Tema 1: Programación basada en reglas con CLIPS

Ejercicio 1.1. Se considera la siguiente base de conocimiento en CLIPS:

```
(defrule regla
  ?h1 <- (resultado $?r)
  ?h2 <- (datos ?x $?d)
  (not (datos ?y&:(< ?y ?x) $?))
  =>
  (retract ?h1 ?h2)
  (assert (resultado $?r ?x)
          (datos ?d)))

(deffacts hechos
  (datos -1 2 5)
  (datos 0 3)
  (resultado))
```

1. Escribir la tabla de seguimiento de su ejecución e indicar los hechos que quedan finalmente en memoria.
2. Explicar brevemente qué condiciones han de cumplir los hechos `(datos $?)` para que el programa anterior tenga sentido.
3. Explicar brevemente qué valor se almacena en el hecho `(resultado $?)` cuando se parte de cualquier cantidad de hechos `(datos $?)`.
4. Construir el predicado `resultado` en Prolog, de forma que a partir de dos datos como los del ejemplo anterior, construya el mismo resultado que se obtiene con la base de conocimiento en CLIPS.

Ejercicio 1.2. Se considera la siguiente base de conocimiento en CLIPS:

```
(defrule regla
  (datos $?ini ?x $?fin)
  =>
  (assert (datos $?ini $?fin)))

(deffacts datos
  (datos 1 2))
```

1. Escribir la tabla de seguimiento de su ejecución e indicar los hechos que quedan finalmente en memoria.
2. ¿Afecta al resultado final el orden en que se disparan las reglas simultáneamente activas?. En caso afirmativo escribir dos tablas de seguimiento en las que, como consecuencia del orden de ejecución de las reglas simultáneamente activas, los conjuntos de hechos finales sean distintos (salvo numeración de los hechos). En caso negativo, explicar razonadamente por qué.

3. Explicar brevemente el significado del programa CLIPS anterior.

4. Construir el predicado `ejercicio` en Prolog, que tenga el mismo efecto que el programa CLIPS anterior. En el ejemplo que se proporciona, la llamada al predicado `ejercicio` será: `ejercicio([1,2],R)` y como resultado de su ejecución en `R` se tendrá que almacenar la lista formada por todos los hechos del tipo `(dato $?)` (representados por la lista `[$?]`), que queden en la base de conocimiento al final de la ejecución: Si después de la ejecución del programa CLIPS los hechos de la base de conocimiento son `(dato 1)` y `(dato 1 2)`, el resultado almacenado en `R` ha de ser `[[1],[1,2]]` o `[[1,2],[1]]`

Ejercicio 1.3. Consideremos un conjunto de variables que pueden tomar un valor dentro de un dominio prefijado y un conjunto de restricciones que establecen ciertas condiciones que tienen que cumplir los valores de dichas variables, por ejemplo:
Variables: A toma valores en el dominio $\{1, 2, 3\}$.
B toma valores en el dominio $\{1, 2, 4\}$.
C toma valores en el dominio $\{1, 3, 4\}$.

Restricciones: El valor de A tiene que ser menor que el de B.
El valor de B tiene que ser menor que el de C.

Los dominios de las variables se pueden reducir analizando las restricciones que las relacionan, mediante un algoritmo que se llama "Algoritmo de consistencia de arcos". El proceso básico dentro de dicho algoritmo consiste en eliminar un valor v del dominio de una variable X, si existe una restricción que relacione X con otra variable Y, para la que no exitan valores que cumplan dicha restricción con respecto a v.
En el ejemplo propuesto, la segunda restricción relaciona los valores de las variables B y C, y para el valor 4 de B, no existe ningún valor en el dominio de C cumpliendo la restricción, por tanto podemos quitar el valor 4 del dominio de B. Similarmente, para el valor 1 de C, no existe ningún valor en el dominio de B cumpliendo la restricción, por tanto también podemos quitar el valor 1 del dominio de B.
Se considera definida la función `(funcion-tipo ?tipo ?v1 ?v2)`, con la que podemos comprobar que los valores `?v1` y `?v2` cumplen una restricción del `?tipo`. Por ejemplo `(funcion-tipo menor ?v1 ?v2)` comprueba `?v1<?v2`.
Se pide:

1. Construir las plantillas `variable`, donde se almacena el nombre y dominio de las variables, y `restriccion`, donde se almacena el tipo de restricción (`distinto`, `menor`, `mayor`, ...) y los nombres de las variables relacionadas.

