IA1 Proyecto1

Primer proyecto del curso de inteligencia artificial 1

Problema 1

Hechos

pareja(persona1, persona2): Indica que la persona 1 es pareja de la persona 2.

padre(persona1, persona2): Indica que la persona 1 es padre de la persona 2.

madre(persona1, persona2): Indica que la persona 1 es madre de la persona 2.

hijo(persona1, persona2): Indica que la persona 1 es hijo de la persona 2.

hermano(persona1, persona2): Indica que la persona 1 es hermano de la persona 2.

hermana(persona1, persona2): Indica que la persona 1 es hermana de la persona 2.

Reglas

es_tio(persona1, persona2): Indica que la persona 1 es tio/tia de la persona 2. Se busca que la persona 1 y una persona 3 sean hermanos/hermanas,y que la persona 3 sea el padre de la persona 2.

```
es_tio(A, C):- (hermano(A, B);hermana(A, B)), hijo(C, B).
```

es_primo(persona1, persona2): Indica que la persona 1 es primo/prima de la persona 2. Se busca que las persona 3 y persona 4 sean hermanos/hermanas, que persona 1 sea hijo/hija de la persona 3, y la persona 2 sea hijo/hija de la persona 4 para que la persona 1 y persona 2 sean primos.

```
es_primo(A, B):- (hermano(C, D); hermano(D, C); hermana(C, D); hermana(D,
C)), hijo(A, C), hijo(B, D).
```

es_sobrino(persona1, persona2): Indica que la persona 1 es sobrino/sobrina de la persona 2. La persona 1 es sobrino/sobrina de la persona 2 si la persona 2 es tio/tia de la persona 1.

```
es_sobrino(A, B):- es_tio(B, A); es_tia(B, A).
```

es_culpable(persona): Indica el culpable del asesinato de Marta.

```
es_culpable(A):- hermana(C, A), pareja(marta, bruce), es_primo(C, clark),
es_tio(barry, C).
```

imprimir_arbol(persona): Devuelve el arbol apartir de la persona seleccionada. Se busca si se tiene pareja la persona, y si tiene hijos, los hijos se recorren y se realiza lo mismo. Las parejas se representan por: `*--*`, y la lista de hijos por `->`.

```
imprimir_hijos([]):- nl.
imprimir_hijos([A|B]):- write(' -> '), write(A), imprimir_hijos(B).

imprimir_descendientes([]):- nl.
imprimir_descendientes([A|B]):- imprimir_arbol(A),
imprimir_descendientes(B).

imprimir_arbol(X):-
  (
    pareja(X, Y) ->
    write(X), write(' *--* '), write(Y), nl
    ;
    write(X)
  ),
  (
    hijo([HijoActual|SiguientesHijos], X) ->
    write(X), write(' -> '), write(HijoActual),
    imprimir_hijos(SiguientesHijos),
    nl,
    nl,
    imprimir_arbol(HijoActual),
    imprimir_descendientes(SiguientesHijos),
    nl
    ;
    nl
  ).
imprimir_arbol(_):- imprimir_arbol(bruce).
```

Modo de uso de las reglas del problema 1

Importar archivo del problema 1 en prolog.

```
consult('problema1.pl').
```

imprimir_arbol

Para imprimir todo el arbol de la familia se puede utilizar la variable especial `_`.

```
imprimir_arbol(_).
```

Salida:

La salida que genera `imprimir_arbol(_)` es el siguiente:

```
marta *--* bruce // pareja
marta -> barry -> rachel -> diana -> may // madre/padre -> lista de hijos


barry *--* pepper // pareja
barry -> lara -> tony // madre/padre -> lista de hijos


lara // persona, sin pareja e hijos
tony // persona, sin pareja e hijos


rachel *--* enrique // pareja
rachel -> clark -> lois // madre/padre -> lista de hijos


clark // persona, sin pareja e hijos
lois  // persona, sin pareja e hijos


diana *--* peter // pareja
diana -> mary -> harry // madre/padre -> lista de hijos


mary  // persona, sin pareja e hijos
harry // persona, sin pareja e hijos


may *--* ben // pareja
may -> ezio -> lorenzo -> sergio // madre/padre -> lista de hijos


ezio    // persona, sin pareja e hijos
lorenzo // persona, sin pareja e hijos
sergio  // persona, sin pareja e hijos
```

es_culpable

Se ingresa una variable, para conocer el culpable del asesinato de Marta, en este caso muestra a Harry.

```
?. es_culpable(X).

X = harry.
```

Problema 2

```

```

Problema 3

Reglas para realizar operaciones con listas.

