notebook

March 9, 2021

[1]:
```
:l Automata
import Automata
```

1 Autómata finito determinista

1.0.1 Definición

Definición de un autómata que reconoce el lenguaje a(a+b)*

$$L_{EmpiezaA} := \{aw : w \in \Sigma^*\} = b(a+b)*$$

[2]:
```
vocab = "ab"
nodes = [Q 0, Q 1, Trash]
initial = Q 0
terminals = [Q 1]


delta :: Char -> Status -> Status
delta 'a' (Q 0) = Q 1
delta _ (Q 1) = Q 1
delta _ _ = Trash

atEmpiezaA = AFD vocab nodes initial delta terminals
```

Definición de un autómata que reconoce el lenguaje

$$L_{EmpiezaB} := \{bw : w \in \Sigma^*\} = b(a+b)*$$

[3]:
```
deltab :: Char -> Status -> Status
deltab 'b' (Q 0) = Q 1
deltab _ (Q 1) = Q 1
deltab _ _ = Trash

atEmpiezaB = AFD vocab nodes initial deltab terminals
```

Definición de un autómata que reconoce el lenguaje

$$L_{atApar} := \{w : |w|_a \ mod \ 2 \ = \ 0\}$$

[4]:
```haskell
deltad :: Char -> Status -> Status
deltad 'a' (Q 0) = Q 1
deltad 'b' (Q 0) = Q 0

deltad 'a' (Q 1) = Q 0
deltad 'b' (Q 1) = Q 1

deltad _ _ = Trash

terminalsd = [Q 0]

atApar = AFD vocab nodes initial deltad terminalsd
```

Definición de un autómata que reconoce el lenguaje

$$L_{ab} := \{(ab)^n : n \geq 0\} = (ab)*$$

[5]:
```haskell
deltac :: Char -> Status -> Status
deltac 'a' (Q 10) = Q 11
deltac 'b' (Q 11) = Q 10
deltac _ _ = Trash

initialc = Q 10
nodesc = [Q 10, Q 11, Trash]
terminalsc = [Q 10]

atab = AFD vocab nodesc initialc deltac terminalsc
```

1.0.2 Funciones sobre un autómata

¿Reconoce una palabra? (isRenewed :: Automata -> [Char] -> Bool)

[6]:
```haskell
isRenewed atEmpiezaA "aabb"
```

```
True
```

[7]:
```haskell
isRenewed atEmpiezaA "bab"
```

```
False
```

[8]:
```haskell
isRenewed atEmpiezaB "aabb"
```

```
False
```

[9]:
```haskell
isRenewed atEmpiezaB "bab"
```

```
True
```

[10]:
```haskell
isRenewed atab "aabb"
```

```
False
```

[11]:
```
isRenewed atab "bab"
```

```
False
```

[12]:
```
isRenewed atab "abab"
```

```
True
```

[13]:
```
isRenewed atApar "abaabb"
```

```
False
```

[14]:
```
isRenewed atApar "abaabba"
```

```
True
```

Función delta Dado un autómata, una palabra y una lista de estados devuelve la lista de estados a los que llega

[15]:
```
deltaB atEmpiezaA "aa" [Q 0]
```

```
[Q 1]
```

[16]:
```
deltaB atab "bab" [Q 10, Q 11]
```

```
[Trash,Q 10]
```

Función normalizeNodes (normalizeNodes :: AFD -> AFD) Dado un AFD retorna el AFD con los estados renombrados a $q_0, q_1, ..., q_n$

[17]:
```
atab' = normalizeNodes atab
atab'
```

```
ab
[Q 0,Q 1,Q 2]
Q 0
[(Q 0,'a',Q 1),(Q 0,'b',Q 2),(Q 1,'a',Q 2),(Q 1,'b',Q 0),(Q 2,'a',Q 2),(Q␣
 ↪2,'b',Q 2)]
[Q 0]
```

[18]:
```
isRenewed atab' "aabb"
isRenewed atab' "bab"
isRenewed atab' "abab"
```

```
False
```

```
False
```

```
True
```

Función reduce (reduce :: AFD -> AFD) Elimina nodos no alcanzables. El ejemplo se verá después.