2. Construir un conjunto de hechos iniciales que contenga las variables y las restricciones del ejemplo.

3. Construir reglas CLIPS para implementar el proceso de reducción de los dominios de las variables como consecuencia del análisis de las restricciones. Este conjunto de reglas tiene que servir para cualquier conjunto de variables y restricciones, no sólo las del ejemplo, siempre que la función `funcion-tipo`, contemple el tipo de las restricciones consideradas.

Se valorará la simplicidad de la solución proporcionada en lo referente a nuevas estructuras (plantillas) y número de reglas necesarias.

[**Nota:** *Para el desarrollo de este ejercicio no se permite el uso de condicionales (*`if ... then ... else`*), el uso de bucles (*`while ... do` *o* `loop-for-count`*), la definición de nuevas funciones (*`deffunction`*) ni el uso de prioridades (*`salience`*)*]

Ejercicio 1.4. Se considera la siguiente base de conocimiento en CLIPS:

```
(defrule regla1
  ?h1 <- (dato1 $?i1 ?x $?f1)
  ?h2 <- (dato2 $?i2 ?x $?f2)
  =>
  (retract ?h1 ?h2)
  (assert (dato1 $?i1 $?f1)
          (dato2 $?i2 $?f2)))

(defrule regla2
  ?h1 <- (dato1 $? ?x $?)
  (not (dato2 $? ?x $?))
  ?h2 <- (dato2 $?)
  =>
  (retract ?h1 ?h2)
  (assert (respuesta NO)))

(defrule regla3
  ?h1 <- (dato1)
  ?h2 <- (dato2)
  =>
  (retract ?h1 ?h2)
  (assert (respuesta SI)))
```

Se pide:

1. Construir una tabla de seguimiento con el siguiente conjunto de hechos iniciales. ¿Qué hechos quedan en la base de conocimiento al terminar la ejecución?.

   ```
   (deffacts ej1
     (dato1 1 2 3 1)
     (dato2 2 1 1 3))
   ```

2. Construir una tabla de seguimiento con el siguiente conjunto de hechos iniciales. ¿Qué hechos quedan en la base de conocimiento al terminar la ejecución?.

   ```
   (deffacts ej1
     (dato1 1 2 3 1)
     (dato2 2 1 2 3))
   ```

3. Explicar brevemente el comportamiento del conjunto de reglas anterior.

4. Existe alguna situación en la que, a partir de cierto conjunto de hechos iniciales del tipo `(dato1 $?)` y `(dato2 $?)`, al terminar la ejecución no aparezca en el conjunto de hechos resultantes ni `(respuesta NO)` ni `(respuesta SI)`. Si es así, proporcionar un conjunto de

hechos iniciales con los que ocurra esto y construir la correspondiente tabla de seguimiento. Proporcionar una regla `regla4` que arregle esta situación manteniendo el comportamiento de la base de conocimiento.

Ejercicio 1.5. Se considera la siguiente base de conocimiento en CLIPS:

```
(defrule r1
  ?h <- (numeros $?i ?x $?m ?y&:(= (mod ?x ?y) 0) $?f)
  =>
  (retract ?h)
  (assert (numeros $?i $?m ?y $?f)))

(defrule r2
  ?h <- (numeros $?i ?x $?m ?y&:(= (mod ?y ?x) 0) $?f)
  =>
  (retract ?h)
  (assert (numeros $?i ?x $?m $?f)))
```

Se pide:

1. Construir una tabla de seguimiento con el siguiente conjunto de hechos iniciales:

   ```
   (deffacts ej1
     (numeros 2 3 4 5 6 7 8 9))
   ```

   En la tabla de seguimiento se tienen que incluir TODAS las activaciones y desactivaciones de las reglas `r1` y `r2`, indicando para cada una de ellas los valores que toman las variables `$?i`, `?x`, `$?m`, `?y` y `$?f`. ¿Qué hechos quedan en la base de conocimiento al terminar la ejecución ?.

