PROBLEMA: (2 puntos) Junio 2003

1) Se pretende resolver mediante un **sistema basado en reglas** el problema del juego del comecocos simplificado. Para ello se dispone de los siguientes tipos de hechos representados mediante patrones:

(**comecocos posX** <x> **posY** <y> **contador** <c> **Vida** <v>)

Sólo existe un hecho de este tipo en cada instante en la Base de Hechos, para indicar la posición actual del comecocos (mediante los valores de sus coordenadas <x>, e <y>), y los contadores del número de frutas que se ha comido (<c>) y el del número de vidas que le quedan antes de perder la partida (<v>).

(**fantasma posX** <x> **posY** <y>)

Existen tantos hechos de este tipo como fantasmas en el juego, e indican la posición en la que se encuentra en este momento el fantasma en cuestión (mediante los valores de sus coordenadas <x>, e <y>).

(**fruta posX** <x> **posY** <y>)

Existen tantos hechos de este tipo como frutas en el juego, e indican la posición en la que se encuentra en este momento la fruta en cuestión (mediante los valores de sus coordenadas <x>, e <y>).

Implementad las siguientes reglas:

a) **COMER**: Esta regla controla si el comecocos se encuentra en la misma posición que una fruta. Si es así, permite comérsela, incrementando el contador de frutas que se ha comido el comecocos en 1, y eliminando dicha fruta del juego.

b) **MORIR**: Esta regla controla si el comecocos se encuentra en la misma posición que un fantasma. En este caso, se decrementa en 1 el número de vidas del comecocos.

c) **GANAR**: Esta regla controla cuando acaba de forma victoriosa el juego, porque el comecocos se ha comido 10 ó más frutas, y avisa al usuario de que ha ganado.

d) **GAMEOVER**: Esta regla controla cuando acaba perdiendo el comecocos, porque ha consumido todas sus vidas, y avisa al usuario de que ha perdido.

NOTA: Cuando el juego finaliza, bien sea por una victoria o por una derrota del comecocos, ya no pueden ser aplicables ninguna de las reglas.

PROBLEMA: (2 puntos) Septiembre 2003

2) Se pretende resolver mediante un **sistema basado en reglas** el problema de la gestión del carro de compra en un hipermercado. Para ello se dispone de los siguientes **tipos** de hechos representados mediante patrones:

(**pasillo** <número> **estante** <nEstante> **producto** <Nombre> **cantidad** <stock> **precio** <valor>)
Existirá al menos un hecho de este tipo por cada pasillo del hipermercado, para indicar la disposición de los distintos productos. Para ello se indicará <número> como un valor del 1 al 12 para indicar el pasillo, <nEstante> como un valor del 1 al 4 para identificar uno de los estantes del pasillo, <Nombre> para indicar el nombre del producto, <stock> para la cantidad almacenada, y <precio> para el valor por cada unidad del producto.

(**lista-compra** <producto> <cantidad>
<producto> <cantidad>
<producto> <cantidad>
…
<producto> <cantidad>)
Existe un único hecho de este tipo en el sistema (pues sólo vamos a considerar un cliente de forma simultánea), e indica los productos (con las cantidades de cada uno) que aún le faltan al usuario por comprar).

(**carro-compra pasillo** <número> **cuenta** <valor>
<producto> <cantidad>
<producto> <cantidad>
…
<producto> <cantidad>)
Existe un único hecho de este tipo en el sistema (pues sólo vamos a considerar un cliente de forma simultánea), e indica la evolución del carro de la compra del cliente, indicando su posición (mediante el <número> del pasillo), el coste total de los productos en el carro (<valor>), y la lista de dichos productos (con las cantidades de cada uno) que ya están en el carro.

Utilizando el lenguaje **CLIPS** implementad las siguientes reglas:

e) **MOVER**: Esta regla controla el movimiento del carro, pasando del pasillo en el que se encuentra en este momento el carro al siguiente (suponiendo que cuando el carro se encuentra en el último pasillo –el 12– pasa de dicho pasillo al 1).

f) **COMPRAR**: Esta regla controla si el carro se encuentra en un pasillo en el que exista algún producto que aún esté en la lista de la compra, y existe suficiente stock para la cantidad que quiere comprar el usuario. En este caso, se borra de la lista de la compra el producto junto con la cantidad que se ha conseguido, al igual que del pasillo que toque se decrementará la cantidad en stock en el número que se compra. Por último se incluye en el carro de la compra el producto junto con la cantidad que se compra, y se incrementa el valor de los productos almacenados en el carro con lo que acabamos de añadirle.

g) **PAGAR**: Esta regla controla cuando acaba de recolectar el cliente todos los productos, indicándole por pantalla el valor de lo que tiene que pagar.

h) **NOHAY**: Esta regla controla cuando existe algún producto de la lista de la compra del que no queda suficiente stock para servir lo que quiere comprar el cliente.