1.0.3 Operaciones sobre AFD

Suma de autómatas (orAFD :: AFD -> AFD -> AFD)

[19]:
```
atEmpiezaAoB = orAFD atEmpiezaA atEmpiezaB

isRenewed atEmpiezaAoB "abaabbbb"
isRenewed atEmpiezaAoB "babaaa"

deltaB atEmpiezaAoB "abaabbbb" [QP (Q 0) (Q 0)]
```

```
True

True

[QP (Q 1) Trash]
```

[20]:
```
atEmpiezaBoesAB = orAFD atEmpiezaB atab

isRenewed atEmpiezaBoesAB "abababababab"
isRenewed atEmpiezaBoesAB "abba"
isRenewed atEmpiezaBoesAB "babbb"
```

```
True

False

True
```

[21]:
```
atEmpiezaBoesAB
```

```
ab
[Trash,QP (Q 0) (Q 10),QP (Q 0) (Q 11),QP (Q 0) Trash,QP (Q 1) (Q 10),QP (Q 1)␣
 ↪(Q 11),QP (Q 1) Trash,QP Trash (Q 10),QP Trash (Q 11),QP Trash Trash]
QP (Q 0) (Q 10)
[(Trash,'a',Trash),(Trash,'b',Trash),(QP (Q 0) (Q 10),'a',QP Trash (Q 11)),(QP␣
 ↪(Q 0) (Q 10),'b',QP (Q 1) Trash),(QP (Q 0) (Q 11),'a',QP Trash Trash),(QP (Q␣
 ↪0) (Q 11),'b',QP (Q 1) (Q 10)),(QP (Q 0) Trash,'a',QP Trash Trash),(QP (Q 0)␣
 ↪Trash,'b',QP (Q 1) Trash),(QP (Q 1) (Q 10),'a',QP (Q 1) (Q 11)),(QP (Q 1) (Q␣
 ↪10),'b',QP (Q 1) Trash),(QP (Q 1) (Q 11),'a',QP (Q 1) Trash),(QP (Q 1) (Q␣
 ↪11),'b',QP (Q 1) (Q 10)),(QP (Q 1) Trash,'a',QP (Q 1) Trash),(QP (Q 1)␣
 ↪Trash,'b',QP (Q 1) Trash),(QP Trash (Q 10),'a',QP Trash (Q 11)),(QP Trash (Q␣
 ↪10),'b',QP Trash Trash),(QP Trash (Q 11),'a',QP Trash Trash),(QP Trash (Q␣
 ↪11),'b',QP Trash (Q 10)),(QP Trash Trash,'a',QP Trash Trash),(QP Trash␣
 ↪Trash,'b',QP Trash Trash)]
[QP (Q 0) (Q 10),QP (Q 1) (Q 10),QP (Q 1) (Q 11),QP (Q 1) Trash,QP Trash (Q 10)]
```

[22]:
```
atEmpiezaBoesAB' = normalizeNodes $ reduce atEmpiezaBoesAB
atEmpiezaBoesAB'
```

```
ab
[Q 0,Q 1,Q 2,Q 3,Q 4,Q 5]
Q 5
```

```
[(Q 0,'a',Q 0),(Q 0,'b',Q 0),(Q 1,'a',Q 4),(Q 1,'b',Q 2),(Q 2,'a',Q 2),(Q␣
↪2,'b',Q 2),(Q 3,'a',Q 3),(Q 3,'b',Q 3),(Q 4,'a',Q 2),(Q 4,'b',Q 1),(Q 5,'a',Q␣
↪4),(Q 5,'b',Q 3)]
[Q 1,Q 3,Q 5]
```

[23]:
```
isRenewed atEmpiezaBoesAB' "abababababab"
isRenewed atEmpiezaBoesAB' "abba"
isRenewed atEmpiezaBoesAB' "babbb"
```

```
True
```

```
False
```

```
True
```

Intersección autómatas (andAFD :: AFD -> AFD -> AFD)

[24]:
```
atEmpiezaAyB = andAFD atEmpiezaA atEmpiezaB

isRenewed atEmpiezaAyB "abaabbbb"
isRenewed atEmpiezaAyB "babaaa"

deltaB atEmpiezaAyB "aa" [QP (Q 0) (Q 0)]
```

```
False
```

```
False
```

```
[QP (Q 1) Trash]
```

[25]:
```
ataParyEmpiezaA = andAFD atApar atEmpiezaA

isRenewed ataParyEmpiezaA "babaaa"
isRenewed ataParyEmpiezaA "abaabbabb"
isRenewed ataParyEmpiezaA "abaabbbb"
```

```
False
```

```
True
```

```
False
```

Autómata complementario (complementaryAFD :: AFD -> AFD)

[26]:
```
atnoEmpiezaA = complementaryAFD atEmpiezaA

isRenewed atnoEmpiezaA "aabb"
isRenewed atnoEmpiezaA "bab"
```

```
False
```

```
True
```

Autómata diferencia (minusAFD :: AFD -> AFD -> AFD)

