Introducción

Considere el siguiente ejemplo de un pequeño Sistema Experto que clasifica animales:

Características del sistema: Generar un procedimiento, que busque y verifique una hipótesis generando las preguntas necesarias y me informe “el animal buscado es...”.

Como hipótesis se trabajarán los siguientes animales: chita, tigre, jirafa, cebra, avestruz, pingüino, albatros (se puede agregar una hipótesis para la especie desconocida por el sistema)

Se utilizarán las siguientes reglas de clasificación (o las que quieran agregar, mejorar, etc):

- Chita es un mamífero, carnívoro que tiene color tostado y tiene manchas oscuras
- Tigre es un mamífero, carnívoro que tiene color tostado y tiene rayas negras.
- Jirafa es un ungulado que tiene cuello largo y piernas largas.
- Cebra es un ungulado que tiene rayas negras.
- Avestruz es un pájaro que no vuela y tiene cuello largo.
- Pingüino es un pájaro que no vuela, nada y es blanco y negro.
- Albatros es un pájaro que vuela bien y aparece en las historias de marineros antiguos.

Para las distintas categorías:

- Mamíferos: tienen pelos o se amamantan.
- Pájaros: tienen alas o vuelan y ponen huevos.
- Carnívoro: comen carne o, tienen dientes afilados y garras
- Ungulado mamífero que tiene pesuñas o caparazón.

Guía para el desarrollo:

Para implementar un sistema experto en Prolog que pregunte por las propiedades de un animal y deduzca de qué animal se trata podemos codificar reglas de este tipo:

```
animal(X) :- satisface(está_vivo,X), satisface(puede_sentir,X), satisface(puede_moverse,X).
mamífero(X) :- animal(X), satisface(da_leche,X), satisface(tiene_pelo,X).
tigre(X) :- mamífero(X), satisface(come_carne,X).
```

...

donde satisface es un predicado que se usa para que el sistema escriba una pregunta («¿Tiene el animal este atributo?») y que se puede programar así:

```
:- dynamic si/1,no/1.
satisface(Atributo,_) :- (si(Atributo) -> true ; ( no(Atributo) -> fail ; pregunta(Atributo))).
```

En la primera línea se declara que «si» y «no» son predicados dinámicos de grado 1, es decir, que se pueden alterar con assert y retract durante la ejecución.

El operador «->» permite construir una especie de «if-then-else» que implementa forward chain. La cláusula que define satisface se puede leer así: «se satisface el atributo si ya está en si (porque ya sea había preguntado antes); si no lo está entonces si ya está en no, no se satisface; y si tampoco está en no entonces pregunta».

La pregunta puede ser codificada así:

```
pregunta(A) :- write('¿Tiene el animal este atributo?: '), write(A), write(' '), read(Resp), nl,
((Resp == s ; Resp == si ; Resp == sí) -> assert(si(A)); assert(no(A) ), fail.
```

Escriba y compruebe el sistema experto que ante la consulta « animal.» va preguntando propiedades hasta deducir el animal (o «desconocido»). Para ello, puede utilizar esta regla:

```
animal :- adivina(Animal,_), write('El animal es: '), write(Animal), nl, borraResp.
```

donde

```
adivina(tigre,X) :- tigre(X).
...
adivina(_,desconocido).
```

borraResp es un predicado que se satisface siempre, pero antes borra (con retract) todas las cláusulas si y no que se hayan añadido (para permitir otra ejecución nueva).

Luego, escriba las reglas que definen el mismo conocimiento incluido en la red semántica (clasificación de los animales) dada previamente.

Trabajo Práctico: Sistema Experto en Prolog

1- Elija un dominio adecuado: problema de clasificación donde se proceda en etapas (categorías que luego se especialicen). Por ej. Clasificación de animales entre chita, tigre, etc. Utilizando la categoría de mamíferos, pájaros, etc.

2- Conceptualizar el dominio: que objetos (con sus atributos) y predicados se necesitan representar, y cuáles son las reglas que en base a las características de los objetos permiten clasificar. Por ej.; tienen pelos, color, amamantan, etc. Luego las reglas que permiten clasificar como las expuestas para los animales.

3- Escriba un programa en Prolog que implemente el sistema experto elegido en un dominio, tomando como ejemplo el que se ha presentado que averigüe de que animal se trata (pequeño sistema experto) preguntando los rasgos del animal que se usarán para la clasificación.

Solución: Identificación de animales

```
/* animal.pro
  animal identification game.

    start with ?- go.     */

go :- hypothesize(Animal),
      write('I guess that the animal is: '),
      write(Animal),
      nl,
      undo.

/* hypotheses to be tested */
hypothesize(cheetah)   :- cheetah, !.
hypothesize(tiger)     :- tiger, !.
hypothesize(giraffe)   :- giraffe, !.
hypothesize(zebra)     :- zebra, !.
hypothesize(ostrich)   :- ostrich, !.
hypothesize(penguin)   :- penguin, !.
hypothesize(albatross) :- albatross, !.
hypothesize(unknown).             /* no diagnosis */

/* animal identification rules */
cheetah :- mammal,
           carnivore,
           verify(has_tawny_color),
           verify(has_dark_spots).
tiger :- mammal,
         carnivore,
         verify(has_tawny_color),
         verify(has_black_stripes).
giraffe :- ungulate,
           verify(has_long_neck),
           verify(has_long_legs).
zebra :- ungulate,
         verify(has_black_stripes).

ostrich :- bird,
           verify(does_not_fly),
           verify(has_long_neck).
penguin :- bird,
           verify(does_not_fly),
           verify(swims),
           verify(is_black_and_white).
albatross :- bird,
             verify(appears_in_story_Ancient_Mariner),
             verify(flys_well).

/* classification rules */
mammal    :- verify(has_hair), !.
mammal    :- verify(gives_milk).
bird      :- verify(has_feathers), !.
bird      :- verify(flys),
             verify(lays_eggs).
carnivore :- verify(eats_meat), !.
```

```
carnivore :- verify(has_pointed_teeth),
             verify(has_claws),
             verify(has_forward_eyes).
ungulate :- mammal,
            verify(has_hooves), !.
ungulate :- mammal,
            verify(chews_cud).

/* how to ask questions */
ask(Question) :-
    write('Does the animal have the following attribute: '),
    write(Question),
    write('? '),
    read(Response),
    nl,
    ( (Response == yes ; Response == y)
      ->
       assert(yes(Question)) ;
       assert(no(Question)), fail).

:- dynamic yes/1,no/1.

/* How to verify something */
verify(S) :-
   (yes(S)
    ->
    true ;
    (no(S)
     ->
     fail ;
     ask(S))).

/* undo all yes/no assertions */
undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail.
undo.
```

El programa es principalmente interesante en cuanto a cómo trata de verificar ciertas propiedades que utiliza para sacar conclusiones, y cómo hace preguntas y registra las respuestas para referencia adicional. Si se hace una pregunta q y la respuesta es "sí", entonces esa respuesta se registra afirmando 'sí (q)', de lo contrario la respuesta se registra afirmando 'no(q)' y fallando.
Incluso las respuestas "sí" deben ser registradas ya que una respuesta posterior de "no" a una pregunta diferente mientras se intenta verificar la misma hipótesis puede hacer que toda la hipótesis fracase, pero esa misma respuesta "sí" podría conducir a una verificación exitosa de una hipótesis diferente más adelante. DE esta forma se evita que el programa haga la misma pregunta dos veces.
El método general de verificación de una condición q es entonces comprobar si 'sí (q)' se ha almacenado en la memoria, y tiene éxito, o 'no (q)' ha sido almacenado, y falla.