NOTA: Cuando la compra finaliza, bien sea porque consigue incluir en el carro todos los productos de la lista de la compra, o bien porque encuentra algún producto del que no queda suficiente stock, ya no pueden ser aplicables ninguna de las reglas.

PROBLEMA: (2 puntos) Junio 2004

3) Se pretende gestionar parte de una librería con un **sistema basado en reglas**, de forma que la base de hechos de dicho sistema estará compuesta por hechos representados mediante patrones con la siguiente estructura:

(**libro** [nuevo] **título** <tit> **autor** <nombre> **tipo** <tip> **ejemplares** <nº>)

Cada libro de la tienda tendrá un hecho de este tipo asociado. En este hecho se recoge el título del libro (el valor <tit> que aparece después de la palabra **título**), el nombre de su autor (<nombre>) el tipo de contenido del mismo (<tip>) y la cantidad de ejemplares de que se dispone en la librería (<nº>). De forma **opcional**, puede aparecer en el hecho la palabra ***nuevo*** para indicar que la información relativa a dicho libro no se ha computado en el hecho relativo a su autor.

(**autor** [masivo] [diverso] **nombre** <nombre> **tipo** <lista_tip>+ **libros** <cont>)

Existirá un hecho de este tipo por cada autor del que se tengan libros en la librería. De esta manera se indicará con este hecho el nombre (con el valor <nombre>), los diferentes tipos de libros que ha escrito (con los valores que aparecerán en la lista de valores <lista_tip>+) y la cantidad de libros distintos que se tienen en la librería que ha escrito dicho autor (<cont>). De forma **opcional**, pueden aparecer en el hecho las palabras ***masivo*** y/o ***diverso*** para indicar si el autor ha escrito muchos libros que hay en la librería o si ha escrito muchos libros de tipos diferentes.

Con estos tipos de hechos, podríamos tener hechos en la base de hechos como los siguientes:

(libro título “Fundación” autor “Isaac Asimov” tipo “ciencia ficción” pvp 25 ejemplares 5)
(libro título “La Nueva Guía de la Ciencia” autor “Isaac Asimov” tipo “divulgación” pvp 30 ejemplares 3)
(libro título “Robots e Imperio” autor “Isaac Asimov” tipo “ciencia ficción” pvp 30 ejemplares 2)
(libro nuevo título “Hyperion” autor “Dan Simmons” tipo “ciencia ficción” pvp 25 ejemplares 5)
…
(autor diverso nombre “Isaac Asimov” tipo “ciencia ficción” “divulgación” libros 3)
(autor nombre “Dan Simmons” tipo “ciencia ficción” libros 1)

Completad las siguientes reglas:

i) **NUEVO AUTOR**: Esta regla debe controlar que si aparece un nuevo hecho de tipo libro ***nuevo*** referente a un autor que aún no existe en la base de hechos se genere un hecho identificándolo. Este hecho incluirá el nombre del autor el tipo del libro en cuestión, y después de la palabra *libros* un 1 (para indicar que sólo existe ese libro de ese autor). Además, después de aplicarse esta regla, el libro deja de considerarse ***nuevo***.

(defrule NUEVOAUTOR
?f <- (libro …

(not (…

=>
(retract …

(assert …

(…

(printout t “El autor “ ?x " acaba de ser dado de alta en la base de hechos” crlf)
)

j) **AUTOR CONOCIDO**: Esta regla debe controlar que si aparece un nuevo hecho de tipo libro ***nuevo*** referente a un autor que ya existe en la base de hechos se modifique el hecho que lo identifica. Este hecho incluirá el nombre del autor el tipo del libro en cuestión, e incrementará en 1 la cantidad que aparece después de la palabra *libros*. Además, después de aplicarse esta regla, el libro deja de considerarse ***nuevo***.

```
(defrule AUTORCONOCIDO
        ?f1 <- (libro  …


        ?f2 <- …


        =>
        (retract …


        (assert …


        (…


        (printout t “La información del autor “ ?x " acaba de ser actualizada” crlf)
)
```

k) **AUTOR DIVERSO**: Esta regla controla que un autor existente en la base de hechos debe pasar a considerarse como ***diverso***, pues existen libros suyos de al menos dos tipos distintos en la librería.

```
(defrule AUTORDIVERSO
        ?f1 <- (autor  …


        (…


        =>
        (retract …


        (assert …


        (printout t “La información del autor “ ?x " acaba de ser actualizada” crlf)
)
```

l) **AUTOR MASIVO**: Esta regla controla que un autor existente en la base de hechos debe pasar a considerarse como ***masivo***, pues existen más de 20 libros distintos suyos en la librería.

```
(defrule AUTORMASIVO
        ?f1 <- (autor  …


        (…


        =>
        (retract …


        (assert …


        (printout t “La información del autor “ ?x " acaba de ser actualizada” crlf)
)
```

PROBLEMA: (2 puntos) Septiembre 2004

4) Se pretende gestionar la construcción / adaptación de la parrilla de programación de la emisora **SRPTV**, de forma que la base de hechos de dicho sistema estará compuesta por hechos representados mediante patrones con la siguiente estructura:

(**programa nombre** <tit> **cadena** <nombre> **tipo** <prio> **duración** <dur>)

Cada programa sobre el cual se quiera guardar información tendrá un hecho de este tipo, indicando el nombre del programa (<tit>), el de la cadena que tiene los derechos de emisión del mismo (<nombre>), el tipo (<prio>) que podrá ser ***"prime time"***, ***"informativos"***, ***"deportes"*** o ***"relleno"***. Por último, aparece, tras la palabra **duración**, el número de minutos que dura (<dur>).