[27]:
```
atAparPeroNoEmpiezaA = minusAFD atApar atEmpiezaA


isRenewed atAparPeroNoEmpiezaA "aabb"
isRenewed atAparPeroNoEmpiezaA "bab"
isRenewed atAparPeroNoEmpiezaA "baba"
```

```
False
```

```
False
```

```
True
```

2 Autómata finito no determinista

$$L_{atAnt} := \{w_1 a w_2 : w_1, w_2 \in \Sigma^*, |w_2| = 2\}$$

[28]:
```
deltae :: Char -> Status -> [Status]
deltae 'a' (Q 0) = [Q 0, Q 1]
deltae _ (Q 0) = [Q 0]

deltae _ (Q 1) = [Q 2]
deltae _ (Q 2) = [Q 3]

deltae _ _ = []

afnAnt = AFN "ab" [Q 0, Q 1, Q 2, Q 3] (Q 0) deltae [Q 3]
```

Los AFN son instancias de Automata y comparten con AFD las funciones isRenewed y deltaB.

[29]:
```
isRenewed afnAnt "abbabb"
isRenewed afnAnt "abbab"
```

```
True
```

```
False
```

[30]:
```
deltaB afnAnt "abbabb" [Q 0]
```

```
[Q 0,Q 3]
```

2.0.1 Convertir a AFD y reducción

[31]:
```
afdAnt' = afnToafd afnAnt
afdAnt'
```

```
ab
[Trash,QT [Q 0,Q 1,Q 2,Q 3],QT [Q 0,Q 1,Q 2],QT [Q 0,Q 1,Q 3],QT [Q 0,Q 1],QT [Q␣
↪0,Q 2,Q 3],QT [Q 0,Q 2],QT [Q 0,Q 3],QT [Q 0],QT [Q 1,Q 2,Q 3],QT [Q 1,Q 2],QT␣
↪[Q 1,Q 3],QT [Q 1],QT [Q 2,Q 3],QT [Q 2],QT [Q 3],QT []]
QT [Q 0]
```

```
[(Trash,'a',Trash),(Trash,'b',Trash),(QT [Q 0,Q 1,Q 2,Q 3],'a',QT [Q 0,Q 1,Q 2,Q␣
 ↪3]),(QT [Q 0,Q 1,Q 2,Q 3],'b',QT [Q 0,Q 2,Q 3]),(QT [Q 0,Q 1,Q 2],'a',QT [Q␣
 ↪0,Q 1,Q 2,Q 3]),(QT [Q 0,Q 1,Q 2],'b',QT [Q 0,Q 2,Q 3]),(QT [Q 0,Q 1,Q␣
 ↪3],'a',QT [Q 0,Q 1,Q 2]),(QT [Q 0,Q 1,Q 3],'b',QT [Q 0,Q 2]),(QT [Q 0,Q␣
 ↪1],'a',QT [Q 0,Q 1,Q 2]),(QT [Q 0,Q 1],'b',QT [Q 0,Q 2]),(QT [Q 0,Q 2,Q␣
 ↪3],'a',QT [Q 0,Q 1,Q 3]),(QT [Q 0,Q 2,Q 3],'b',QT [Q 0,Q 3]),(QT [Q 0,Q␣
 ↪2],'a',QT [Q 0,Q 1,Q 3]),(QT [Q 0,Q 2],'b',QT [Q 0,Q 3]),(QT [Q 0,Q 3],'a',QT␣
 ↪[Q 0,Q 1]),(QT [Q 0,Q 3],'b',QT [Q 0]),(QT [Q 0],'a',QT [Q 0,Q 1]),(QT [Q␣
 ↪0],'b',QT [Q 0]),(QT [Q 1,Q 2,Q 3],'a',QT [Q 2,Q 3]),(QT [Q 1,Q 2,Q 3],'b',QT␣
 ↪[Q 2,Q 3]),(QT [Q 1,Q 2],'a',QT [Q 2,Q 3]),(QT [Q 1,Q 2],'b',QT [Q 2,Q 3]),(QT␣
 ↪[Q 1,Q 3],'a',QT [Q 2]),(QT [Q 1,Q 3],'b',QT [Q 2]),(QT [Q 1],'a',QT [Q␣
 ↪2]),(QT [Q 1],'b',QT [Q 2]),(QT [Q 2,Q 3],'a',QT [Q 3]),(QT [Q 2,Q 3],'b',QT␣
 ↪[Q 3]),(QT [Q 2],'a',QT [Q 3]),(QT [Q 2],'b',QT [Q 3]),(QT [Q 3],'a',QT␣
 ↪[]),(QT [Q 3],'b',QT []),(QT [],'a',QT []),(QT [],'b',QT [])]
[QT [Q 0,Q 1,Q 2,Q 3],QT [Q 0,Q 1,Q 3],QT [Q 0,Q 2,Q 3],QT [Q 0,Q 3],QT [Q 1,Q␣
 ↪2,Q 3],QT [Q 1,Q 3],QT [Q 2,Q 3],QT [Q 3]]
```

