify ?objetivo (estado satisfecho)))

(defrule Sobre_Objeto_Sostiene "Deja Objeto para subir"
  (objetivo (estado activo) (tipo sobre) (argumentos ?obj))
  (cosa (nombre ?obj) (localizacion ?lugar))
  (mono (localizacion ?lugar) (sostiene ~nada))
  (not (objetivo (estado activo) (tipo sostiene) (argumentos nada)))
  =>
  (assert (objetivo (estado activo) (tipo sostiene) (argumentos nada))))

(defrule Sobre_Objeto_Localizacion "Camina a nueva localizacion para subir sobre
  (objetivo (estado activo) (tipo sobre) (argumentos ?obj))
  (cosa (nombre ?obj) (localizacion ?lugar) (sobre suelo))
  (mono (localizacion ~?lugar))
  (not (objetivo (estado activo) (tipo localizacion) (argumentos ?lugar)))
  =>
  (assert (objetivo (estado activo) (tipo localizacion) (argumentos ?lugar))))

(defrule Sobre_Objeto_Satisfecho "Ya esta sobre el objeto"
  ?objetivo <- (objetivo (estado activo) (tipo sobre) (argumentos ?obj))
  (mono (sobre ?obj))
  =>
  (printout t "El mono ya est'a sobre el objeto " ?obj crlf)
  (modify ?objetivo (estado satisfecho)))

; ***************************************************************
;
; Grupo para el Objetivo de tipo "SOSTIENE"
;
; Dise~no: Hay dos acciones
;           Deja <objeto> -> Sostiene_Nada
```

Código

Myb.rul
Sigue ...


```
;
; ****************************************************************
;Grupo para la accion DEJA OBJETO
; Sostiene_Nada implementa la accion de dejar el objeto que sostiene
;   Reglas que estan en conflicto siempre con esta: Imposible
; Sostiene_Nada_Satisfecho Informa que el mono no sostiene nada
;   Reglas que estan en conflicto siempre con esta: Imposible
;---------------------------------------------------------------------
(defrule Sostiene_Nada "Deja el objeto que lleve"
  ?objetivo <- (objetivo (estado activo) (tipo sostiene)
                         (argumentos nada))
  ?mono <- (mono (localizacion ?lugar)
                 (sostiene ?obj&~nada))
  ?cosa <- (cosa (nombre ?obj))
  =>
  (printout t "mono deja " ?obj "." crlf)
  (modify ?objetivo (estado satisfecho))
  (modify ?mono (sostiene nada))
  (modify ?cosa (sobre suelo)))

(defrule Sostiene_Nada_Satisfecho "Deja el objeto que lleve ya satisfecho"
  ?objetivo <- (objetivo (estado activo) (tipo sostiene)
                         (argumentos nada))
  ?mono <- (mono (localizacion ?lugar) (sobre ?obj2)
                 (sostiene nada))
  =>
  (printout t "El mono no sostiene nada." crlf)
  (modify ?objetivo (estado satisfecho)))
;---------------------------------------------------------------------
;Grupo para la accion COGE OBJETO del TECHO
; Sostiene_Objeto_Techo regla general, implementa la accion de coger
;   objeto - el mono esta sobre la escalera directamente bajo el objeto
;   a coger y no sostiene nada.
;   Reglas que estan en conflicto siempre con esta: Imposible
; Sostiene_Objeto_Techo_Sobre Si la escalera esta bajo el objeto a coger,
;   establece el objetivo de estar sobre la escalera.
;   Reglas que estan en conflicto siempre con esta:
;      Sostiene_Objeto_Sostiene. Pero siempre se escoge
;      Sostiene_Objeto_Techo_Sobre por ser mas espec'ifica.
;   NOTA: La regla no menciona la localizacion del mono. el objetivo
;         "sobre" establecer'a el objetivo de mover el mono si procede.
; Sostiene_Objeto_Techo_Localizacion Si la escalera no esta debajo del
;   objeto a coger, establece el objetivo de mover la escalera.
;   Reglas que estan en conflicto siempre con esta: Imposible
;   NOTA: La regla no menciona el mono, poque hasta que la escalera no
;      esta bajo el objeto, el mono no es importante.
;---------------------------------------------------------------------
```

Código

Myb.tst

http://diana.cps.unizar.es/banares/IC/practicas/P1/myb.tst - Microsoft Internet Explorer

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa  iaaa iaaa iaaa iaaa;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Fichero myb.tst
;;; Problema clasico del mono y las bananas.
;;; Objetivo: Ilustrat una metodologia de  programacion y depuracion
;;;           de un programa basado en reglas.
;;; Autor:
;;; Modificado: Jose Angel Banares
;;; Fecha: 17-2-1998
;;; Ficheros: myb.dec, myb.rul, myb.tst.
;;;
;;; Conjunto de tests para el problema del mono y las bananas
;;; Las reglas estan organizadas por los grupos que fueron dise~nados.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa iaaa  iaaa iaaa iaaa iaaa;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; **********************************************************************
;
; Grupo de test para el Objetivo de tipo "SOBRE"
;
; Dise~no: Dos acciones, Salta sobre el suelo, Sube sobre <cosa>
;
; SALTA_SOBRE_SUELO
;   Test_Sobre_Suelo,  Comprueba que el mono saltara sobre el suelo
;   Test_Sobre_Suelo_Satisfecho  Comprueba que satisfacen condiciones
; SUBE_SOBRE_COSA
;   Test_Sobre_Objeto comprueba si el mono subira a un objeto
;   Test_Sobre_Objeto_Sostiene comprueba que se crea el objetivo dejar'a
;     un objeto
;   Test_Sobre_Objeto_Localizacion comprueba que se crea el objetivo
;     localizacion
;   Test_Sobre_Objeto_Satisfecho comprueba condiciones satisfechas
; **********************************************************************
; ----------------------------------------------------------------------
; SALTA_SOBRE_SUELO
```