(**parrilla cadena** <cad> **nombre** <tit> **día** <dia_semana> **hora_comienzo** <hora> **minuto_comienzo** <min>)

Cada programa de cada cadena que se esté emitiendo deberá tener un hecho de este tipo, para indicar cuándo y dónde se emite. Así, un hecho de este tipo almacena el nombre de la cadena que lo emite (el valor <cad> que aparece después de la palabra **cadena**), el del programa (<tit>), el día de la emisión (<dia_semana>) que será un valor de entre los posibles **L**, **M**, **X**, **J**, **V**, **S** o **D**. Por último se indicará la hora y el minuto en el que comienza dicho programa (en los valores <hora> y <min> respectivamente).

Con estos tipos de hechos, el sistema almacenará conocimiento no sólo de los programas y parrilla de nuestra cadena, sino también de los del resto de cadenas, pudiendo tener hechos en la base de hechos como los siguientes:

(programa nombre "Hoy Examen" tipo "Prime Time" duración 180)
(parrilla cadena SRPTV nombre "Hoy Examen" día J hora_comienzo 16 minuto_comienzo 0)
(programa nombre "Dark Angel" tipo "Relleno" duración 60)
(parrilla cadena A3 nombre "Dark Angel" día J hora_comienzo 0 minuto_comienzo 5)

Completad las siguientes reglas:

m) **ASIGNAR HORARIO PRIME TIME**: Esta regla permite asignar un horario nuevo a un programa de tipo ***Prime Time***. Este tipo de programas empezará sobre las 22:00, teniendo que evitar aquellos días en los que otra cadena tenga ya un programa de este tipo en dicha franja horaria (por simplificar, controlaremos tan sólo que no exista otro programa que empiece exactamente a las 22:00h.).

(defrule ASIGNAR_PRIME_TIME
(programa nombre …

(**not** (**parrilla …**

(**not** (**and** (**parrilla** …

(**programa** …

=>
(**assert** …

(printout t "El programa " ?x " ha entrado en la parrilla de nuestra cadena" crlf)
)

n) **CAMBIAR HORARIO PRIME TIME**: Si otra cadena coloca un programa ***Prime Time*** en la misma franja horaria en la que nosotros tenemos un programa de este tipo, cambiaremos el programa a otro día, siempre que quede algún día en la parrilla sin un programa de ***Prime Time*** en el horario de las 22:00 horas. De forma similar al caso anterior, consideraremos que se encuentran en la misma franja horaria dos programas que comiencen exactamente a la misma hora (en este caso las 22:00 horas).

(defrule CAMBIAR_PRIME_TIME
(**programa nombre** ...

?f <- (**parrilla cadena** …

(**parrilla cadena** …

(test (neq …

(**programa nombre** ...

(**not (and** (**parrilla cadena** …

(**programa …**

=>
(retract …

(assert …

(printout t “El programa “ ?x " ha entrado en la parrilla de nuestra cadena” crlf)
)

o) **ASIGNAR HORARIO RELLENO**: Si otra cadena coloca un programa ***Prime Time*** en la misma franja horaria en la que nosotros tenemos un programa de este tipo, cambiaremos el programa por uno del tipo "***relleno***". De forma similar a los casos anteriores, consideraremos que se encuentran en la misma franja horaria dos programas que comiencen exactamente a la misma hora (en este caso las 22:00 horas).

(defrule ASIGNAR_HORARIO_RELLENO
(programa …

(parrilla …

(test (neq …

(programa …

?f <- (parrilla …

(**programa** …

(**not** (**parrilla** …

=>
(**retract** …

(**assert** (**parrilla** …

(printout t "El programa " ?x " ha entrado en la parrilla de nuestra cadena como relleno" crlf)
)

p) **AVISAR DEPORTES**: Siempre que un programa que no sea de tipo "***relleno***" sea puesto en la parrilla por otra cadena en el mismo horario en el que tenemos nosotros un programa de deportes, esta regla mostrará un aviso por pantalla.

(defrule AVISAR_DEPORTES
(programa …

(test (neq …

(test (neq …

(parrilla …

(programa …

(parrilla …

=>
(printout t "El programa " ?x " de la cadena "?cad" solapa con deportes" crlf)
)