[32]:
```
afdAnt = normalizeNodes $ reduce afdAnt'
afdAnt
```

```
ab
[Q 0,Q 1,Q 2,Q 3,Q 4,Q 5,Q 6,Q 7,Q 8]
Q 8
[(Q 0,'a',Q 0),(Q 0,'b',Q 0),(Q 1,'a',Q 7),(Q 1,'b',Q 8),(Q 2,'a',Q 6),(Q␣
 ↪2,'b',Q 5),(Q 3,'a',Q 2),(Q 3,'b',Q 1),(Q 4,'a',Q 4),(Q 4,'b',Q 3),(Q 5,'a',Q␣
 ↪2),(Q 5,'b',Q 1),(Q 6,'a',Q 4),(Q 6,'b',Q 3),(Q 7,'a',Q 6),(Q 7,'b',Q 5),(Q␣
 ↪8,'a',Q 7),(Q 8,'b',Q 8)]
[Q 1,Q 2,Q 3,Q 4]
```

[33]:
```
isRenewed afdAnt' "abbabb"
isRenewed afdAnt' "abbab"
```

```
True
```

```
False
```

[34]:
```
isRenewed afdAnt "abbabb"
isRenewed afdAnt "abbab"
```

```
True
```

```
False
```

3 Autómata finito no determinista con transicciones libres

$$L_{EmpiezaAoTerminaB} := \{aw : w \in \Sigma^*\} \cup \{wb : w \in \Sigma^*\} = a(a+b)* + (a+b)*b$$

[35]:
```
deltax :: Char -> Status -> [Status]
deltax 'a' (Q 0) = [Q 1]
```

```
deltax 'a' (Q 1) = [Q 1]
deltax 'b' (Q 1) = [Q 1]
deltax 'a' (Q 2) = [Q 2]
deltax 'b' (Q 2) = [Q 2, Q 3]
deltax _ _ = []


epsilon :: Status -> [Status]
epsilon (Q 100) = [Q 0, Q 2]
epsilon _ = []

afneAB = AFNe "ab" [Q 100, Q 0, Q 1, Q 2, Q 3] (Q 100) deltax [Q 1, Q 3] epsilon
```

[36]:
```
afneAB
```

```
ab
[Q 100,Q 0,Q 1,Q 2,Q 3]
Q 100
[(Q 100,'a',[]),(Q 100,'b',[]),(Q 0,'a',[Q 1]),(Q 0,'b',[]),(Q 1,'a',[Q 1]),(Q␣
 ↪1,'b',[Q 1]),(Q 2,'a',[Q 2]),(Q 2,'b',[Q 2,Q 3]),(Q 3,'a',[]),(Q 3,'b',[])]
[Q 1,Q 3]
[(Q 100,[Q 0,Q 2]),(Q 0,[]),(Q 1,[]),(Q 2,[]),(Q 3,[])]
```

3.0.1 Reconocimiento de cadenas

[37]:
```
isRenewed afneAB "abbba"
isRenewed afneAB "bbbab"
isRenewed afneAB "baaba"
```

```
True

True

False
```

