```
(*Practica 2: Andreu Mut Portes. Ejercicios*)

(*Ejercicio 1: Per ser determinista des de un estat i un simbol nomes pot anar
  a un altre estat. Si des d'un estat amb un simbol pots anar a dos estats,
deixa de ser AFD. Si per exemple des del estat 1 puc
 anar a 2 o 3 utilitzant una "a" com a simbol*)

Ex1[aut_] := Module[{est, alf, trans, ini, fin, i, j, aux},
             módulo
  (*Enmagatzemar la info del automat que li passem en diverses variables*)
  est = aut[[1]]; alf = aut[[2]];
  trans = aut[[3]]; ini = aut[[4]]; fin = aut[[5]];
  For[i = 1, i ≤ Length[est], i++, (*<- Recorrer tot els estats...*)
  para cada          longitud
   For[j = 1, j ≤ Length[alf], j++,
   para cada          longitud
     (*<- Recorrer tot els estats... per a cada simbol*)
     aux = Cases[trans, {est[[i]], alf[[j]], _}];
           casos
     (*Torna el numero de coincidencies de transicions*)
     If[Length[aux] > 1, Return[False]]; (*Si fora major que 1,
     si  longitud              retorna  falso
     ja no es AFD xk dun estat a un altre pots anar de mes d'una manera*)
    ];
  ];
  Return[True]; (*Si ha passat tots els bucles
  retorna  verdadero
   d'abans i no ha eixit pel False, es que es verdader*)
                              falso
 ]

automat1 = {{0, 1, 2}, {a, b, c}, {{0, a, 0}, {0, a, 1}, {0, a, 2}, {0, b, 1}, {0, b, 2},
    {0, c, 2}, {1, b, 1}, {1, b, 2}, {1, c, 2}, {2, c, 2}}, 0, {0, 1, 2}};

Ex1[automat1]

False

automat2 = {{0, 1, 2, 3}, {a, b},
   {{0, a, 2}, {0, b, 1}, {1, a, 3}, {2, a, 2}, {2, b, 3}, {3, b, 0}}, 0, {0, 3}};

Ex1[automat2]

True
```

```
(*Ejercicio 2:
 Bideterminisme:
 -Condicio 1:Mirar si el estat final es 1. Si te mes de 1, False.
                                                          falso
   -Condicio 2: Un estat X nomes pot rebre un unic simbol. Si el estat 2,
rep una "a" del estat 0 i una "a" buclejat en si mateix,
ja no seria bideterminista*)
```

```
Ex2[aut_] := Module[{est, alf, trans, ini, fin, i, j, aux},
             módulo
  est = aut[[1]]; alf = aut[[2]];
  trans = aut[[3]]; ini = aut[[4]]; fin = aut[[5]];
  (*Condicio 1:Mirar si el estat final es 1. Si te mes de 1, False.*)
                                                                  falso
  If[Length[fin] > 1, Return[False] ];
  si  longitud              retorna  falso
  (*Condicio 2:Un estat X nomes pot rebre un unic simbol.*)
  For[i = 1, i ≤ Length[est], i++, (*<- Recorrer tot els estats...*)
  para cada          longitud
   For[j = 1, j ≤ Length[alf], j++,
   para cada          longitud
     (*<- Recorrer tot els estats... per a cada simbol*)
     (*La linia següent diu: Al estat iessim i al estat
        jessim quantes transicions arriben???*)
     aux = Cases[trans, {_, alf[[j]], est[[i]]}];
           casos
     (*Si el aux es major que 1 es que
      arriba mes d'una transicio per al meteix simbol.*)
     If [Length[aux] > 1, Return[False]];
     si  longitud              retorna  falso
    ];
  ];
 ]
```

```
(*Ejercicio 3 Es complet quan en per a tot estat i tot simbol,
te definida una transicio. Si per a un estat/simbol no la troba,
que done False directament*)
          falso
```

```
Ex3[aut_] := Module[{est, alf, trans, ini, fin, i, j, aux},
             módulo
  (*Enmagatzemar la info del automat que li passem en diverses variables*)
  est = aut[[1]]; alf = aut[[2]];
  trans = aut[[3]]; ini = aut[[4]]; fin = aut[[5]];
  For[i = 1, i ≤ Length[est], i++, (*<- Recorrer tot els estats...*)
  para cada          longitud
   For[j = 1, j ≤ Length[alf], j++,
   para cada          longitud
     (*<- Recorrer tot els estats... per a cada simbol*)
     aux = Cases[trans, {est[[i]], alf[[j]], _}];
           casos
     (*Torna el numero de coincidencies de transicions*)
     If [Length[aux] == 0, Return[False]];
     si  longitud              retorna  falso
     (*Si es zero es que no esta definida la transicio.*)
    ];
  ];
  Return [True]; (*Si ha passat tots els bucles
  retorna   verdadero
   d'abans i no ha eixit pel False, es que es verdader*)
                                  falso
 ]
```

```
(*Ejercicio 4 *)

Ex4[automat_, cadena_] := Module[{trans, fin, estatActual, i, aux},
                          módulo
  trans = automat[[3]]; fin = automat[[5]];
   estatActual = automat[[4]];


  (*Bucle que recorre la cadena*)
  For [i = 1, i ≤ Length[cadena], i++,
  para cada          longitud
   aux = Cases[trans, {estatActual, cadena[[i]], _}];
         casos
   If[aux ≠ {}, estatActual = aux[[1, 3]], Return[False]];
   si                                      retorna  falso
  ];
  Return [MemberQ[fin, estatActual]];
  retorna   ¿contenido en?
