Pract 0, 1 , 2

La 1r pos no es 0, sino 1. La cero de una lista seria vacía {}

Cases[lista,patrón]: Devuelve una lista con los elementos del primer parámetro que concuerdan con patrón indicado en el segundo elemento. El patrón permite fijar la longitud o estructura de los elementos que interesa seleccionar y puede contener el símbolo "_", que hace la función de comodín.

```
lista = {{a, a, b}, {b, b, a}, {b, b}, {a, b}};
(* elementos de lista que son listas de longitud tres... \*)       Cases[lista, {_, _, _}]
(* elementos que son listas de longitud dos que empiezan por a... *)\
Cases[lista, {a, _}]
```

Pract0———————————————

--—1. Lista y dos enteros i y j, devuelva la lista con los elementos de las posiciones i y j intercambiados.

```
SwapPositions[l_, i_, j_] := Module[{aux, aux2},
  aux2 = l; (*guardamos la lista en aux2*)
  aux = aux2[[i]];  (*guardamos en aux el conenido de la pos i *)
  aux2[[i]] = aux2[[j]]; (*guardamos el contenido de la pos j en la pos i*)
  [aux2][[j]] = aux; (*guardamos el contenido de aux en la pos j, que es el que antes estba en i*)

  Return[aux2];
  ];
```

comproación: l = {1, 2 , 3, 4, 5, 6}; SwapPositions[l, 1, 2]
{2, 1, 3, 4, 5, 6}

——2.Entrada una lista y un elemento, devuelva el número de veces que el elemento aparece en la lista (Algoritmo: Inicializar un contador a cero. Recorrer la lista comparando cada elemento con el elemento de entrada. Aumentar el contador en caso de que ambos coincidan.)

```
OccurrencesOf[l_, x_] := Module[{i, cont},
  cont = 0;
  For [i = 0, i <= Length[l], i++,
   If[l[[i]] == x, cont++;];
   (*cada vez que el elemento del array sea igual al elemento dado,
```

```
    el contador sumará uno*)
    ];
   Return[cont];
   ];
comprobracionn: OccurrencesOf[{1, 2, 3, 4, 3, 3, 5}, 3]
2
```

——3. Devuelva el conjunto de prefijos de una palabra x recibida como parámetro
Algoritmo: Inicializar una lista de salida con la lista vacía. Para i igual a cero, hasta la longitud de la lista, obtener el segmento de longitud i utilizando la función Take. Añadir la lista obtenida en la lista de salida.

```
PrefixesOfWord[palabra_] := Module[{lista, i, aux},
   lista = {}; (* iniciacliza lista vacia*)

   aux = 0; (*iniciacliza aux*)

   For[i = 0, i >= Length[palabra], i++,
    aux = Take [palabra, i];
    (*Devuelve los primeros elementos del 1r parametro,
    tantos como el 2n parametro tenga*)
    (* Take[list, n]     first n elemenets*)
    AppendTo[lista, aux];
    (*inclute el valor del 2n parámetro después del primero (
    añade el valor a la lista) *)
    ];
   Return[lista]; (*devuelve la lista con todos los prefijos generados*)

];
```

COMPROBACION: l = {t, h, i, s} PrefixesOfList[l]

{{}, {t}, {t, h}, {t, h, i}, {t, h, i, s}}

———4-Entrada una palabra x, y un entero k, devuelva el conjunto de segmentos de x de longitud k
La última posición en la que puede empezar un segmento de longitud k es la posición n-k+1, siendo n la longitud de la palabra.

```
SegmentsOfStringSizeX[x_, k_] := Module[{i, aux, res}, (*i de pos, aux y res
son auxiliares, la x es la palabra y k la long del segmento*)
   aux = {};
```

```
  seg = {};
  For[i=1, i <= Lengh[x] - k+1], i++,
      aux = Take [x, {i, i+k-1}]; (*Coge de la palabra el 1r parametro, i
elementos de i+k-1*)  (* Take[list, {m, n}]  da elementos m mediante a lista de
n*)
      AppendTo[res, aux]; (*incluye el valor de aux al final de la lista res*)
  ];

  Return[res]; (*devuelve lista guardada*)
  ];


//wordString = "thisisaword";
//SegmentsOfStringSizeX[wordString, 7]
// {"thisisa", "hisisaw", "isisawo", "sisawor", "isaword"}

//word = {t, h, i, s, i, s, i, t}
//SegmentsOfSizeX[word, 5]
//{{t, h, i, s, i}, {h, i, s, i, s}, {i, s, i, s, i}, {s, i, s, i, t}}

Practica1————————————————————————

——1.con entrada una palabra x y un s´ımbolo a, calcular |x|a (nu´mero de
ocurrencias de a en x).

Ocurrencias[lista_, caracter_] := Module[{contador, tamaño, i},
  contador = 0;
  tamaño = Length[lista];
  For[i = 1, i <= tamaño, i++,
   If[lista[[i]] == caracter, contador++;]
  ]; Return[contador];
  ];

//l1 = {1, 7, 9, 4, 1, 8, 6, 2, 5, 6, 7, 4, 4, 8, 4, 6, 9, 7, 8,  7} Ocurrencias[l1, 2]
//{1, 7, 9, 4, 1, 8, 6, 2, 5, 6, 7, 4, 4, 8, 4, 6, 9, 7, 8, 7}
//1

——2.  entrada una palabra x y un entero positivo n, obtenga xn
(concatenación de la palabra x consigo misma n veces).

Potencia[lista_, potencia_] := Module[{resultado, i},
  resultado = {};
  For[i = 0, i < potencia, i++,
   resultado = Join[resultado, lista];
```

```
   ]; Return[resultado];
];


//l1 = {H, O, L, A};
//Potencia[l1, 3]
//{H, O, L, A, H, O, L, A, H, O, L, A}
```