Reglas

reverso(X, Y): Invierte una lista. Donde **X** es la lista invertir, **Y** es la lista resultado. Esta regla obtiene una lista de entrada, utiliza una lista acumuladora para invertir la lista, una lista para guardar el resultado.

```
reverso(Lista, Resultado) :-
  reverso(Lista, [], Resultado).

reverso([], ListaInvertida, ListaInvertida).

reverso([Cabeza|Cola], RestoCola, ListaInvertida) :-
  reverso(Cola, [Cabeza|RestoCola], ListaInvertida).
```

es_palindroma(X): Verifica que una lista es palindroma. Utiliza la regla reverso, en la cual verifica si en inverso y la lista son iguales para confirmar que la lista es palindroma.

```
es_palindroma(Lista) :-
  reverso(Lista, Lista).
```

duplicar(X, Y): Duplicar la lista (ejemplo: [a,b,c] ->[aa,bb,cc]). En esta regla se obtiene lista para duplicar, y otra donde se guardara el resultado. Esta regla va obteniendo descomponiendo la lista

actual obteniendo la cabeza en cada iteración, y la combina con la regla nativa `atom_concat` y la inserta en la lista resultado.

```
duplicar([],[]).
duplicar([Cabeza|Cola],[Concat|ColaResultado]) :-
  atom_concat(Cabeza, Cabeza, Concat),
  duplicar(Cola,ColaResultado).
```

dividir(X, Y, Z): Divide la lista en dos y retornar las dos listas resultantes (Si la lista es par retorna 2 listas de igual tamaño, si no 2 de diferente tamaño). Esta regla se le envia la lista, y guarda cada uno de las mitades en las otras dos listas representadas por **Y**, y **Z**.

```
dividir([], [], []).
dividir([X], [X], []).
dividir([X, Y|RestoLista], [X|ListaPrimeraMitad], [Y|ListaSegundaMitad]) :-
  dividir(RestoLista, ListaPrimeraMitad, ListaSegundaMitad).
```

insertar(X, Y, Z): Insertar un elemento en la ultima posición de dicha lista. **X** representa la lista, **Y** el elemento a insertar, y **Z** la lista resultado.

```
insertar([], ElementoNuevo, [ElementoNuevo]).
insertar([Cabeza|RestoLista], ElementoNuevo, [ElementoNuevo,
Cabeza|RestoLista]) :-
  ElementoNuevo @< Cabeza, !.
insertar([Cabeza|RestoLista1], ElementoNuevo, [Cabeza|RestoLista2]) :-
  insertar(RestoLista1, ElementoNuevo, RestoLista2).
```

Modo de uso de las reglas del problema 3

```
?- consult('problema3.pl').
true.

?- reverso([1, 2, 3, 4], X).
X = [4, 3, 2, 1].

?- es_palindroma([1, 2, 2, 1]).
true.

?- duplicar([1, 2, 3, 4], X).
X = ['11', '22', '33', '44'].
```

```
?- dividir([1, 2, 3, 4], X, Y).
X = [1, 3],
Y = [2, 4].

?- insertar([1, 2, 3], 4, Z).
Z = [1, 2, 3, 4].
```

Problema 4

Hechos

numero(X): Hecho la cual indica que X es un numero entre 1 al 4.

```
% hechos
numero(1).
numero(2).
numero(3).
numero(4).
```