3.0.2 Conversión a AFN y AFD

[38]:
```
afnAB = afneToafn afneAB
afdAB = afnToafd afnAB
afdAB
```

```
ab
[Trash,QT [Q 100,Q 0,Q 1,Q 2,Q 3],QT [Q 100,Q 0,Q 1,Q 2],QT [Q 100,Q 0,Q 1,Q␣
 ↪3],QT [Q 100,Q 0,Q 1],QT [Q 100,Q 0,Q 2,Q 3],QT [Q 100,Q 0,Q 2],QT [Q 100,Q␣
 ↪0,Q 3],QT [Q 100,Q 0],QT [Q 100,Q 1,Q 2,Q 3],QT [Q 100,Q 1,Q 2],QT [Q 100,Q␣
 ↪1,Q 3],QT [Q 100,Q 1],QT [Q 100,Q 2,Q 3],QT [Q 100,Q 2],QT [Q 100,Q 3],QT [Q␣
 ↪100],QT [Q 0,Q 1,Q 2,Q 3],QT [Q 0,Q 1,Q 2],QT [Q 0,Q 1,Q 3],QT [Q 0,Q 1],QT [Q␣
 ↪0,Q 2,Q 3],QT [Q 0,Q 2],QT [Q 0,Q 3],QT [Q 0],QT [Q 1,Q 2,Q 3],QT [Q 1,Q 2],QT␣
 ↪[Q 1,Q 3],QT [Q 1],QT [Q 2,Q 3],QT [Q 2],QT [Q 3],QT []]
QT [Q 100]
```

```
[(Trash,'a',Trash),(Trash,'b',Trash),(QT [Q 100,Q 0,Q 1,Q 2,Q 3],'a',QT [Q 1,Q␣
 ↪2,Q 1,Q 1,Q 2]),(QT [Q 100,Q 0,Q 1,Q 2,Q 3],'b',QT [Q 2,Q 3,Q 1,Q 2,Q 3]),(QT␣
 ↪[Q 100,Q 0,Q 1,Q 2],'a',QT [Q 1,Q 2,Q 1,Q 1,Q 2]),(QT [Q 100,Q 0,Q 1,Q␣
 ↪2],'b',QT [Q 2,Q 3,Q 1,Q 2,Q 3]),(QT [Q 100,Q 0,Q 1,Q 3],'a',QT [Q 1,Q 2,Q 1,Q␣
 ↪1]),(QT [Q 100,Q 0,Q 1,Q 3],'b',QT [Q 2,Q 3,Q 1]),(QT [Q 100,Q 0,Q 1],'a',QT␣
 ↪[Q 1,Q 2,Q 1,Q 1]),(QT [Q 100,Q 0,Q 1],'b',QT [Q 2,Q 3,Q 1]),(QT [Q 100,Q 0,Q␣
 ↪2,Q 3],'a',QT [Q 1,Q 2,Q 1,Q 2]),(QT [Q 100,Q 0,Q 2,Q 3],'b',QT [Q 2,Q 3,Q 2,Q␣
 ↪3]),(QT [Q 100,Q 0,Q 2],'a',QT [Q 1,Q 2,Q 1,Q 2]),(QT [Q 100,Q 0,Q 2],'b',QT␣
 ↪[Q 2,Q 3,Q 2,Q 3]),(QT [Q 100,Q 0,Q 3],'a',QT [Q 1,Q 2,Q 1]),(QT [Q 100,Q 0,Q␣
 ↪3],'b',QT [Q 2,Q 3]),(QT [Q 100,Q 0],'a',QT [Q 1,Q 2,Q 1]),(QT [Q 100,Q␣
 ↪0],'b',QT [Q 2,Q 3]),(QT [Q 100,Q 1,Q 2,Q 3],'a',QT [Q 1,Q 2,Q 1,Q 2]),(QT [Q␣
 ↪100,Q 1,Q 2,Q 3],'b',QT [Q 2,Q 3,Q 1,Q 2,Q 3]),(QT [Q 100,Q 1,Q 2],'a',QT [Q␣
 ↪1,Q 2,Q 1,Q 2]),(QT [Q 100,Q 1,Q 2],'b',QT [Q 2,Q 3,Q 1,Q 2,Q 3]),(QT [Q 100,Q␣
 ↪1,Q 3],'a',QT [Q 1,Q 2,Q 1]),(QT [Q 100,Q 1,Q 3],'b',QT [Q 2,Q 3,Q 1]),(QT [Q␣
 ↪100,Q 1],'a',QT [Q 1,Q 2,Q 1]),(QT [Q 100,Q 1],'b',QT [Q 2,Q 3,Q 1]),(QT [Q␣
 ↪100,Q 2,Q 3],'a',QT [Q 1,Q 2,Q 2]),(QT [Q 100,Q 2,Q 3],'b',QT [Q 2,Q 3,Q 2,Q␣
 ↪3]),(QT [Q 100,Q 2],'a',QT [Q 1,Q 2,Q 2]),(QT [Q 100,Q 2],'b',QT [Q 2,Q 3,Q␣
 ↪2,Q 3]),(QT [Q 100,Q 3],'a',QT [Q 1,Q 2]),(QT [Q 100,Q 3],'b',QT [Q 2,Q␣
 ↪3]),(QT [Q 100],'a',QT [Q 1,Q 2]),(QT [Q 100],'b',QT [Q 2,Q 3]),(QT [Q 0,Q 1,Q␣
 ↪2,Q 3],'a',QT [Q 1,Q 1,Q 2]),(QT [Q 0,Q 1,Q 2,Q 3],'b',QT [Q 1,Q 2,Q 3]),(QT␣
 ↪[Q 0,Q 1,Q 2],'a',QT [Q 1,Q 1,Q 2]),(QT [Q 0,Q 1,Q 2],'b',QT [Q 1,Q 2,Q␣
 ↪3]),(QT [Q 0,Q 1,Q 3],'a',QT [Q 1,Q 1]),(QT [Q 0,Q 1,Q 3],'b',QT [Q 1]),(QT [Q␣
 ↪0,Q 1],'a',QT [Q 1,Q 1]),(QT [Q 0,Q 1],'b',QT [Q 1]),(QT [Q 0,Q 2,Q 3],'a',QT␣
 ↪[Q 1,Q 2]),(QT [Q 0,Q 2,Q 3],'b',QT [Q 2,Q 3]),(QT [Q 0,Q 2],'a',QT [Q 1,Q␣
 ↪2]),(QT [Q 0,Q 2],'b',QT [Q 2,Q 3]),(QT [Q 0,Q 3],'a',QT [Q 1]),(QT [Q 0,Q␣
 ↪3],'b',QT []),(QT [Q 0],'a',QT [Q 1]),(QT [Q 0],'b',QT []),(QT [Q 1,Q 2,Q␣
 ↪3],'a',QT [Q 1,Q 2]),(QT [Q 1,Q 2,Q 3],'b',QT [Q 1,Q 2,Q 3]),(QT [Q 1,Q␣
 ↪2],'a',QT [Q 1,Q 2]),(QT [Q 1,Q 2],'b',QT [Q 1,Q 2,Q 3]),(QT [Q 1,Q 3],'a',QT␣
 ↪[Q 1]),(QT [Q 1,Q 3],'b',QT [Q 1]),(QT [Q 1],'a',QT [Q 1]),(QT [Q 1],'b',QT [Q␣
 ↪1]),(QT [Q 2,Q 3],'a',QT [Q 2]),(QT [Q 2,Q 3],'b',QT [Q 2,Q 3]),(QT [Q␣
 ↪2],'a',QT [Q 2]),(QT [Q 2],'b',QT [Q 2,Q 3]),(QT [Q 3],'a',QT []),(QT [Q␣
 ↪3],'b',QT []),(QT [],'a',QT []),(QT [],'b',QT [])]
[QT [Q 100,Q 0,Q 1,Q 2,Q 3],QT [Q 100,Q 0,Q 1,Q 2],QT [Q 100,Q 0,Q 1,Q 3],QT [Q␣
 ↪100,Q 0,Q 1],QT [Q 100,Q 0,Q 2,Q 3],QT [Q 100,Q 0,Q 3],QT [Q 100,Q 1,Q 2,Q␣
 ↪3],QT [Q 100,Q 1,Q 2],QT [Q 100,Q 1,Q 3],QT [Q 100,Q 1],QT [Q 100,Q 2,Q 3],QT␣
 ↪[Q 100,Q 3],QT [Q 0,Q 1,Q 2,Q 3],QT [Q 0,Q 1,Q 2],QT [Q 0,Q 1,Q 3],QT [Q 0,Q␣
 ↪1],QT [Q 0,Q 2,Q 3],QT [Q 0,Q 3],QT [Q 1,Q 2,Q 3],QT [Q 1,Q 2],QT [Q 1,Q 3],QT␣
 ↪[Q 1],QT [Q 2,Q 3],QT [Q 3]]
```