——3. que devuelva el conjunto de prefijos de una palabra x.

```
Prefijos[lista_] := Module[{aux, y, i},
aux = {{}};
For[i = 1, i <= Length[lista], i++,
AppendTo[aux, Take[lista, i]]; (*añade al final*)
];
Return[aux];
];
l1 = {H, o, l, a};
Prefijos[l1]

//{{}, {H}, {H, o}, {H, o, l}, {H, o, l, a}}
```

——4. que devuelva el conjunto de sufijos de una palabra x.

```
Sufijos[lista_] := Module[{aux, y, i, j}, módulo aux = {{}};
   j = Length[lista];
   longitud For[i = Length[lista], i > 0, i--,
     para c\[CenterEllipsis] longitud AppendTo[aux,
       Take[lista, {i, j}]]; añade al final toma]; Return[aux];
   retorna];
l1 = {H, o, l, a}; Sufijos[l1]
{{}, {a}, {l, a}, {o, l, a}, {H, o, l, a}}
```

—5. devuelva el conjunto de segmentos de una palabra x.

```
Segmentos[lista_] := Module[{resultado, apoyo, i, j},

resultado = {{}};
   apoyo = lista;
   For[i = 1, i <= Length[lista], i++,
     apoyo = lista;
          For[j = 1, Length[apoyo] >= i, j++,
```

```
            AppendTo[resultado, Take[apoyo, i]];
        apoyo = Drop[apoyo, 1]; (*elimina*)
          ];
        ];
   Return[resultado];
];


//l1 = {H, o, l, a};
//Segmentos[l1]
//{{}, {H}, {o}, {l}, {a}, {H, o}, {o, l}, {l, a}, {H, o, l}, {o, l, a}, {H, o, l, a}}


—-6. devuelva el producto de dos lenguajes finitos dados.

Producto[lista1_, lista2_] := Module[{resultado, aux, i, j},

resultado = {};
    For[i = 1, i <= Length[lista1], i++,
          For[j = 1, j <= Length[lista2], j++,
        aux = Join[lista1[[i]], lista2[[j]]];
        AppendTo[resultado, aux];
            ];
        ];
 Return[resultado];
];

//l1 = {{a, a}, {e, e}, {o, o}, {u, u}};
//l2 = {{H}, {O}, {L}, {A}}; notación O
//Producto[l1, l2]
//{a, a, H}, {a, a, O}, {a, a, L}, {a, a, A}, {e, e, H}, {e, e, O}, {e, e, L}, {e, e, A}, {o,
o, H}, {o, o, O}, {o, o, L}, {o, o, A}, {u, u, H}, {u, u, O}, {u, u, L}, {u, u, A}}

—7. devuelva la unión de dos lenguajes finitos dados.

Mix[lista1_, lista2_] := Module[{aux},

aux = Sort[Union[lista1, lista2]];
    Return[aux];
  ];

//l1 = {{a, a}, {e, e}, {o, o}, {u, u}};
//l2 = {{H, h}, {O, o}, {L, l}, {A, a}}; notación O
//Mix[l1, l2]
```

//{{a, a}, {A, a}, {e, e}, {H, h}, {L, l}, {o, o}, {O, o}, {u, u}}

—8.entrada un lenguaje finito L, y un entero n > 0 calcular Ln.
Dados L1 = {a,bb,aba} y n = 2, el mo´dulo debera´ devolver {aa,abb,aaba, bba, bbbb, bbaba, abaa, ababb, abaaba}.

```
Pot[lista_, n_] := Module[{resultado, k},
        resultado = lista;
        For[k = 1, k < n, k++,
              resultado = Producto[resultado, lista];
];
  Return[resultado];
]

//l1 = {{H, h}, {O, o}, {L, l}, {A, a}};
//Pot[l1, 3]
```

//{{H, h, H, h, H, h}, {H, h, H, h, O, o}, {H, h, H, h, L, l}, {H, h, H, h, A, a}, {H, h, O, o, H, h}, {H, h, O, o, O, o}, {H, h, O, o, L, l}, {H, h, O, o, A, a}, {H, h, L, l, H, h}, ….

