Inteligencia Artificial.

Práctica 2.

Juan Siverio Rojas.

Alu0100585704@ull.edu.es

Problema acertijo del pastor, lobo, oveja y cabra.

"Un pastor debe transportar un lobo, una oveja y una col a través de un río, usando una barca en la que sólo cabe él junto a uno de ellos. El lobo y la oveja no pueden quedarse solos en la misma orilla sin el pastor, ni la oveja puede quedarse sola con la col. El pastor puede hacer tantos viajes a través del río como sea necesario."

Requisitos.

- Debe partir de una etapa inicial en la que todos los elementos del problema se encuentren en una orilla y terminar cuando se hayan trasladado todos a la otra.
- Se debe utilizar acumuladores para generar la secuencia de etapas paso a paso.
- Debe indicarse como hechos las posibles combinaciones de elementos que pueden darse en una orilla, de modo que el sistema no acepte soluciones que incluyan etapas incompatibles con el enunciado (como por ejemplo [[pastor, col], [oveja, lobo]]).
- Para evitar obtener soluciones superfluas y bucles infinitos, el sistema no debe repetir etapas previas.
- Las permutaciones de los contenidos de cada orilla no deben generar etapas diferentes: por ejemplo, [pastor, col, oveja] debe equivaler a [col, oveja, pastor] a la hora de realizar las comprobaciones.

Este programa utiliza la siguiente sintaxis:

Sintaxis:

:-solucion(A,B,X). Donde A=lista orilla 1, B=lista orilla 2, X = contendrá resultado de la traza y solución

Ejemplo:

:-solucion([pastor,lobo,oveja,col],[],X). Devolverá la traza desde la posición todos en la orilla 1.

:-solucion([lobo,col],[pastor,oveja],X). Devolverá la traza desde la posición en la que lobo y col están en la orilla 1 y pastor y oveja están en la orilla 2.

:-solucion([lobo,oveja,col],[pastor],X). Devolverá falso, puesto que esta posición inicial no se puede dar, ya que el lobo se comería a la oveja o la oveja a la col.

Este programa devuleve la traza seguida para el acertijo del pastor la oveja, lobo y col con el siguiente formato:

X = [[[pastor,lobo,oveja,col], []], [[lobo, col], [pastor, oveja]], [[pastor, lobo, col], [oveja]], [[col], [pastor, lobo, oveja] [[pastor, oveja, col], [lobo]], [[oveja], [pastor, col, lobo]], [[pastor, oveja], [col, lobo]], [[], [pastor,oveja,col,lobo]]]

Este programa permite la permutación de las listas y además he incorporado un parámetro para resolver incluso sin tener porque partir de la posición inicial.

- Se agrega un wraper para no tener que especificar todos los parámetros.
- Se definen todos los hechos de la siguiente manera para aceptar posibles variaciones en las listas, permutaciones:

orillas(X,Y):-permutacion(X,[lobo]) , permutacion(Y,[pastor,oveja,col]).

- Creo mi propio buscar porque el predicado "member" no me funciona con el SLDDRAW 3.4, programa que he usado para realizar trazas del programa a través de árboles.

buscar(Valor,[Valor | _]).

buscar(Valor,[_|Cola]):- buscar(Valor,Cola).

- Creo mi propio agregar porque el predicad "append" no me funciona con el SLDDRAW 3.4, programa que he usado para realizar trazas del programa a través de árboles.

agregar([], L, L).

agregar([H|L1], L2, [H|L3]):- agregar(L1, L2, L3).

- Los predicados extraer y permutacion han sido dados por el profesor.

- El predicado ***"mover"*** con la siguiente sintaxis:

  mover(pastor,Origen,Destino,NuevoOrigen,NuevoDestino).

  Quita de la lista Origen el objeto (pastor, oveja, lobo o col) especificado y lo inserta en la lista destino. Las nuevas listas se devuelven en NuevoOrigen y NuevoDestino.

  Mover el pastor es el único que no comprueba si está en la lista antes de moverlo, puesto que se llega habiendo hecha ya esa comprobación en el predicado ***solucionR***.

  Además, mover el pastor es el único movimiento que se podría realizar solo, los otros tres, siempre mueven primero al pastor y después el objeto seleccionado por backtraking.

- El predicado ***"ultimoMovimiento"*** con la siguiente sintaxis:
  ultimoMovimiento(UltimoOrigen,UltimoDestino,OrigenActual,DestinoActual).

Unifica si el último movimiento realizado es igual al actual, o sea, al que acabo de realizar. De esta forma, como lo niego en la regla, solo acepta el movimiento si realmente no unificó, o sea, no fue igual al anterior, así evitó bucles.

- Wraper:
  solucion(ListaInicial,DestinoInicial,Camino):-
  solucionR(ListaInicial,DestinoInicial,[],Camino,[],[]).

- Predicado solucionR con la siguiente sintaxis:
  solucionR(Origen,Destino,PasoAPaso,Camino,UltimoOrigen,UltimoDestino).

  De este predicado hay dos, básicamente lo que diferencia cada predicado solucionR, es si el pastor está en la orilla1 o en la orilla 2, comprobación que hago al principio de la regla.

  En base a eso, el algoritmo es el siguiente:

  1.- El pastor siempre lo muevo.

  2.- Compruebo que ese movimiento no me llevó a una situación idéntica a la del paso anterior.

  3.- Si el movimiento ya se había realizado en el paso anterior, se busca por backtraking otro movimiento diferente.

  4.- Si el movimiento no se había realizado, comprueba que se haya llegado a una posición legal, comprobando con los hechos.

  5.- Si es posición legal, agrego a la traza el nuevo estado y vuelvo a seguir buscando soluciones o comprobar si ya es la solución final.

  6.- En caso que la posición no sea legal, por backtraking, hago otro movimiento y continuo.