[39]:
```
afdAB' = normalizeNodes $ reduce afdAB
afdAB'
```

```
ab
[Q 0,Q 1,Q 2,Q 3,Q 4,Q 5]
Q 5
[(Q 0,'a',Q 0),(Q 0,'b',Q 0),(Q 1,'a',Q 1),(Q 1,'b',Q 3),(Q 2,'a',Q 4),(Q␣
 ↪2,'b',Q 2),(Q 3,'a',Q 1),(Q 3,'b',Q 3),(Q 4,'a',Q 4),(Q 4,'b',Q 2),(Q 5,'a',Q␣
 ↪4),(Q 5,'b',Q 3)]
```

```
[Q 2,Q 3,Q 4]
```

[40]:
```
isRenewed afdAB' "abbba"
isRenewed afdAB' "bbbab"
isRenewed afdAB' "baaba"
```

```
True
```

```
True
```

```
False
```

4 Expresiones regulares

[41]:
```
a = RexChar 'a'
b = RexChar 'b'
c = RexChar 'c'
```

4.0.1 Concatenación de expresiones regulares

[42]:
```
ab = a|++|b
abc = a|++|b|++|c
```

[43]:
```
ab
```

```
RexConcat (RexChar 'a') (RexChar 'b')
```

4.0.2 Suma de expresiones regulares

[44]:
```
aOb = a|+|b
aOc = a|+|c
```

[45]:
```
aOb
```

```
RexSum (RexChar 'a') (RexChar 'b')
```

Cierre de una expresión regular

[46]:
```
a' = (|^|) a
a'
```

```
RexClosing (RexChar 'a')
```

$$L_{er} := c(a)^*(ab)^*cb$$

[47]:
```
ab' = (|^|) ab
er = c |++| a' |++| ab' |++| c |++| b
```

[48]:
```
afneraro = regToAFNe er
isRenewed afneraro "abacba"
```

```
isRenewed afneraro "caaaababababababcb"
isRenewed afneraro "ccb"
isRenewed afneraro "cabcb"
isRenewed afneraro "acabcb"
```

```
False

True

True

True

False
```