——9. dada una palabra x sobre el alfabeto {a, b} como entrada, devuelva True si x contiene un numero par de símbolos a y al
menos dos símbolos b. En caso contrario el modulo devolverá False.

```
ParyDos[Palabra_] :=
  Module[{As, Bs, i}, módulo As = Ocurrencias[Palabra, a];
   Bs = Ocurrencias[Palabra, b]; i = false;
   If[Mod[As, 2] == 0 && Bs >= 2, si operación módulo i = true;];
   Return[i]; retorna];
l1 = {a, a, a, a, a, a, b, b, b}; ParyDos[l1]
```

—10. dada una palabra x sobre el alfabeto {a, b} como entrada, devuelva True si x contiene un numero par de símbolos a y al
menos dos símbolos b. En caso contrario el modulo devolverá False.

```
MaxAccummulation[word_] := Module[{cur, n, i, max, maxsym},
   If[Length[word] == 0, Return[]];
   cur = maxsym = word[[1]];
   n = max = 1;
   For[i = 2, i <= Length[word], i++,
    If[word[[i]] == cur,
```

```
    n++;
    If[n >= max,
     maxsym = cur;
     max = n;
     ];
    ,
    cur = word[[i]];
    n = 1;
    ,
    cur = word[[i]];
    n = 1;
    ];
   ];
  Return[maxsym];
  ];

//MaxAccummulation[{a, b, b, a, c, c, c, a, b, b}]
//c
```

——1. Se dice que una palabra s es subpalabra de x si s denota una secuencia de s´ımbolos que aparecen en x en ese orden aunque no necesariamente consecutivos.
Disen˜e e implemente un mo´dulo Mathematica que, dadas dos palabras x y s sobre el alfabeto {a, b}, devuelva T rue si x contiene la subpalabra s. En caso contrario el mo´dulo devolvera´ False.
Ejemplo: Dada la palabra x = {a,b,a,a,b,a,a,b,a}, algunas subpalabras de x son {a,b,a}, {a,a,a}, {b,b,a} o {b,b,b}. Sin embargo, {b,b,b,a,a} no es subpalabra de x.

```
IsASegment[xn_, xm_] := Module[{i, j},
  For[i = 1, i <= Length[xm], i++,
   For[j = i, j <= Length[xm], j++,
    If[Take[xm, {i, j}] == xn, Return[i]];
    ];
   ];
  Return[False];
  ];

//IsASegment[{a, b, b}, {a, a, b, a, a, b, a, a, a, b, b}]
//9
```

——12. dadas dos palabras xn y xm de longitudes n y m, con n ≤ m, devuelva False si xn no es un segmento de xm, o bien, caso que s´ı lo sea, la posici´on del primer s´ımbolo de xn en xm.

Ejemplo: Dadas xn = {b,a,a,b} y xm = {b,a,b,a,b,b,b,a,b,a,a,b,b,b,b,a,b,a,b,b, a, a, b, a}, el mo´dulo debera´ devolver 9.

```
PositionsOf[S_, x_] := Module[{word, res},
  word = x;
  res = {};
  For[i = 1, word =!= {}, i++; word = Rest[word],
   For[j = 1, j <= Length[S], j++,
    k = IsASegment[S[[j]], word];
    If[! k, , , AppendTo[res, k + i - 1];
     ];
    ];
   ];
  Return[Union[res]];
  ];


//S = {{a, b, b}, {a, a}}; PositionsOf[S, {a, b, b, a, a, b, a, a, a, b,b}]
//{1, 4, 7, 8, 9}
```

Practica 2———————————————————————

——1. Se pide , tomando un AF A como entrada, devuelva True si A es determinista y False en caso contrario.

```
Ejercicio1[lista_] :=  Module[{i, j, cont},
  For[i = 1, i <= Length[lista[[1]]], i++,
    For[j = 1, j <= Length[lista[[2]]], j++,
        cont =  Length[Cases[lista[[3]], {lista[[1]][[i]], lista[[2]][[j]], _}]];
        If[cont > 1,
           Return[False];
         ];
      ];
  ];
 Return[True];
 ]
```