2. Construir una tabla de seguimiento con el siguiente conjunto de hechos iniciales:

   ```
   (deffacts ej2
     (numeros 6 2 5 4 3 2 7))
   ```

   En la tabla de seguimiento se tienen que incluir TODAS las activaciones y desactivaciones de las reglas `r1` y `r2`, indicando para cada una de ellas los valores que toman las variables `$?i`, `?x`, `$?m`, `?y` y `$?f`. ¿Qué hechos quedan en la base de conocimiento al terminar la ejecución?.

3. Explicar brevemente el comportamiento del conjunto de reglas anterior.

Ejercicio 1.6. Consideremos fórmulas de la lógica proposicional construidas con las conectivas negación (`-`), disyunción (`\/`), conjunción (`/\`), implicación (`->`) e equivalencia (`<->`). Ejemplos de estas fórmulas son los siguientes:
$F_1 : (p \wedge q) \rightarrow r$
$F_2 : ((\neg q) \vee (\neg r)) \leftrightarrow (\neg(q \wedge r))$
Para representar estás fórmulas en CLIPS utilizamos la plantilla `formula`:

```
(deftemplate formula
  (slot id)
  (slot tipo)
  (multislot hijos))
```

En esta plantilla el campo `id` almacena un identificador asociado a la fórmula que se esta representando, el campo `tipo` indica la conectiva principal de la fórmula (si se trata de una variable este campo contendrá el símbolo correspondiente a esta variable, en otro caso contendrá la cadena `-`, `/\`, `\/`, `->` o `<->`, según corresponda) y el campo `hijos` almacena los identificadores de las subfórmulas (tantos como corresponda a la conectiva principal y vacío en el caso de representar una variable). Por ejemplo, la fórmula $(p \wedge q) \rightarrow r$ se puede representar con los siguientes hechos:

```
(deffacts formula-1
  (formula (id id1) (tipo ->) (hijos id2 id3))
  (formula (id id2) (tipo /\) (hijos id4 id5))
  (formula (id id3) (tipo r) (hijos))
  (formula (id id4) (tipo p) (hijos))
  (formula (id id5) (tipo q) (hijos)))
```

Toda fórmula proposicional se puede transformar a *forma normal negativa* aplicando las siguientes reglas de transformación:

$$
\begin{array}{lcl}
(X \leftrightarrow Y) & \Longrightarrow & (X \rightarrow Y) \wedge (Y \rightarrow X) \\
(X \rightarrow Y) & \Longrightarrow & (\neg X) \vee Y \\
(\neg(X \wedge Y)) & \Longrightarrow & (\neg X) \vee (\neg Y) \\
(\neg(X \vee Y)) & \Longrightarrow & (\neg X) \wedge (\neg Y) \\
(\neg(\neg X)) & \Longrightarrow & X
\end{array}
$$

Donde X e Y son dos fórmulas cualesquiera.
Se pide:

1. Construir un conjunto de hechos para representar la fórmula $F_2 : ((\neg q) \vee (\neg r)) \iff (\neg(q \wedge r))$.

2. Construir un conjunto de reglas CLIPS que sirvan para transformar una fórmula proposicional en forma normal negativa aplicando las reglas de transformación anteriores. (**Nota:** para generar símbolos nuevos para los identificadores de las fórmulas se puede utilizar la función `(gensym)`)

[**Nota:** *Para el desarrollo de este ejercicio no se permite el uso de condicionales (*`if ... then ... else`*), el uso de bucles (*`while ... do`* o* `loop-for-count`*), la definición de nuevas funciones (*`deffunction`*) ni el uso de prioridades (*`salience`*)*]