[49]:
```
afneraro
```

```
cab
[QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 1) (Q 0)))),QP (Q 1) (QP (Q 1) (QP (Q 1)␣
 ↪(QP (Q 1) (Q 1)))),QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 0)))),QP (Q 1)␣
 ↪(QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 1)))),QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2)␣
 ↪(QP (Q 1) (Q 0))))),QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (Q␣
 ↪1))))),QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 0))),QP (Q 1) (QP (Q 1) (QP (Q 2) (Q␣
 ↪1))),QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 1) (Q 0))))),QP (Q 1) (QP␣
 ↪(Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 1) (Q 1))))),QP (Q 1) (QP (Q 1) (QP (Q 2) (QP␣
 ↪(Q 1) (QP (Q 2) (Q 0))))),QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 2) (Q␣
 ↪1))))),QP (Q 1) (QP (Q 2) (Q 0)),QP (Q 1) (QP (Q 2) (Q 1)),QP (Q 2) (Q 0),QP␣
 ↪(Q 2) (Q 1)]
QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 1) (Q 0))))
```

```
[(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 1) (Q 0)))),'c',[QP (Q 1) (QP (Q 1) (QP (Q␣
↪1) (QP (Q 1) (Q 1)))]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 1) (Q␣
↪0)))),'a',[]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 1) (Q 0)))),'b',[]),(QP (Q␣
↪1) (QP (Q 1) (QP (Q 1) (QP (Q 1) (Q 1)))),'c',[]),(QP (Q 1) (QP (Q 1) (QP (Q␣
↪1) (QP (Q 1) (Q 1)))),'a',[]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 1) (Q␣
↪1)))),'b',[]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 0)))),'c',[]),(QP (Q␣
↪1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 0)))),'a',[]),(QP (Q 1) (QP (Q 1) (QP (Q␣
↪1) (QP (Q 2) (Q 0)))),'b',[]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (Q␣
↪1)))),'c',[]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 1)))),'a',[]),(QP (Q␣
↪1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 1)))),'b',[]),(QP (Q 1) (QP (Q 1) (QP (Q␣
↪1) (QP (Q 2) (QP (Q 1) (Q 0))))),'c',[]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q␣
↪2) (QP (Q 1) (Q 0))))),'a',[QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1)␣
↪(Q 1)))))]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (Q␣
↪0))))),'b',[]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (Q␣
↪1))))),'c',[]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (Q␣
↪1))))),'a',[]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (Q␣
↪1))))),'b',[]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 0))),'c',[]),(QP (Q 1) (QP (Q␣
↪1) (QP (Q 2) (Q 0))),'a',[]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 0))),'b',[]),(QP␣
↪(Q 1) (QP (Q 1) (QP (Q 2) (Q 1))),'c',[]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (Q␣
↪1))),'a',[]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 1))),'b',[]),(QP (Q 1) (QP (Q 1)␣
↪(QP (Q 2) (QP (Q 1) (QP (Q 1) (Q 0))))),'c',[]),(QP (Q 1) (QP (Q 1) (QP (Q 2)␣
↪(QP (Q 1) (QP (Q 1) (Q 0))))),'a',[QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP␣
↪(Q 1) (Q 1)))))]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 1) (Q␣
↪0))))),'b',[]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 1) (Q␣
↪1))))),'c',[]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 1) (Q␣
↪1))))),'a',[]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 1) (Q␣
↪1))))),'b',[]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 2) (Q␣
↪0))))),'c',[]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 2) (Q␣
↪0))))),'a',[]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 2) (Q␣
↪0))))),'b',[QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 2) (Q 1)))))]),(QP␣
↪(Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 2) (Q 1))))),'c',[]),(QP (Q 1) (QP␣
↪(Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 2) (Q 1))))),'a',[]),(QP (Q 1) (QP (Q 1) (QP␣
↪(Q 2) (QP (Q 1) (QP (Q 2) (Q 1))))),'b',[]),(QP (Q 1) (QP (Q 2) (Q 0)),'c',[QP␣
↪(Q 1) (QP (Q 2) (Q 1))]),(QP (Q 1) (QP (Q 2) (Q 0)),'a',[]),(QP (Q 1) (QP (Q␣
↪2) (Q 0)),'b',[]),(QP (Q 1) (QP (Q 2) (Q 1)),'c',[]),(QP (Q 1) (QP (Q 2) (Q␣
↪1)),'a',[]),(QP (Q 1) (QP (Q 2) (Q 1)),'b',[]),(QP (Q 2) (Q 0),'c',[]),(QP (Q␣
↪2) (Q 0),'a',[]),(QP (Q 2) (Q 0),'b',[QP (Q 2) (Q 1)]),(QP (Q 2) (Q␣
↪1),'c',[]),(QP (Q 2) (Q 1),'a',[]),(QP (Q 2) (Q 1),'b',[])]
[QP (Q 2) (Q 1)]
```

```
[(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 1) (Q 0)))),[]),(QP (Q 1) (QP (Q 1) (QP (Q␣
↪1) (QP (Q 1) (Q 1)))),[QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 0))))]),(QP␣
↪(Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 0)))),[QP (Q 1) (QP (Q 1) (QP (Q 1) (QP␣
↪(Q 2) (QP (Q 1) (Q 0))))),QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (Q␣
↪1))))]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 1)))),[QP (Q 1) (QP (Q 1)␣
↪(QP (Q 2) (Q 0)))]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (Q␣
↪0))))),[]),(QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (Q 1))))),[QP (Q␣
↪1) (QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 1)))),QP (Q 1) (QP (Q 1) (QP (Q 1) (QP (Q␣
↪2) (QP (Q 1) (Q 0))))]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 0))),[QP (Q 1) (QP␣
↪(Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 1) (Q 0))))),QP (Q 1) (QP (Q 1) (QP (Q 2) (Q␣
↪1)))]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (Q 1))),[QP (Q 1) (QP (Q 2) (Q 0))]),(QP␣
↪(Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 1) (Q 0))))),[]),(QP (Q 1) (QP (Q␣
↪1) (QP (Q 2) (QP (Q 1) (QP (Q 1) (Q 1))))),[QP (Q 1) (QP (Q 1) (QP (Q 2) (QP␣
↪(Q 1) (QP (Q 2) (Q 0))))]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 2)␣
↪(Q 0))))),[]),(QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP (Q 2) (Q 1))))),[QP␣
↪(Q 1) (QP (Q 1) (QP (Q 2) (Q 1))),QP (Q 1) (QP (Q 1) (QP (Q 2) (QP (Q 1) (QP␣
↪(Q 1) (Q 0))))]),(QP (Q 1) (QP (Q 2) (Q 0)),[]),(QP (Q 1) (QP (Q 2) (Q␣
↪1)),[QP (Q 2) (Q 0)]),(QP (Q 2) (Q 0),[]),(QP (Q 2) (Q 1),[])]
```