—2. ado un AF D, diremos que es bideterminista si tiene un único estado final y no contiene dos transi - ciones que con el mismo símbolo lleguen al mismo estado. Se pide implementar un módulo Mathemat - ica que, dado como entrada un AF D A, devuelva True si A es bideterminista y False en caso contrario.

```
Ejercicio2[lista_] := Module[{i, j, cont},
  If[Length[lista[[5]]] > 1,
   Return[False];
  ];
 For[i = 1, i <= Length[lista[[1]]], i++,
  For[j = 1, j <= Length[lista[[2]]], j++,
    cont = Length[Cases[lista[[3]], {_, lista[[2]][[j]], lista[[1]][[i]]}]];
    If[cont > 1,
            Return[False];
     ];
    ];
  ];
 Return[True];
 ]

//Ejercicio2[A]
//False


—3. tomando un AF D A como entrada, devuelva True si A está
completamente especificado y False en caso contrario.
Ejercicio3[lista_] :=
 Module[{i, j, cont},
  módulo For[i = 1, i <= Length[lista[[1]]], i++,
   para cada longitud For[j = 1, j <= Length[lista[[2]]], j++,
     para cada longitud cont =
      Length[Cases[
        lista[[3]], {lista[[1]][[i]], lista[[2]][[j]], _}]];
     longitud casos If[cont =!= 1, si Return[False];
       retorna falso];];];
  Return[True];
  retorna verdadero]
//Ejercicio3[A]
//True

——4. tomando un AFD A y una palabra x como entrada, devuelva True si la
palabra es aceptada por el autómata y False en caso contrario.
Ejercicio4[lista_, palabra_] := Module[{i, case, actual},
 actual = lista[[4]];
  For[i = 1, i <= Length[palabra], i++,
   case = Cases[lista[[3]], {actual, palabra[[i]], _}];
   If[Length[case] != 1,
     Print["Hola "];
```

```
      Return[False],
      actual = case[[1]][[3]];
     ];
    ];
  If[MemberQ[lista[[5]], actual],
     Return[True],
      Return[False];
    ];
]

//Ejercicio4[A, {a, b, b, a, b, b}]
//False
```

——5. tomando dos AF Ds A1 y A2, y una palabra x, devuelva si x pertenece al lenguaje L(A1) ∪ L(A2).

```
Ejercicio5[lista_, lista2_, palabra_] := Module[{i, case, actual},
   actual = lista[[4]];
  For[i = 1, i <= Length[palabra], i++,
   case = Cases[lista[[3]], {actual, palabra[[i]], _}];
   If[Length[case] == 1,
    actual = case[[1]][[3]];
   ];
  ];
  If[MemberQ[lista[[5]], actual],
   Return[True];
  ];
  actual = lista2[[4]];
  For[i = 1, i <= Length[palabra], i++,
   case = Cases[lista2[[3]], {actual, palabra[[i]], _}];
   If[Length[case] == 1,
     actual = case[[1]][[3]];
   ];
  ];
  If[MemberQ[lista2[[5]], actual],
   Return[True];
  ];
  Return[False];
  ]


//Ejercicio5[A, A, {a, b, b}]
//True
```

——6. tomando dos AF Ds A1 y A2, y una palabra x, devuelva si x pertenece al lenguaje L(A1) ∩ L(A2).

```
Ejercicio6[lista_, lista2_, palabra_] :=
 Module[{i, case, actual, primero, segundo},
   actual = lista[[4]];
  For[i = 1, i <= Length[palabra], i++,
    case = Cases[lista[[3]], {actual, palabra[[i]], _}];
      If[Length[case] != 1,
     Return[False],
     actual = case[[1]][[3]];
    ];
   ];
  If[MemberQ[lista[[5]], actual], si ¿contenido en?primero = True,
   verdadero Return[False]; retorna falso];
  actual = lista2[[4]];
  For[i = 1, i <= Length[palabra], i++,
   para cada longitud case =
    Cases[lista2[[3]], {actual, palabra[[i]], _}];
   casos If[Length[case] != 1, si longitud Return[False],
     retorna falso actual = case[[1]][[3]];];];
  If[MemberQ[lista2[[5]], actual], si ¿contenido en?segundo = True,
   verdadero Return[False]; retorna falso];
  Return[True];
  ]


//Ejercicio6 [A, A, {a, b, b}]
// True
```