 ]

automat1 = {{0, 1, 2, 3}, {a, b},
   {{0, a, 2}, {0, b, 1}, {1, a, 3}, {2, a, 2}, {2, b, 3}, {3, b, 0}}, 0, {0, 3}};

Ex4[automat1, {a, b, a}]
```

```
False
```

```
Ex4[automat1, {a, b}]
```

```
True
```

```
(*Ejercicio 5 *)

Ex5[aut1_, aut2_, cadena_] := Module[{val1, val2},
                                módulo
      val1 = Ex4[aut1, cadena];
      val2 = Ex4[aut2, cadena];
  Return[val1 || val2];
   retorna
 ]
```

```
(*Ejercicio 6*)

Ex6[aut1_, aut2_, cadena_] := Module[{val1, val2},
                                módulo
      val1 = Ex4[aut1, cadena];
      val2 = Ex4[aut2, cadena];
  Return[val1 && val2];
   retorna
 ]
```

```
(*Ejercicio 7 *)
```

```
Ex7[autom_, conj_, simb_] := Module[{transicions, i, aux, aux2},
                              módulo
  transicions = autom[[3]];
  aux = {};
  For[i = 1, i ≤ Length[conj], i++,
  para cada          longitud
      aux2 = Cases[transicions, {conj[[i]], simb, _}];
             casos
      aux = Join[aux, aux2];
            junta
  ];
  (*Print["Despres primer bucle", aux];*)
    escribe
  (*Aci en este punt aux te guardades les transicion que
   tenen el conjunt inicial CONJ amb el simbol donat SIMB.*)
  (*Ara cal recorrer aux i treure el tercer element de cada llista
   i crear una llista nova.*)
  aux2 = {};
  For[i = 1, i ≤ Length[aux], i++,
  para cada          longitud
   aux2 = Append[aux2, aux[[i]][[3]]];
          añade
  ];
  Return[Union[aux2]];
  ret···     unión
 ]

automat1 = {{1, 2, 3}, {a, b}, {{1, a, 1}, {1, a, 2}, {1, b, 2},
    {2, a, 3}, {2, a, 1}, {2, b, 3}, {3, a, 2}, {3, b, 3}}, 1, {1, 2}};

conj1 = {1, 3};
simb1 = a;

Ex7[automat1, conj1, simb1]
{1, 2}

conj2 = {2, 3};
simb2 = b;

Ex7[automat1, conj2, simb2]
{3}
```

```
(*Ejercicio 8 *)
```

```
Ex8[automat_, cadena_] := Module[{trans, fin, conjuntActual, i},
                          módulo
  trans = automat[[3]]; fin = automat[[5]];
  conjuntActual = {};
   conjuntActual = Append[conjuntActual, automat[[4]]];
                   añade
  (*Bucle que recorre la cadena*)
  For [i = 1, i ≤ Length[cadena], i++,
  para cada          longitud
    conjuntActual = Ex7[automat, conjuntActual, cadena[[i]]];

    If[conjuntActual == {}, Return[False]];
    si                      retorna  falso
  ];
  (*Opcio 1: Encarna. Es mes curta.*)
  Return[Intersection[fin, conjuntActual] ≠ {}];
         intersección

  (* Opcio 2: Andreu. Aquesta funciona.
     For[i=1, i≤Length[conjuntActual],i++,
                 longitud
      If[MemberQ[fin,conjuntActual[[i]]],Return[True]];
      si ¿contenido en?                  retorna verdadero
     ];
  Return[False]; *)
  falso
 ]

automat8 = {{1, 2, 3}, {a, b, c}, {{1, a, 1}, {1, a, 2}, {1, a, 3}, {1, b, 2}, {1, b, 3},
    {1, c, 3}, {2, b, 2}, {2, b, 3}, {2, c, 3}, {3, c, 3}}, 1, {1, 2, 3}};

cadSi = {a, b, b, c};
cadNo = {a, b, b, c, a, a, a};

Ex8[automat8, cadSi]
True

Ex8[automat8, cadNo]
False
```

```
(*Ejercicio 9: Recorrer els simbols amb un Cases que utilitze el simbols.
                                                  casos
   Comprovar de l'ultim components si el tercer component se manté.
