Lecture 10: Implementing RegExp the hard way

Talen en Compilers 2023-2024, period 2

Lawrence Chonavel

Department of Information and Computing Sciences, Utrecht University

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Recap: RegExp

```
<|> :: R → R → R
<+> :: R → R → R
many  :: R → R
many1 :: R → R
option  :: R → R
symbol  :: Char → R
satisfy :: (Char → Bool) → R
type R = Parser Char String
```

Recap: RegExp

```
<|> :: R → R → R
<+> :: R → R → R
many  :: R → R
many1 :: R → R
option  :: R → R
symbol  :: Char → R
satisfy :: (Char → Bool) →
type R = Parser Char String
```

$r_1|r_2$

r_1r_2

r*

r+

r?

c

```
\d \s \S [a-z] ...
```

```
{}|{-?\d+(\.\d+)?}
|{(-?\d+(\.\d+)?,)+-?\d+(\.\d+)?}
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Recap: RegExp

```
<|> :: R → R → R
<+> :: R → R → R
many  :: R → R
many1 :: R → R
option  :: R → R
symbol  :: Char → R
satisfy :: (Char → Bool) →
type R = Parser Char String
```

$r_1|r_2$

r_1r_2

`r*`

`r+`

`r?`

`c`

```
\d \s \S [a-z] ...
```

```
{}|{-?\d+(\.\d+)?}
|{(-?\d+(\.\d+)?,)+-?\d+(\.\d+)?}
```

```
(0b)?(0|1)+
```

Recap: RegExp

```
<|> :: R → R → R
<+> :: R → R → R
many  :: R → R
many1 :: R → R
option  :: R → R
symbol  :: Char → R
satisfy :: (Char → Bool) →
type R = Parser Char String
```

```
r1|r2
r1r2
r*
r+
r?
c
\d \s \S [a-z] ...
```

```
{}|{-?\d+(\.\d+)?}
|{(-?\d+(\.\d+)?,)+-?\d+(\.\d+)?}
```

```
(0b)?(0|1)+
```

```
co(bra|d)
```

Recap: RegExp performance

```
head $ matchRegExp "a*aaaba*$" "aaaaaaabaa"