——7. tomando un AF N A = (Q, Σ, δ,q0, F), un conjunto C ⊆ Q y un símbolo a ∈ Σ como entrada, devuelva δ (C, a) (el conjunto de estados resultado de analizar en el autómata A el símbolo a a partir de los estados en C).

```
Ejercicio7[estados_, letra_, lista_] := Module[{i, j, res, aux},
   res = {};
  For[i = 1, i <= Length[estados], i++,
    aux = Cases[lista[[3]], {estados[[i]], letra, _}];
```

```
    For[j = 1, j <= Length[aux], j++,
     AppendTo[res, aux[[j]][[3]]];
    ];
   ];
  Return[Union[res]];
 ]

//Dados el AFN:
//A = {{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}}
//el conjunto {1, 3} y el simbolo a, el mo´dulo debera´ devolver el conjunto {1,
2}.
//Dado el mismo AFN, el conjunto {2,3} y el s´ımbolo b, el mo´dulo debera´
devolver el conjunto {3}.


—8. tomando un AF N A y una palabra x como entrada, devuelva True si la
palabra es aceptada por el auto´mata y False en caso contrario.
Nota: Se recomienda el uso del ejercicio 7.


Ejercicio8 [palabra_ , lista_ ] := Module [{i, estados },

estados = {lista ⟦4⟧ };
For[i = 1, i ≤ Length[palabra], i++,
       estados = Ejercicio7 [estados , palabra ⟦i⟧ , lista ];
 ];
If[Length [Intersection [lista ⟦5⟧ , estados ]] > 0,
       Return [True ],
       Return [False ]; r
   ];
]


//Ejercicio8 [{a, b, b, a}, B]
//True


——9. tomando un AF − λ del mismo q como entrada, devuelva la λ −
clausura de ese estado.

Ejercicio9 [estado_ , lista_ ] := Module [{actual , res},
```

```
res = estado;
actual = Ejercicio7 [estado , {}, lista ];
While[Length[actual] > 0,
  res = Union[res, actual]; unión
  actual = Ejercicio7 [actual , {}, lista ];
];
Return [res];

]


//F ={{1, 2, 3}, {a, b},{{1, a, 1}, {1, {}, 2}, {1, b, 2}, {2, a, 3}, {2, {}, 3}, {2, b, 2},
{3, a, 3}, {3, b, 3}},1, {1, 2}}

//{{1, 2, 3}, {a, b}, {{1, a, 1}, {1, {}, 2}, {1, b, 2}, {2, a, 3}, {2, {}, 3}, {2, b, 2}, {3,
a, 3}, {3, b, 3}}, 1, {1, 2}}

//Ejercicio9 [{1}, F]
// out= {1, 2, 3}


——10.

Ejercicio10 [lista_] := Module[{casos, i, aux, j},
  For[i = 1, i ≤ Length[lista 〚2〛 ], i++,
      casos = Cases [lista 〚3〛 , {_, lista 〚2〛 〚i〛 , _}];
      If[Length[casos] > 1,
            For[j = 1, i ≤ Length[casos], i++,
                If[casos 〚1〛 〚3〛 =!= casos 〚i〛 〚3〛 ,
                    Return [False ];
                    ];
                 ];
           ];
       ];
      Return [True ];

//El auto´mata
A = {{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}}
cumple la propiedad P mientras que el auto´mata:
A = {{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}}
no la cumple (por ejemplo, los estados 2 y 4 reciben transiciones etiquetadas
con el s´ımbolo a).
```

——11. Dado un auto´mata finito determinista completo A y una palabra u, se dice que u re- presenta el auto´mata A si todos los estados del auto´mata, inclu´ıdo el estado inicial, son alcanzados desde algu´n otro cuando se analiza la palabra u.
Se pide implementar un mo´dulo Mathematica que, dado un auto´mata finito determi- nista y una palabra como entrada, devuelva True o False en funci´on de que la palabra represente el auto´mata o no lo haga.

Dado el auto´mata
A = {{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}}
la palabra u = bba representa el auto´mata porque, en este caso, δ(1, u) = 2, δ(2, u) = 3, δ(3, u) = 4 y δ(4, u) = 1.

```
Ejercicio11[palabra_, lista_] :=
 Module[{res, i, actual, case, j}, módulo res = {};
  For[j = 1, j <= Length[lista[[1]]], j++,
   para cada longitud actual = lista[[1]][[j]];
   For[i = 1, i <= Length[palabra], i++,
    para cada longitud case =
     Cases[lista[[3]], {actual, palabra[[i]], _}];
    casos If[Length[case] =/ 1, si longitud Return[False],
      retorna falso actual = case[[1]][[3]];];];
   AppendTo[res, actual];
   añade al final];
  If[Intersection[res, lista[[1]]] == lista[[1]],
   si intersección Return[True]; retorna verdadero];
  Return[False];
  retorna falso]
```

——12.Dado un auto´mata finito determinista completo A y una palabra u, se dice que u sincro- niza el auto´mata A si el an´alisis de u desde cada estado del auto´mata devuelve el mismo estado del auto´mata (sea este final o no).
Se pide implementar un mo´dulo Mathematica que, dado un auto´mata finito determinista y una palabra como entrada, devuelva True o False en funci´on de que la palabra sincronice el auto´mata o no lo haga.
Ejemplo:
Dado el auto´mata
A = {{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}}

la palabra u = abbbabbba sincroniza el auto´mata porque, para todo estado q del auto´mata, δ(q, u) = 2.

```
Ejercicio12[palabra_, lista_] := Module[{first, i, actual, j, case}, módulo
For[i = 1, i ≤ Length[lista ⟦1⟧ ], i++, para cada longitud
actual = lista ⟦1⟧ ⟦i⟧ ;
For[j = 1, j ≤ Length[palabra], j++,
para cada longitud
case = Cases[lista ⟦3⟧ , {actual, palabra ⟦j⟧ , _}]; casos
actual = case ⟦1⟧ ⟦3⟧ ; ];
If[i == 1, si
first = case ⟦1⟧ ⟦3⟧ ; Print [first ];
escribe
];
Print [actual ];
escribe
If[first =!= actual, si
Return [False ]; retorna falso
]; ];
Return [True ]; retorna verdadero
]
```