       Fer union de la llista i si la longitud de la llista es 1
  significa que tenim un estat nomes i ho complix. Si es mes de 1,
es que hi han mes estats i no compleix la propietat.*)
```

```
Ex9[automat_] := Module[{trans, simbols, i, aux, aux2, j},
                 módulo
  trans = automat[[3]]; simbols = automat[[2]];
  aux = {};
  (*Este primer bucle treu totes les transicions per a cada simbol*)
  For[i = 1, i ≤ Length[simbols], i++,
  para cada          longitud
   aux2 = Cases[trans, {_, simbols[[i]], _}];
          casos
   aux = Join[aux, aux2];
         junta
  ];
  (*Print[aux];*)
    escribe
  (*El segon bucle extreu de cada tupla el tercer nombre*)
  aux2 = {};
  For[i = 1, i ≤ Length[aux], i++,
  para cada          longitud
   aux2 = Append[aux2, aux[[i]][[3]]];
          añade
  ];

  (*Ara que tenim en aux2 nomes els estats desti,
  se compara si el nombre de estats desti es igual  al nombre de
   simbols. Si es veritat, es que cada simbol tindra 1 estat desti
   UNIC. Si fora mes que el nombre de simbols, es que almenys un
   dels estats te com a desti mes de un i ja NO seria UNIC.*)
  If[Length[Union[aux2]] == Length[simbols], Return[True], Return[False]];
  si  longitud  unión              longitud              retorna  verdad···  retorna  falso

 ]

automat9Si = {{1, 2, 3}, {a, b},
   {{1, a, 2}, {1, b, 3}, {2, a, 2}, {2, b, 3}, {3, a, 2}, {3, b, 3}}, 1, {2}};

Ex9[automat9Si]

True

automat9No = {{1, 2, 3, 4}, {a, b}, {{1, a, 2}, {1, b, 4}, {2, a, 4},
    {2, b, 3}, {3, a, 4}, {3, b, 2}, {4, a, 4}, {4, b, 3}}, 1, {2}};

Ex9[automat9No]

False
```

```
(*Exercici 10: Agafar tots els estats com si foren l'inicial i
  apuntar on s'arriba gastant la cadena u des de cada estat inicial.
   Tindre una llista amb tots els estats finals
  recorreguent la cadena des de cada estat inixial.
   Comprovar si s'ha arribat a tots els estats, es a dir,
si la llista resultant es igual al
 conjunt d'estats. Amb un Union i comparant se pot.
                              unión
  Min 35 del video aprox*)
  mínimo

Ex10[autom_, cadena_] :=
 Module[{i, j, estatsini, trans, aux, resposta, estatActual},
 módulo
  estatsini = autom[[1]]; trans = autom[[3]];
  resposta = {};
  For[i = 1, i ≤ Length[estatsini], i++,
  para cada          longitud
       estatActual = estatsini[[i]];

       For[j = 1, j ≤ Length[cadena], j++,
       para cada          longitud
        aux = Cases[trans, {estatActual, cadena[[j]], _}];
              casos
        If[aux ≠ {}, estatActual = aux[[1, 3]], Return[False]];
        si                                      retorna  falso
   ];
       resposta = Append[resposta, estatActual];
                  añade

  ];
  resposta = Union[{}, resposta];
             unión

  If[resposta == estatsini, Return[True], Return[False]];
  si                        retorna  verdad···  retorna  falso
 ]

automat10 = {{1, 2, 3, 4}, {a, b}, {{1, a, 2}, {1, b, 1}, {2, a, 3},
    {2, b, 2}, {3, a, 4}, {3, b, 3}, {4, a, 1}, {4, b, 4}}, 1, {2}};

cadena10 = {b, b, a};

Ex10[automat10, cadena10]
True
```

```
(*Exercici 11*)
```

```
Ex11[autom_, cadena_] :=
 Module[{i, j, estatsini, trans, aux, resposta, estatActual},
  módulo
  estatsini = autom[[1]]; trans = autom[[3]];
  resposta = {};
  For[i = 1, i ≤ Length[estatsini], i++,
   para cada          longitud
       estatActual = estatsini[[i]];

       For[j = 1, j ≤ Length[cadena], j++,
        para cada          longitud
        aux = Cases[trans, {estatActual, cadena[[j]], _}];
              casos
        If[aux ≠ {}, estatActual = aux[[1, 3]], Return[False]];
         si                                     retorna  falso
   ];
       resposta = Union[resposta, Append[resposta, estatActual]];
                  unión           añade
       If[Length[resposta] > 1, Return[False]];
        si  longitud                retorna  falso
  ];
  Return[True];
   retorna  verdadero
 ]

automat11 = {{1, 2, 3, 4}, {a, b}, {{1, a, 2}, {1, b, 2}, {2, a, 2},
    {2, b, 3}, {3, a, 3}, {3, b, 4}, {4, a, 4}, {4, b, 1}}, 1, {1}};

cadena11Si = {a, b, b, b, a, b, b, b, a};
cadena11No = {a, b, b, b, a, b, b, b, b};

Ex11[automat11, cadena11Si]
True

Ex11[automat11, cadena11No]
False
```