Código

Myb.tst
sigue ...

http://diana.cps.unizar.es/banares/IC/practicas/P1/myb.tst - Microsoft Internet Explorer

Archivo Edición Ver Favoritos Herramientas Ayuda

Atrás Búsqueda Favoritos Historial

Dirección http://diana.cps.unizar.es/banares/IC/practicas/P1/myb.tst Ir a

Vínculos - Asignatura IAIC1 Universidad de Zaragoza - Inteligencia Artificial Hotmail gratuito

```
; SALTA_SOBRE_SUELO
; ------------------------------------------------------------------------
(defrule Test_Sobre_Suelo ""
  (casoTest (tipo sobre) (nombre suelo))
  =>
  (assert (cosa (nombre escalera) (sobre suelo)
                (localizacion t5-7) (peso ligero)))
  (assert (mono (sobre escalera) (localizacion t5-7)))
  (assert (objetivo (estado activo) (tipo sobre) (argumentos suelo))))

(defrule Test_Sobre_Suelo_Satisfecho ""
 (casoTest (tipo sobre) (nombre Suelo_Satisfecho))
  =>
  (assert (cosa (nombre escalera) (sobre suelo)
                (localizacion t5-7) (peso ligero)))
  (assert (mono (sobre suelo) (localizacion t5-7)))
  (assert (objetivo (estado activo) (tipo sobre) (argumentos suelo))))
; ------------------------------------------------------------------------
; SUBE_SOBRE_COSA
; ------------------------------------------------------------------------
(defrule Test_Sobre_Objeto ""
 (casoTest (tipo sobre) (nombre Objeto))
  =>
  (assert (cosa (nombre escalera) (sobre suelo)
                (localizacion t5-5) (peso ligero)))
  (assert (mono (sobre suelo) (localizacion t5-5)))
  (assert (objetivo (estado activo) (tipo sobre) (argumentos escalera))))

(defrule Test_Sobre_Objeto_Sostiene ""
 (casoTest (tipo sobre) (nombre Sostiene_Objeto))
  =>
  (assert (cosa (nombre escalera) (sobre suelo)
                (localizacion t5-5) (peso ligero)))
  (assert (mono (sobre suelo) (localizacion t5-5) (sostiene escalera)))
  (assert (objetivo (estado activo) (tipo sobre) (argumentos escalera))))

(defrule Test_Sobre_Objeto_Localizacion ""
 (casoTest (tipo sobre) (nombre Localizacion_Objeto))
  =>
  (assert (cosa (nombre escalera) (sobre suelo)
                (localizacion t6-5) (peso ligero)))
```

Angel Bañares

Ejemplo clips

TextPad - [I:\pub\personal\jangelb\docencia\ISBC\transparencias\

File Edit Search View Tools Macros Configure Window Help

```
;;;======================================================
;;;   Monkees and Bananas Sample Problem
;;;
;;;     This is an extended version of a
;;;     rather common AI planning problem.
;;;     The point is for the monkee to find
;;;     and eat some bananas.
;;;
;;;     CLIPS Version 6.0 Example
;;;
;;;     To execute, merely load, reset and run.
;;;======================================================

;;;**************
;;;* TEMPLATES *
;;;**************

(deftemplate monkey
   (slot location
      (type SYMBOL)
      (default green-couch))
   (slot on-top-of
      (type SYMBOL)
      (default floor))
   (slot holding
      (type SYMBOL)
      (default nothing)))

(deftemplate thing
   (slot name
      (type SYMBOL)
      (default ?NONE))
   (slot location
      (type SYMBOL)
      (default ?NONE))
   (slot on-top-of
      (type SYMBOL)
      (default floor))
```

TextPad - [I:\pub\personal\jangelb\docencia\ISBC\transparencias\1.Reglas...

