Prolog 2

1.- T1. Implementación de un autómata no determinista

Dado el autómata finito no determinista definido en el boletín de prácticas, donde cada posible transición viene etiquetada por la letra que acepta y la transición lambda (vacía) está sin etiquetar. El estado final es el estado e3.

a) Represente el autómata por los siguientes hechos (Esto sería la base de hechos inicial o conjunto

de predicados iniciales):

- e1 es el estado inicial.
- e3 es un estado final.
- hay 7 transiciones en el autómata, cinco que están etiquetadas y dos que no (transiciones lambda). Representa cada transición, mediante un predicado con tres argumentos: estado de partida, etiqueta de la transición y estado resultante.

La base de hechos sería la siguiente:

```
inicio(e1).
fin(e3).
transicion(e1,a,e1).
transicion(e1,a,e2).
transicion(e1,b,e1).
transicion(e2,b,e3).
transicion(e2,_,e4).
transicion(e3,_,e1).
transicion(e3,b,e4).
```

b) Implemente las reglas necesarias que verifiquen si el autómata acepta o no una determinada lista de caracteres de entrada.

Por ejemplo:

- ? - aceptacadena([b,a,b,a,b]). true .
- ? - aceptacadena([a,a,a]). False .

La implementación de estas reglas podría ser la siguiente:

```
aceptacadena([X|R]) :- inicio(I), transicion(I, X, Y), aceptacadenaactual(R, Y).
aceptacadenaactual([X|[]], E) :- fin(F), transicion(E,X,F).
aceptacadenaactual([X|R], E) :- transicion(E, X, Y), aceptacadenaactual(R, Y).
```

c) ¿Cómo realizaría la consulta para saber qué cadena de longitud 3 acepta el autómata?

Una consulta como la siguiente nos diría las combinaciones de cadenas de longitud 3 que se aceptarían por el automata:

```
?- aceptacadena([X,Y,Z]).
X = Y, Y = a,
Z = b ;
X = Z, Z = b,
Y = a ;
false.
```

d) Implemente una regla que permita consultar si el autómata acepta una cadena de longitud determinada.

Una posible implementación de dicha regla podría ser:

```
aceptacadenalg(Arr,N):- length(Arr,N), aceptacadena(Arr).
```

La cual podremos llamar de la siguiente forma para obtener:

```
?- aceptacadenalg(C,4).
C = [a, a, a, b] ;
C = [a, b, a, b] ;
C = [b, a, a, b] ;
C = [b, b, a, b] ;
false.
```

e) Implemente una regla que permita consultar las cadenas de longitud menor o igual que una dada que acepta el autómata. Nota: chequee el predicado predefinido between.

Para obtener dicha regla podríamos añadir a nuestro archivo fuente lo siguiente:

```
aceptacadenabet(Arr,N):- between(0,N,L), aceptacadenalg(Arr,L).
```

Obteniendo la siguiente salida para el comando mostrado:

```
?- aceptacadenabet(C,4).
C = [a, b] ;
C = [a, a, b] ;
C = [b, a, b] ;
C = [a, a, a, b] ;
C = [a, b, a, b] ;
C = [b, a, a, b] ;
C = [b, b, a, b] ;
false.
```

2. Implementación de un problema de búsqueda en un espacio de estados

Finalmente, en este apartado realizaremos una búsqueda en un espacio de estados. La representación definirá una sala con un mono que quiere obtener unos plátanos coldados del techo para lo cual tendrá que coger una silla, moverla hacia el centro de la sala, subirse a ella y coger los plátanos. En los siguientes apartados definiremos los estados y operadores entre ellos para solucionar el problema.

1.- ¿Qué situaciones (o estados) podríamos identificar?

Para representar los estados introduciremos unas reglas con la siguiente forma, siendo el estado inicial el

mostrado al final de la siguiente porción de código:

```
% Esta será la representación de los estados y sus posibles valores:
% estado(LOC_MONO, ALT_MONO, LOC_SILLA, LOC_PLANATOS)
%         puerta    suelo     puerta      abajo
%         ventana   silla     ventana     arriba
%         centro              centro

% Este será el estado inicial:
% estado(puerta, suelo, ventana, arriba).

% Y este el estado final que queremos alcanzar:
% estado(centro,silla,centro,abajo).
```

2&3.- Implementación del programa

La implementación final que hemos realizado tiene la siguiente forma, con una regla recursiva que va comprobando las transiciones posibles dado el estado en el que nos encontramos.

```
estado_final(estado(centro, silla, centro, abajo)).

% ---------TRANSICIONES---------
transicion(bajar_platano,estado(centro,silla,centro,arriba),estado(centro,silla,ce
ntro,abajo)).
transicion(subir_silla(X),estado(X,suelo,X,arriba),estado(X,silla,X,arriba)).
transicion(mover_silla(X,Z),estado(X,suelo,X,arriba),estado(Z,suelo,Z,arriba)).
transicion(mover_mono(X,Z),estado(X,suelo,Y,arriba),estado(Z,suelo,Y,arriba)).


% ----------OPERADORES----------
soluciona(EstadoIn,[]) :- estado_final(EstadoIn).
soluciona(EstadoIN,[O|Os]) :- transicion(O,EstadoIN, EstadoOUT), soluciona(EstadoO
UT,Os),!.
```

4.- Cambie el orden de los operadores y ejecute de nuevo el programa. Compruebe que sigue funcionando bien.

Hemos realizado varias pruebas cambiando el orden de las reglas siendo un ejemplo el siguiente código:

```
estado_final(estado(centro, silla, centro, abajo)).

% ---------TRANSICIONES---------
transicion(mover_silla(X,Z),estado(X,suelo,X,arriba),estado(Z,suelo,Z,arriba)).
transicion(subir_silla(X),estado(X,suelo,X,arriba),estado(X,silla,X,arriba)).
transicion(mover_mono(X,Z),estado(X,suelo,Y,arriba),estado(Z,suelo,Y,arriba)).
transicion(bajar_platano,estado(centro,silla,centro,arriba),estado(centro,silla,ce
ntro,abajo)).


% ----------OPERADORES----------
soluciona(EstadoIn,[]) :- estado_final(EstadoIn).
soluciona(EstadoIN,[O|Os]) :- transicion(O,EstadoIN, EstadoOUT), soluciona(EstadoO
UT,Os),!.
```

Al tener las reglas de forma general lo que ocurre es que entra en bucles infinitos con las reglas de mover el mono o mover la silla. Esto se debe a que los estados de las transiciones para subirse a la silla y para bajar los plátanos también son válidos para las transiciones de movimiento. Esto causa que si alguna regla de movimiento esté por encima de las reglas de subirse a la silla o coger plátanos entremos en un bucle constante de mover o el mono o la silla.