Reglas

sudoku: solicita al usuario el sudoku de la siguiente forma `[[numero, _, _, numero], [_, _, numero, numero], [_, _, _, numero], [numero, _, numero, _]]` en donde `numero` indica los numeros a colocar, y `_` indica los espacios en blanco del sudoku.

resolver_sudoku(X): regla para solucionar el sudoku donde `X` es la lista ingresada.

```
% Column - C, Row - R
resolver_sudoku([
  [R1C1, R1C2, R1C3, R1C4],
  [R2C1, R2C2, R2C3, R2C4],
  [R3C1, R3C2, R3C3, R3C4],
  [R4C1, R4C2, R4C3, R4C4]
]) :-
  resolver(
    R1C1, R1C2, R1C3, R1C4,
    R2C1, R2C2, R2C3, R2C4,
    R3C1, R3C2, R3C3, R3C4,
    R4C1, R4C2, R4C3, R4C4
  ),
  write('--------------'), nl,
  imprimir_fila(R1C1, R1C2, R1C3, R1C4),
  imprimir_fila(R2C1, R2C2, R2C3, R2C4),
  write('--------------'), nl,
```

```
  imprimir_fila(R3C1, R3C2, R3C3, R3C4),
  imprimir_fila(R4C1, R4C2, R4C3, R4C4),
  write('--------------')
  .
```

imprimir_fila(X, W, Z, Y): regla para imprimir una fila del sudoku.

```
imprimir_fila(X, W, Z, Y)
  :- write('|'), write(X),
  write('|'), write(W),
  write('||'), write(Z),
  write('|'), write(Y), write('|'),nl
```

resolver(R1C1, R1C2, R1C3, R1C4, R2C1, R2C2, R2C3, R2C4, R3C1, R3C2, R3C3, R3C4, R4C1, R4C2, R4C3, R4C4): regla para resolver el sudoku.

```
resolver(
  R1C1, R1C2, R1C3, R1C4,
  R2C1, R2C2, R2C3, R2C4,
  R3C1, R3C2, R3C3, R3C4,
  R4C1, R4C2, R4C3, R4C4
) :-
  diferente(R1C1, R1C2, R1C3, R1C4), % primera fila
  diferente(R2C1, R2C2, R2C3, R2C4), % segunda fila
  diferente(R3C1, R3C2, R3C3, R3C4), % tercera fila
  diferente(R4C1, R4C2, R4C3, R4C4), % cuarta fila
  diferente(R1C1, R2C1, R3C1, R4C1), % primera columna
  diferente(R1C2, R2C2, R3C2, R4C2), % segunda columna
  diferente(R1C3, R2C3, R3C3, R4C3), % tercera columna
  diferente(R1C4, R2C4, R3C4, R4C4), % cuarta columna
  diferente(R1C1, R1C2, R2C1, R2C2), % primer cuadrante
  diferente(R1C3, R1C4, R2C3, R2C4), % segundo cuadrante
  diferente(R3C1, R3C2, R4C1, R4C2), % tercer cuadrante
  diferente(R3C3, R3C4, R4C3, R4C4). % cuarto cuadrante
```

diferente(X, W, Z, Y): verifica que todos los numeros sean diferentes o iguales.

```
diferente(X, W, Z, Y)
  :-
    numero(X), numero(W), numero(Z), numero(Y),
    X\=W, X\=Z, X\=Y, W\=Z, W\=Y, Z\=Y.
```

Modo de uso de las reglas del problema 4

```
?- consult('problema4.pl').
true.

?- resolver_sudoku([[4, _, _, _],[_, 3, _, _],[_, _, 1, _],[_, 1, _, 2]]).
----------
|4|2||3|1|
|1|3||2|4|
----------
|2|4||1|3|
|3|1||4|2|
----------
true .

?- resolver_sudoku([[3, _, 4, _],[_, 1, _, 2],[_, 4, _, 3],[2, _, 1, _]]).
----------
|3|2||4|1|
|4|1||3|2|
----------
|1|4||2|3|
|2|3||1|4|
----------
true .
```