File Edit Search View Tools Macros Configure Window Help

```
      (default light)))

(deftemplate chest
   (slot name
      (type SYMBOL)
      (default ?NONE))
   (slot contents
      (type SYMBOL)
      (default ?NONE))
   (slot unlocked-by
      (type SYMBOL)
      (default ?NONE)))

(deftemplate goal-is-to
   (slot action
      (type SYMBOL)
      (allowed-symbols hold unlock eat move on walk-to)
      (default ?NONE))
   (multislot arguments
      (type SYMBOL)
      (default ?NONE)))

;;;***************************
;;;* CHEST UNLOCKING RULES *
;;;***************************

(defrule hold-chest-to-put-on-floor ""
  (goal-is-to (action unlock) (arguments ?chest))
  (thing (name ?chest) (on-top-of ~floor) (weight light))
  (monkey (holding ~?chest))
  (not (goal-is-to (action hold) (arguments ?chest)))
  =>
  (assert (goal-is-to (action hold) (arguments ?chest))))

(defrule put-chest-on-floor ""
  (goal-is-to (action unlock) (arguments ?chest))
  ?monkey <- (monkey (location ?place) (on-top-of ?on) (holdin
  ?thing <- (thing (name ?chest))
  =>
  (printout t "Monkey throws the " ?chest " off the "
              ?on " onto the floor." crlf)
  (modify ?monkey (holding blank))
  (modify ?thing (location ?place) (on-top-of floor)))

(defrule get-key-to-unlock ""
  (goal-is-to (action unlock) (arguments ?obj))
  (thing (name ?obj) (on-top-of floor))
  (chest (name ?obj) (unlocked-by ?key))
  (monkey (holding ~?key))
  (not (goal-is-to (action hold) (arguments ?key)))
  =>
  (assert (goal-is-to (action hold) (arguments ?key))))
```

Ejemplo Clips

```
  (assert (goal-is-to (action on) (arguments ?on))))

(defrule climb-directly ""
  ?goal <- (goal-is-to (action on) (arguments ?obj))
  (thing (name ?obj) (location ?place) (on-top-of ?on))
  ?monkey <- (monkey (location ?place) (on-top-of ?on) (holdin
  =>
  (printout t "Monkey climbs onto the " ?obj "." crlf)
  (modify ?monkey (on-top-of ?obj))
  (retract ?goal))

(defrule already-on-object ""
  ?goal <- (goal-is-to (action on) (arguments ?obj))
  (monkey (on-top-of ?obj))
  =>
  (retract ?goal))

;;;*********************
;;;* EAT OBJECT RULES *
;;;*********************

(defrule hold-to-eat ""
  (goal-is-to (action eat) (arguments ?obj))
  (monkey (holding ~?obj))
  (not (goal-is-to (action hold) (arguments ?obj)))
  =>
  (assert (goal-is-to (action hold) (arguments ?obj))))

(defrule satisfy-hunger ""
  ?goal <- (goal-is-to (action eat) (arguments ?name))
  ?monkey <- (monkey (holding ?name))
  ?thing <- (thing (name ?name))
  =>
  (printout t "Monkey eats the " ?name "." crlf)
  (modify ?monkey (holding blank))
  (retract ?goal ?thing))

;;;************************
;;;* INITIAL STATE RULE *
;;;************************

(defrule startup ""
  =>
  (assert (monkey (location t5-7) (on-top-of green-couch) (hol
  (assert (thing (name green-couch) (location t5-7) (weight he
  (assert (thing (name red-couch) (location t2-2) (weight heav
  (assert (thing (name big-pillow) (location t2-2) (on-top-of
  (assert (thing (name red-chest) (location t2-2) (on-top-of b
  (assert (chest (name red-chest) (contents ladder) (unlocked-
  (assert (thing (name blue-chest) (location t7-7) (on-top-of
  (assert (chest (name blue-chest) (contents bananas) (unlocke
  (assert (thing (name blue-couch) (location t8-8) (weight hea
  (assert (thing (name green-chest) (location t8-8) (on-top-of
  (assert (chest (name green-chest) (contents blue-key) (unloc
  (assert (thing (name red-key) (location t1-3)))
  (assert (goal-is-to (action eat) (arguments bananas))))
```