 aaa❌
a aaa❌
aa aaa❌
aaa aaa❌
aaaa aaab aa ✅
```

Today: matching RegExp fast 🚀

⏱️ Goal: `O(length input)` matching time

Today: matching RegExp fast 🚀

⏱️ Goal: `O(length input)` matching time

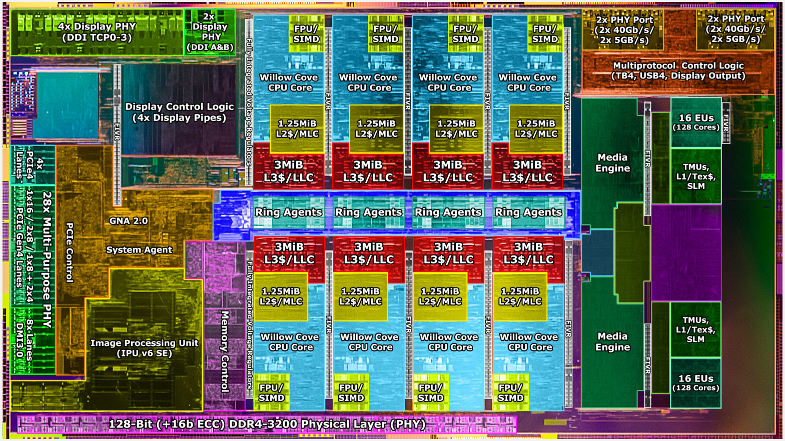
✨ New algorithm

Today: matching RegExp fast 🚀

⏱️ Goal: `O(length input)` matching time

✨ New algorithm from the bottom up

Problem: computers are complicated



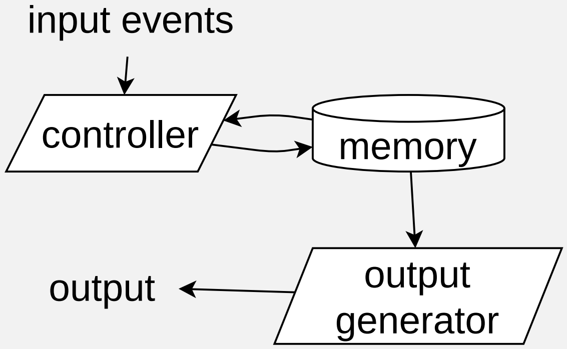
4x Display PHY (DDI TCP0-3)
2x Display PHY (DDI A&B)
FPU/SIMD
Willow Cove CPU Core
1.25MiB L2$/MLC
3MiB L3$/LLC
2x PHY Port (2x 40Gb/s/ 2x 5GB/s)
Multiprotocol Control Logic (TB4, USB4, Display Output)
Display Control Logic (4x Display Pipes)
Fully Integrated Voltage Regulators
FIVR