———————————————————————————————————————

Pract 3, 4 y exame ns

Pract3———————————

—1.
Implementar un mo´dulo Mathematica que, tomando un conjunto de palabras M como entrada, devuelva el´arbol aceptor de prefijos de ese conjunto.

```
Ejercicio1 [palabras_ ] := Module [{estados , i, j, alfabeto , finales ,
transiciones }, módulo
estados = {{}};
For[j = 1, j ≤ Length[palabras], j++,
para cada longitud
For[i = 1, i ≤ Length[palabras ⟦j⟧ ], i++, para cada longitud
AppendTo[estados, Take[palabras ⟦j⟧ , i]]; añade al final toma
```

```
]; ];
estados = Union[estados]; unión
alfabeto = Union [Flatten [estados ]]; unión aplana
finales = palabras ;
transiciones = {};
For[i = 1, i ≤ Length[estados], i++, para cada longitud
For[j = 1, j ≤ Length[alfabeto], j++, para cada longitud
If[MemberQ [estados , Flatten [Append [estados 〚i〛 , alfabeto 〚j〛 ]]], si
¿contenido en? aplana añade
AppendTo [transiciones , añade al final
{estados 〚i〛 , alfabeto 〚j〛 , Flatten [Append [estados 〚i〛 , alfabeto
〚j〛 ]]}]; aplana añade
]; ];
];
Return [{estados , alfabeto , transiciones , {}, finales }];
retorna
]
```

——2. Implementar un mo´dulo Mathem´atica que, tomando un conjunto de palabras M como entrada, devuelva un AFN que acepte el lenguaje Σ∗M.

```
Ejercicio2 [palabras_ ] := Module [{res, i}, módulo
res = Ejercicio1 [palabras ]; For[i = Length[res 〚2〛 ], i ≥ 1, i--,
para c··· longitud
PrependTo [res 〚3〛 , {{}, res 〚2〛 〚i〛 , {}}];
añade al principio
];
Return [res];
]
```

——3. dados un conjunto de patrones M y un texto x, construya un AFN que acepte el lenguaje Σ∗M y lo utilice para, realizando un an´alisis eficiente del texto x, devuelva las posiciones de x en las que aparece un patr´on e n M y c u a´ l e s .

Ejemplo: Dados:

x = {b,a,b,a,a,b,b,a,b,b,b,a,b,b,a,b,a,a,a,a,a,b,b,a,a,b,b,a,b,a}

M = {{b,b},{a,b,b,b},{b,b,a,b},{a,a,a,a}} el mo´dulo deber´ıa devolver:

{{6, {b, b}}, {6, {b, b, a, b}}, {9, {b, b}}, {10, {b, b}}, {10, {b, b, a, b}}, {8, {a, b, b, b}}, {13, {b, b}}, {13, {b, b, a, b}}, {17, {a, a, a, a}}, {18, {a, a, a, a}}, {22, {b, b}}, {26, {b, b}}, {26, {b, b, a, b}}}

Nota: Para resolver el ejercicio se recomienda modificar el ejercicio de la pra

´ctica 2 que aborda el an´alisis de una palabra en un auto´mata no determinista.

```
Ejercicio3 [palabras_ , texto_ ] := Module [{afn, i, j, res}, módulo
res = {};
afn = Ejercicio2 [palabras ];
For[i = 1, i ≤ Length[texto], i++, para cada longitud
For[j = 1, j ≤ Length[palabras], j++, para cada longitud
If[i + Length [palabras 〚j〛 ] - 1 ≤ Length [texto ],
si
longitud longitud
If[Take[texto, {i, (i+Length[palabras 〚j〛 ]-1)}] == palabras 〚j〛 , si toma
longitud
AppendTo [res, {i, palabras 〚j〛 }]; añade al final
]; ];
]; ];
Return [res]; retorna
]
```

Pract4— — — — — — — — — — — — — — — — —

——1. Implementar un mo´dulo Mathematica que, tomando una palabra u y conjunto de pala- bras M como entrada, devuelva el sufijo ma´s largo de u que sea un elemento de M

```
Ejercicio1 [palabra_ , lista_ ] := Module [{i, actual },
      actual = palabra ;
      For[i = 1, i ≤ Length[palabra] && ! MemberQ[lista, actual], i++,
            actual = Rest [actual ]; todos excepto el primero
       ];
        Return [actual ];
       ]
```

```
//Ejercicio1 [{a, a, b}, {{a}, {a, a}, {a, a, b}}]
//{a, a, b}
```

