Lab 3

```
In[ ]:= M = {{a}, {b, b}, {a, a, a}, {a, a, b}, {a, b, b},
          {a, a, a, b}, {a, a, b, a}, {a, a, b, a, b}, {a, a, b, b, b}};
```

Exercise 1 - Prefixes of a list of words

```
In[ ]:= Prefixes[M_] := Module[{list, i, aux},
                        módulo
          list = {{}};
          aux = M;

          For[i = 1, i ≤ Length[aux], i++,
          para cada          longitud
           While[Length[aux[[i]]] > 0,
           mient···longitud
              AppendTo[list, aux[[i]]];
              añade al final
              aux[[i]] = Drop[aux[[i]], -1];
                                elimina
            ];
          ];
          Return[Union[list]];
          retorna   unión
         ];
```

```
In[ ]:= Prefixes[M]
```

```
Out[ ]= {{}, {a}, {b}, {a, a}, {a, b}, {b, b}, {a, a, a}, {a, a, b}, {a, b, b},
         {a, a, a, b}, {a, a, b, a}, {a, a, b, b}, {a, a, b, a, b}, {a, a, b, b, b}}
```

Exercise 2 - Suffixes of a list of words

```
In[ ]:=
Suffixes[M_] := Module[{list, i, aux},
                módulo
  list = {{}};
  aux = M;

  For[i = 1, i ≤ Length[aux], i++,
  para cada      longitud
   While[Length[aux[[i]]] > 0,
   mient···longitud
     AppendTo[list, aux[[i]]];
     añade al final
     aux[[i]] = Rest[aux[[i]]];
                todos excepto el primero
    ];
  ];
  Return[Union[list]];
  retorna  unión
 ];
```

```
In[ ]:= Suffixes[M]
```

```
Out[ ]= {{}, {a}, {b}, {a, a}, {a, b}, {b, a}, {b, b}, {a, a, a},
  {a, a, b}, {a, b, a}, {a, b, b}, {b, a, b}, {b, b, b}, {a, a, a, b},
  {a, a, b, a}, {a, b, a, b}, {a, b, b, b}, {a, a, b, a, b}, {a, a, b, b, b}}
```

Exercise 3 - Generate Prefix Acceptor Tree from a word list

An automata composed by the alphabet used in the language (updates each time a word is analyzed.
The set of states is the set of prefixes of the language. Only states that correspond to a word in the lenguage are marked final
The initial state is

The way to analyze is the following. Given the list of prefixes, build the states.
Add lambda (to the states, as initial state and without transitions. Check if it needs to be added to final.

A={Prefixes[M],{a,b},{},{},M}
[FOR] a given prefix:
1-. Add the transition from the parent to the son. {current minus last letter, last-letter, current}

Once all prefixes have been analyzed, change the name of the states for shorter indicators, such as numbers (use replace function).

```
AcceptorTree[M_] := Module[{A, i(*parent,curr*)},
                    módulo
   A = {Prefixes[M], Union[Flatten[M]], {}, {}, M};
                     unión   aplana
   For[i = 2, i ≤ Length[A[[1]]], i++,
   para cada        longitud
    AppendTo[A[[3]], {Drop[A[[1, i]], -1], Last[A[[1, i]]], A[[1, i]]}];
    añade al final          elimina              último
   ];
   (*The following commented code turns states to numbers for simplicity*)
   (*A[[4]]=1;
   For[i=1,i≤Length[A[[1]]],i++,
   para cada     longitud
    A=Replace[A,{A[[1,i]]→i},{2,3}];
      sustituye
   ];*)
   (*END of conversion*)
   Return[A];
   retorna
  ];
```

```
AcceptorTree[M]
```

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

Exercise 4 - Generate NFA from a word list

```
AutomataFromText[M_] := Module[{A, i},
                        módulo
   A = AcceptorTree[M];
   For[i = 1, i ≤ Length[A[[2]]], i++, PrependTo[A[[3]], {A[[4]], A[[2, i]], A[[4]]}];];
   para cada        longitud                añade al principio
   Return[A];
   retorna
  ];
```

```
AutomataFromText[M]
```

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

Exercise 5 - Scan a word using a NFA

The scan must follow each branch until it drops or the scan ends. If it ends with any branch in a final state, the word is accepted.