16 EUs (128 Cores)
TMUs, L1/Tex$, SLM
Media Engine
4x PCIe4 Lanes
28x Multi-Purpose PHY
1x16 / 2x8 / 1x8 + 2x4 PCIe Gen4 Lanes
8x Lanes DMI3.0
PCIe Control
GNA 2.0
System Agent
Ring Agents
Image Processing Unit (IPU v6 SE)
Memory Control
128-Bit (+16b ECC) DDR4-3200 Physical Layer (PHY)

10nm SF Tiger Lake 8Core processor (2021) die shot, by @Locuza_ on twitter

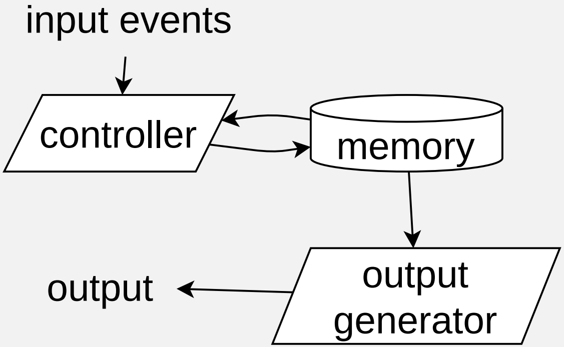
Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Simplification: Moore Machine

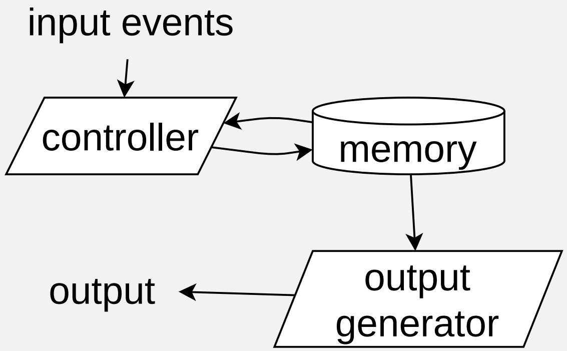


Simplification: Moore Machine



a.k.a. Finite State Machine (FSM)

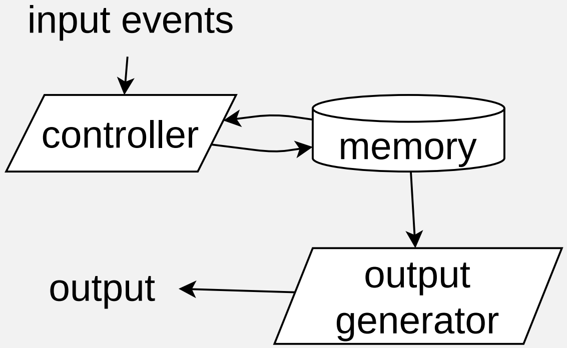
Simplification: Moore Machine



a.k.a. Finite State Machine (FSM)

v.s.t Finite State Automaton (FSA)

