Inteligencia Artificial II **(9–Junio–2000)**

Ejercicio 1 [4 puntos]

Dada una base de conocimiento definida puede calcularse el conjunto de sus consecuencias por encadenamiento hacia adelante. Por ejemplo, a partir de la base de conocimiento

```
a <- [b, c].  % R1
b <- [d, e].  % R2
b <- [g, e].  % R3
c <- [e].     % R4
d <- [].      % R5
e <- [].      % R6
f <- [a, g].  % R7
```

se obtiene la siguiente sucesión de consecuencias

`d`, por la regla R5 (que no tiene condiciones)
`e`. por la regla R6
`b`. por la regla R2 (porque sus condiciones `d` y `e` se verifican)
`c`. por la regla R4
`a`. por la regla R1

1. Definir el predicado `consecuencias(L)` de forma que se verifique si `L` es el conjunto de consecuencias del programa objeto obtenidas por encadenamiento hacia adelante. Por ejemplo, si escribimos la base de conocimiento anterior en el fichero `ejemplo_1.pl` y el predicado `consecuencias` en el fichero `progresivo.pl` se obtendría la siguiente sesión Prolog:

```
?- [progresivo, ejemplo_1].  =>  Yes
?- consecuencias(I).         =>  I = [a, c, b, e, d]
```

2. Explicar las modificaciones necesarias en la definición del predicado `consecuencias(L)` para que pueda aplicarse además a bases de conocimiento de primer orden. Por ejemplo, si escribimos en el fichero `ejemplo_2.pl` la base de conocimiento de primer orden

```
arco(a,b)   <- [].
arco(b,c)   <- [].
arco(c,a)   <- [].
nodo(X)     <- [arco(X,Y)].
nodo(Y)     <- [arco(X,Y)].
camino(X,X) <- [nodo(X)].
camino(X,Y) <- [camino(X,Z), arco(Z,Y)].
```

Apellidos: .....
Nombre: .....

y el predicado `consecuencias` en el fichero `progresivo.pl` se obtendría la siguiente sesión Prolog:

```
?- [progresivo, ejemplo_2].
Yes
?- consecuencias(L).
L = [camino(a,c),camino(a,b),camino(b,a),camino(b,c),
     camino(c,b),camino(c,a),camino(c,c),camino(b,b),
     camino(a,a),nodo(a),nodo(b),nodo(c),arco(c,a),
     arco(b,c),arco(a,b)]
Yes
```

3. Definir el predicado `prueba(A)` que sea un meta–intéprete con razonamiento Prolog (con encadenamiento hacia atrás) para las bases de conocimientos anteriores. Por ejemplo,

```
?- [ejemplo_1].       =>  Yes
?- prueba(a).         =>  Yes
?- prueba(g).         =>  No
?- [ejemplo_2].       =>  Yes
?- prueba(nodo(a)).   =>  Yes
?- prueba(nodo(p)).   =>  No
```

4. Definir el predicado `deducibles(L)` que se verifique si `L` es la lista de los átomos deducibles a partir del programa mediante el procedimiento `prueba`. Por ejemplo,

```
?- [ejemplo_1].     =>  Yes
?- deducibles(L).   =>  L = [a, b, c, d, e]
```

5. ¿Es cierto que para cualquier base de conocimiento todas sus consecuencias son deducibles; es decir, que si `A` verifica la relación
   `consecuencias(L1), member(A,L1)`
   entonces tiene que verificar la relación
   `deducibles(L2), member(A,L2)`?. Justificar la respuesta.

[**Nota:** *Se valorá la simplicidad de las definiciones de los predicados*]

Ejercicio 2 [3 puntos]
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 3 [3 puntos]
Consideremos la siguiente GCD:

```
oración           -->  sintagma_nominal,
                       sintagma_verbal.
sintagma_nominal  -->  nombre.
sintagma_nominal  -->  artículo,
                       nombre.
sintagma_verbal   -->  verbo,
                       sintagma_nominal.
artículo          -->  [el].
```

```
nombre          -->  [gato].
nombre          -->  [perro].
nombre          -->  [pescado].
nombre          -->  [carne].
verbo           -->  [come].
```

Ejemplos de frases admitidas por esta gramática son: `el perro come carne` y `el gato come pescado`.

El objetivo de este ejercicio es el de implementar en CLIPS el proceso de análisis sintáctico asociado a esta gramática. Para ello utilizaremos dos plantillas:

- `regla`: que servirá para almacenar las reglas de la gramática. Tendrá dos campos, uno para almacenar el símbolo no terminal del lado izquierdo (`izquierda`) y otro para almacenar los símbolos del lado derecho (`derecha`).
- `analisis`: que servirá para almacenar información relativa al proceso de análisis. Tendrá dos campos, uno para almacenar la frase que se quiere analizar (`frase`) y otro (`simbolos`) para almacenar el o los símbolos (terminales o no) a partir de los que se tiene que deducir la frase.

Se pide:

1. Construir las plantillas `regla` y `analisis` tal y como se han descrito.
2. Construir un conjunto de hechos para almacenar las reglas de la gramática, utilizando para ello la plantilla `regla`.
3. Construir reglas CLIPS (se puede hacer con tres reglas) que implemente el proceso de análisis sintáctico, de manera que a partir de las reglas de la gramática y un hecho que almacene un frase válida, por ejemplo:

   ```
   (analisis (simbolos F) (frase el gato come pescado))
   ```

   imprima en pantalla un mensaje diciendo que la frase es admitida.

[**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`*) ni la definición de nuevas funciones (*`deffunction`*)*]