The scan begins in the initial state.
For each letter in the word.
  For each current state:
    1-. Check transitions with current state and letter.
    2-. Delete the current state from the state list
    3-. Add the result of all existing transitions.

When the execution ends. states UNION finals is not empty, true, else false.

```
ScanWordNFA[w_, A_] := Module[{states, newStates, i, j, k, trans},
                       módulo
   states = {A[[4]]};
   For[i = 1, i ≤ Length[w], i++,
   para cada          longitud
    newStates = {};
    For[j = 1, j ≤ Length[states], j++,
    para cada          longitud
     trans = Cases[A[[3]], {states[[j]], w[[i]], _}];
             casos
     For[k = 1, k ≤ Length[trans], k++, AppendTo[newStates, trans[[k, 3]]];];
      para cada          longitud                añade al final
    ];
    states = newStates;
   ];
   Return[Intersection[states, A[[5]]] ≠ {}];
   retorna   intersección
  ];
```

```
ScanWordNFA[{b, b, a, a, a, b, a, b, b}, AutomataFromText[M]]
```

```
True
```

Testing...

Exercise 6 - Return instances of a word list in a text ?

SPECIFICATION

Modify the exercise 5 to return a number each time a final state is reached.

Keep a counter for each state {{state, count},{state,count}...} to tell how many letters has the word accepted.

Add the positions to a list (can be repeated, as a word can be a prefix of another word). Return that list.

The scan must begin on each letter. For every other than the first, the scan must continue from the previous state list + the initial state.

ALGORITHM

M: a pattern set

x: a text

pos: the positions where a pattern in M starts in the text x

states: current states, each one with the number of scanned letters.

The states list

For each letter in the text, do

1-. Add the initial state to the list of current states, with counter = 0 (THIS letter has begun THAT state)

2-. For each state in the state list do

1-. Check transitions from that state with the given letter

2-. For each transition, add the 3rd component to the state list, and increment its counter

3-. For each state in the new state list do

1-. If the state (component 1) is in the final states list, add its second component to the pos list.

```
DetectWordsInText[M_, x_] := Module[{A, states, newStates, i, j, k, trans, pos},
                             módulo
  pos = {};
  A = AcceptorTree[M];
  states = {};
  For[i = 1, i ≤ Length[x], i++,
  para cada          longitud
   AppendTo[states, {A[[4]], 0}];
   añade al final
   newStates = {};
   For[j = 1, j ≤ Length[states], j++,
   para cada          longitud
    trans = Cases[A[[3]], {states[[j, 1]], x[[i]], _}];
            casos
    For[k = 1, k ≤ Length[trans], k++,
    para cada          longitud
     AppendTo[newStates, {trans[[k, 3]], states[[j, 2]] + 1}];];
     añade al final
   ];
   states = newStates;
   For[j = 1, j ≤ Length[states], j++,
   para cada          longitud
    If[Intersection[A[[5]], {states[[j, 1]]}] ≠ {},
    si  intersección
      AppendTo[pos, i - states[[j, 2]] + 1];];];
      añade al final
  ];
  Return[Sort[pos]];
  retorna  ordena
 ];
```

This analysis should output {1, 1, 2}, for the segments {a},{a,b,b},{b,b}

```
DetectWordsInText[M, {a, b, b}]
```

```
{1, 1, 2}
```

```
ScanWordNFA[M_, x_] := Module[{states, newStates, i, j, k, trans, A, pos, finals},
                       módulo
  A = AutomataFromText[M];
  states = {A[[4]]};
  pos = {};
  For[i = 1, i ≤ Length[x], i++,
  para cada          longitud
   newStates = {};
   For[j = 1, j ≤ Length[states], j++,
   para cada          longitud
    trans = Cases[A[[3]], {states[[j]], x[[i]], _}];
            casos
    For[k = 1, k ≤ Length[trans], k++, AppendTo[newStates, trans[[k, 3]]];];
     para cada          longitud                 añade al final
   ];
   states = newStates;
   finals = Intersection[states, A[[5]]];
            intersección
   For[j = 1, j ≤ Length[finals], j++,
   para cada          longitud
    AppendTo[pos, i + 1 - finals[[j]]];];
     añade al final
  ];
  Return[Sort[pos]];
  retorna  ordena
 ];
```