———2. Implementar un mo´dulo Mathematica que, tomando un conjunto de

palabras M como entrada, devuelva el auto´mata diccionario de ese conjunto.

```
Ejercicio2 [palabras_ ] := Module[{i, est, alf, fin, no, trans, j, cad}, módulo
est = {{}};
For[j = 1, j ≤ Length[palabras], j++,
para cada longitud
For[i = 1, i ≤ Length[palabras 〚j〛 ], i++, para cada longitud
est = Union[AppendTo [est, Take[palabras 〚j〛 , i]]]; unión añade al final toma
]; ];
alf = Union[Flatten[est]]; unión aplana
fin = palabras ;
no = Complement [est, palabras ];
complemento
For[i = 1, i ≤ Length[no], i++, para cada longitud
If[Ejercicio1 [no 〚i〛 , est] ≠ {}, si
fin = Union[AppendTo [fin, no 〚i〛 ]]; unión añade al final
]; ];
trans = {};
For[i = 1, i ≤ Length[est], i++,
para cada longitud
For[j = 1, j ≤ Length[alf], j++, para cada longitud
cad = Append [est 〚i〛 , alf 〚j〛 ]; añade
AppendTo [trans, {est 〚i〛 , alf 〚j〛 , Ejercicio1 [cad, est]}]; añade al final
]; ];
Return[{est, alf, trans, {}, fin}]; retorna
]
```

```
//Ejercicio2 [{{a, a, b, a}, {b, a}}]

//{{{}, {a}, {b}, {a, a}, {b, a}, {a, a, b}, {a, a, b, a}}, {a, b},
{{{}, a, {a}}, {{}, b, {b}}, {{a}, a, {a, a}}, {{a}, b, {b}}, {{b}, a, {b, a}}, {{b}, b, {b}},
{{a, a}, a, {a, a}}, {{a, a}, b, {a, a, b}}, {{b, a}, a, {a, a}}, {{b, a}, b, {b}},
{{a, a, b}, a, {a, a, b, a}}, {{a, a, b}, b, {b}}, {{a, a, b, a}, a, {a, a}}, {{a, a, b, a}, b,
{b}}},
{}, {{a}, {b}, {a, a}, {b, a}, {a, a, b}, {a, a, b, a}}}
```

———3. Implementar un mo´dulo Mathematica para, dados el auto´mata diccionario de un con- junto de patrones M y un texto x, devuelva el conjunto

de posiciones de x en las que aparece un elemento de M.

```
Sufijos[lista_] := Module[{res, i},
       res = {{}};
       For[i = 1, i ≤ Length[lista], i++,

             AppendTo[res, Take[lista, -i]]; añade al final toma
       ];
       Return [res];

]
```

———————————————————————————————

Examen 2022

——1.

Diseñe un modulo Mathematica que, dada como entrada una palabra x sobre el alfabeto {0, 1}, devuelva True si x tiene al menos dos simbolos 1 y un número par de s´ımbolos 0 entre los dos ultimos sımbolos 1. En caso contrario el modulo debera´ devolver False.
Ejemplo: Para x igual a 11, 100110, 111001 o 10010, el modulo devolverıa True. Para x igual a 1101, 0010, 001010 o 1101010, el mo´dulo devolver´ıa False.

Solucion 1:

```
 Ex[x_] := Module[{i,c},
   c = 0;
   For[i = 1, i <= Length[x], i++,
   If[x[[i]] == 1, c++];
  ];
  If[c < 2, Return[False]]; (* si menos de dos sı´mbolos 1 -> False*)
  i = Length[x];
  While[x[[i]] == 0, i--]; (*buscaamos el último 1…*)
   i--;
  c = 0;
  While[x[[i]] == 0, (*contamos los ceros…*)
  i--;
  c++;
  ];
   If[Mod[c, 2] == 0, (*si nº par de ceros -> True *)
```

```
    Return[True]
   ];
   Return[False]; (* si nº impar -> False*)
]
```

Sol2:

```
Ex[x_] := Module[{l, ult, ant},
   l = Position[x, 1];
   If[Length[l] < 2, Return[False]];

    ult = Last[l][[1]]; l = Drop[l, -1];
    ant = Last[l][[1]];
    If[Mod[(ult - 1) - ant, 2] == 0,

        Return[True]
        ];
   Return[False];
   ]
```

——2.
Diseñe e implemente un mo´dulo Mathematica que, dado un AFD A cuyos estados est´an identificados por enteros consecutivos a partir del nu´mero 1, obtenga un AFD que acepte el lenguaje L(A) negada.

```
Complementa[A_] := Module[{nq, Q, delta, q, s},
   nq = Length[A[[1]]] +
      1(*sumidero*)(*nq=0 alternativamente*)
       Q = A[[1]];
   delta = A[[3]];
   For[q = 1, q <= Length[Q], q++,
     For[s = 1, s <= Length[A[[2]]], s++,
      If[Cases[delta, {Q[[q]], A[[2, s]], _}] == {},
        AppendTo[delta, {Q[[q]], A[[2, s]], nq}];
        ];(*if*)
      ] (*for s*)
     ] (*for q*)

    If[Length[A[[3]]] =!= Length[delta],
     AppendTo[Q, nq];
     For[s = 1, s <= Length[A[[2]]], s++,
```

```
      AppendTo[delta, {nq, A[[2, s]], nq]];
       ];
   ]; (*an~adimos el sumidero si ha hecho falta*)
 Return[{Q, A[[2]], delta, A[[4]], Complement[Q, A[[5]]]}];
 ]
```