4.0.3 Conversión a AFD

[50]:
```
afnraro = afneToafn afneraro
afdraro = afnToafd afnraro
```

[51]:
```
afdraro' = normalizeNodes $ reduce afdraro
afdraro'
```

```
cab
[Q 0,Q 1,Q 2,Q 3,Q 4,Q 5,Q 6,Q 7,Q 8]
Q 8
[(Q 0,'c',Q 0),(Q 0,'a',Q 0),(Q 0,'b',Q 0),(Q 1,'c',Q 6),(Q 1,'a',Q 6),(Q␣
↪1,'b',Q 2),(Q 2,'c',Q 5),(Q 2,'a',Q 1),(Q 2,'b',Q 6),(Q 3,'c',Q 6),(Q 3,'a',Q␣
↪6),(Q 3,'b',Q 6),(Q 4,'c',Q 5),(Q 4,'a',Q 4),(Q 4,'b',Q 2),(Q 5,'c',Q 6),(Q␣
↪5,'a',Q 6),(Q 5,'b',Q 3),(Q 6,'c',Q 6),(Q 6,'a',Q 6),(Q 6,'b',Q 6),(Q 7,'c',Q␣
↪5),(Q 7,'a',Q 4),(Q 7,'b',Q 6),(Q 8,'c',Q 7),(Q 8,'a',Q 6),(Q 8,'b',Q 6)]
[Q 3]
```

[52]:
```
isRenewed afdraro' "abacba"
isRenewed afdraro' "caaaababababcb"
isRenewed afdraro' "ccb"
isRenewed afdraro' "cabcb"
isRenewed afdraro' "acabcb"
```

```
False

True

True

True
```

```
False
```

5 Otras funciones

renewedFile :: Automata a => a -> [Char] -> IO ()

[53]:
```
renewedFile afdraro' "palabras.txt"
```

```
[False,True,True,True,False]
```