ejercicio 2

1. Implementación de un autómata no determinista

Sea el siguiente autómata finito no determinista, 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

```
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

```
aceptaCadena([X|R]) :- inicio(A), transicion(A, X, Y), aceptaCadena2(R, Y).
aceptaCadena2([X|[]], E) :- fin(F), transicion(E,X,F).
aceptaCadena2([X|R], E) :- transicion(E, X, Y), aceptaCadena2(R, Y).
```

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

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

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

```
cadenaLongitudExacta(C,L) :- length(C,L) ,aceptaCadena(C).
```

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

```
cadenaLongitudMenorExacta(C,L) :- between(0,L,X), length(C,X) ,aceptaCadena(C).
```

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

Un mono se encuentra en la puerta de una habitación. En el centro de la habitación hay unos plátanos colgados del techo. El mono está hambriento y desea coger el plátano, pero no lo alcanza desde el suelo.

En la ventana de la habitación hay una silla que el mono puede usar. El problema puede resolverse mediante una búsqueda en un espacio de estados.

1) ¿Qué situaciones (o estados) podríamos identificar? Una posible solución es primero considerar los tres elementos que definen la situación: el mono, la silla y el plátano. los dos primeros elementos se pueden encontrar en las tres posibles localizaciones de interés para el problema: en la puerta, en el centro de la habitación o al lado de la ventana. El plátano puede estar colgado del techo o no (en cuyo caso, lo tendrá el mono). Además, el mono puede encontrarse directamente sobre el suelo o encima de una silla. ¿Cómo representarías un estado o situación del problema? Define la base de hechos inicial con un hecho que defina el estado final.

estado:

- posicion mono
- altura mono
- posicion caja
- posiicion plátano

inicio(estado(puerta,abajo,ventana,arriba)). fin(estado(centro,arriba,centro,arriba)).

2) Implementar las reglas necesarias para las acciones posibles. Es decir, una regla por cada operador a aplicar.

```
transicion(subirSilla(X),estado(X,abajo,X,arriba),estado(X,arriba,X,arriba)).
transicion(moverSilla(Y,centro),estado(Y,abajo,Y,arriba),estado(centro,abajo,centro,arriba)).
transicion(moverMono(X,Y),estado(X,abajo,Z,arriba),estado(Y,abajo,Z,arriba)).
```

3) Implementar, mediante recursión, las reglas necesarias para obtener la solución aplicando los operadores definidos en el apartado anterior sobre un estado inicial.

```
resolver(STATE, [OP]) :-  fin(FINAL), transicion(OP, STATE, FINAL), !.
resolver(STATE, [OP|OP2]) :- transicion(OP, STATE, STATE2), resolver(STATE2, OP2), !.
```

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

Deja de funcionar porque entra en bucle infinitos