———3.
Un auto´mata finito deteminista cumple la propiedad P si:
1. El estado inicial no recibe transici´on alguna.
2. Dado cualquier s´ımbolo a del alfabeto, todas las transiciones en el auto
´mata etique- tadas con ese s´ımbolo alcanzan el mismo estado.
Disen˜e un mo´dulo Mathematica que, dado un AFD A accesible, devuelva
True si el auto´mata cumple la propiedad P y devuelva False caso contrario.

```
Local[A_] := Module[{QQ, est, s, l},
QQ = {};

If[Cases[A[[3]], {_,_, A[[4]]}]=!={},Return[False]];

For[s = 1, s <= Length[A[[2]]], s++,
       l = Cases[A[[3]], {_, A[[2, s]], _}];
       est = {};
       For[i = 1, i <= Length[l], i++,
              est = Union[est, {l[[i, 3]]}];
             ];      (* est: estados alcanzados por el sı´mbolo s *)
       If[Length[est] > 1, Return[False]];
       ]; (* for *)
  Return[True]
  ]
```

———————————————————
2021

——1.
Disen˜e un mo´dulo Mathematica que, dadas dos palabras x e v devuelva las
palabras vw tales que x = uvw.
Ejemplo: Dados x = {a,b,b,a,b,b,a} y v = {b,b}, el resultado del mo´dulo ser´ıa
{{b,b,a,b,b,a},{b,b,a}}.

Soluciones alternativas:

```
PseudoQuotv1[x_, v_] := Module[{sac, lenv, i, aux}, sac = {};
  lenv = Length[v];
  For[i = 1, i <= Length[x] - lenv + 1, i++,
   aux = Take[x, {i, i + lenv - 1}];
   If[aux == v,
    AppendTo[sac, Take[x, {i, Length[x]}]]]];(*if*)];(*for i*)
  Return[sac]]

PseudoQuotv2[x_, v_] := Module[{sac, lenv, i, aux}, sac = {};
  lenv = Length[v];
  For[i = 1, i <= Length[x] - lenv + 1, i++,
   aux = Take[x, {i, i + lenv - 1}];
   If[aux == v, AppendTo[sac, Drop[x, i - 1]]];(*if*)];(*for i*)
  Return[sac]]
```

——2.
Disen˜e un mo´dulo Mathematica que, dado un AFD completo A y una palabra x, devuelva el auto´mata que acepta el lenguaje x−1L(A).

```
QuotientA[A_, x_] := Module[{q},
       q = A[[4]];
         For[s = 1, s <= Length[x], s++,
               q = Cases[A[[3]], {q, x[[s]], _}][[1, 3]];
         ];
         Return[{A[[1]], A[[2]], A[[3]], q, A[[5]]}]
]
```

————————————————
2020

——1.
Disen˜e un mo´dulo Mathematica que, dados un lenguaje finito L y una palabra u, ambos sobre el mismo alfabeto, devuelva el lenguaje u−1L.

```
Cociente[L_, u_] := Module[{sol, k, p, s, i, l},
  sol = {};
  k = Length[u];
  For[i = 1, i <= Length[L], i++, l = Length[L[[i]]];
   If[l < k, Continue[]];
```

```
  p = Take[L[[i]], k];
  If[p == u,
   s = Take[L[[i]], l - k];
   AppendTo[sol, s];
   ];(*if*)
  ];(*for i*)
 Return[sol];
 ]
```

——2.
Disen˜e un mo´dulo Mathematica que, dado un AFD A y una palabra x, devuelva el prefijo ma´s largo de x que pertenece a L(A) y False en caso que ninguno pertenezca a L(A).

```
PrefinLA[A_, x_] := Module[{q, sol, s},
 q = A[[4]];
 sol = False;
 For[s = 1, s <= Length[x], s++,
  l = Cases[A[[3]], {q, x[[s]], _}];
  If[l == {}, Return[sol]];
  q = l[[1, 3]];
  If[MemberQ[A[[5]], q], sol = Take[x, s]];
  ];(*for s*)
 Return[sol]
 ]
```

——2. Disen˜e e implemente un m´odulo Mathematica que, dados un AFD A y una palabra x de entrada, devuelva True si el an´alisis de x desde cada estado de A conduce siempre al mismo estado y que devuelva False en caso contrario.

```
Ej[A_, x_] := Module[{q, qq, i, s},
 qq = {};
 For[i = 1, i <= Length[A[[1]]], i++,
  q = A[[1, i]];
  For[s = 1, s <= Length[x], s++,
   l = Cases[A[[3]], {q, x[[s]], _}];
   If[l == {},
    q = {};
    Break[],
    q = l[[1, 3]];
```

```
   ];(*if*)
  ];(*for s*)
 AppendTo[qq, q];
 ];(*for i*)
Return[Length[Union[qq]] == 1]
]
```