Simplification: Moore Machine

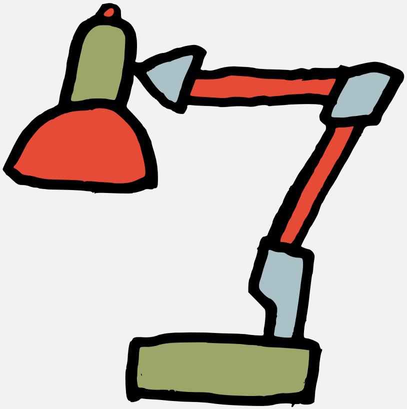


a.k.a. Finite State Machine (FSM)

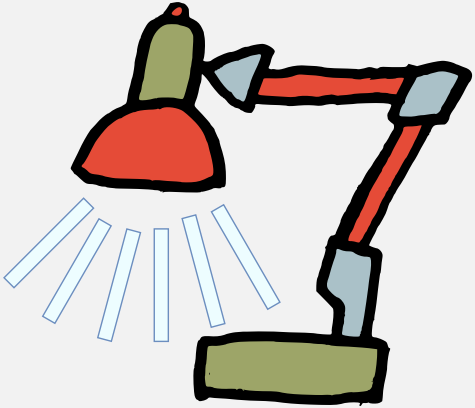
v.s.t Finite State Automaton (FSA)

a.k.a. Deterministic Finite Automaton (DFA)

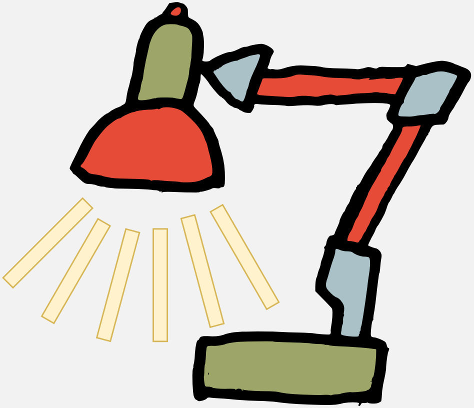
Concrete Example



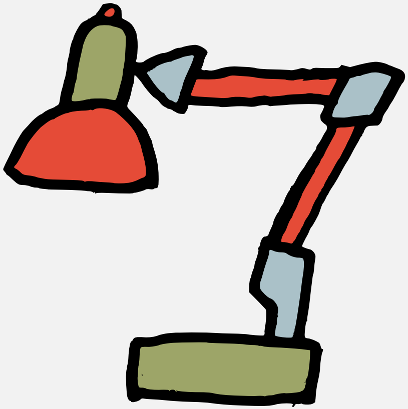
Concrete Example



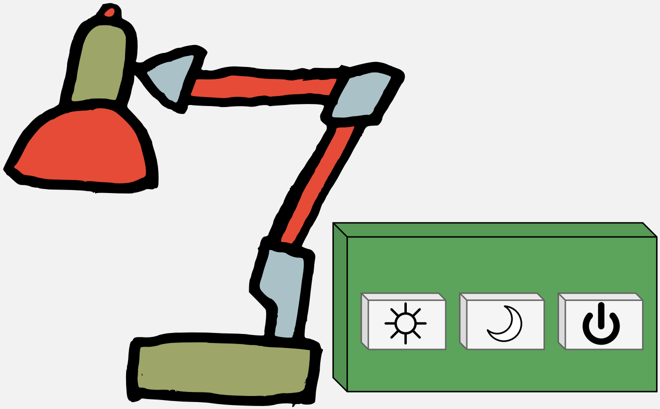
Concrete Example



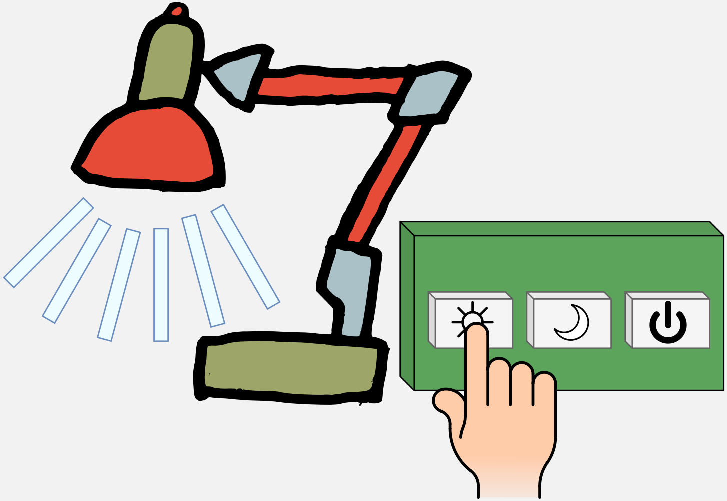
Concrete Example



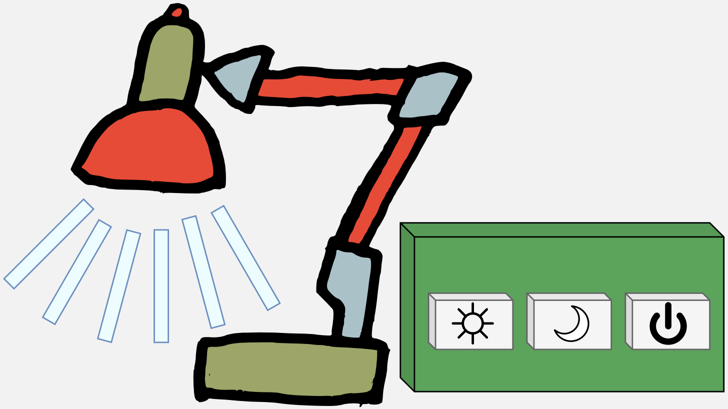
Concrete Example



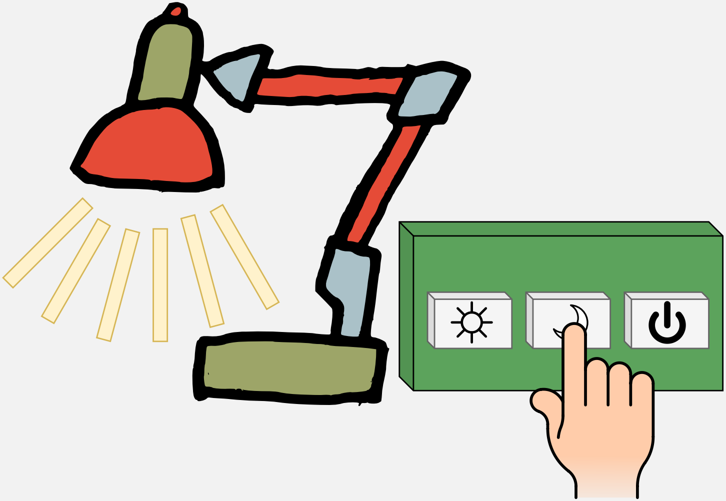
Concrete Example



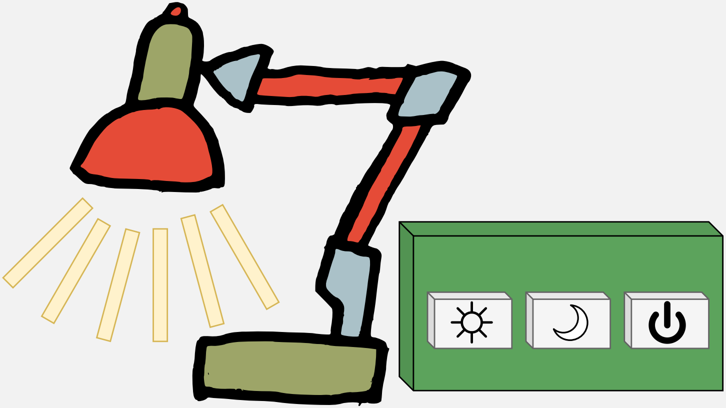
Concrete Example



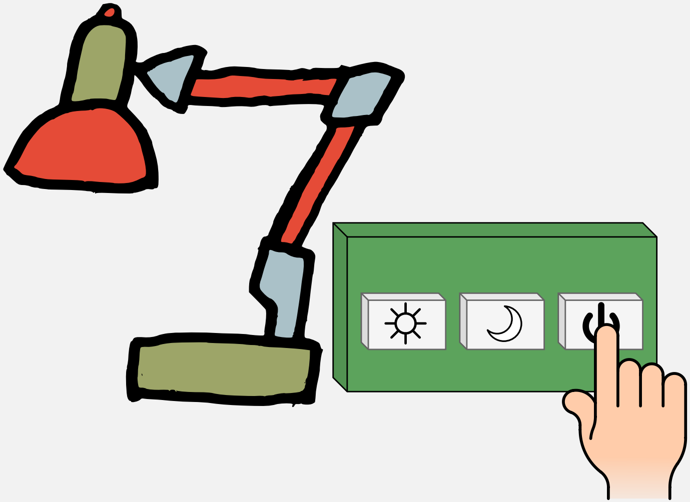
Concrete Example



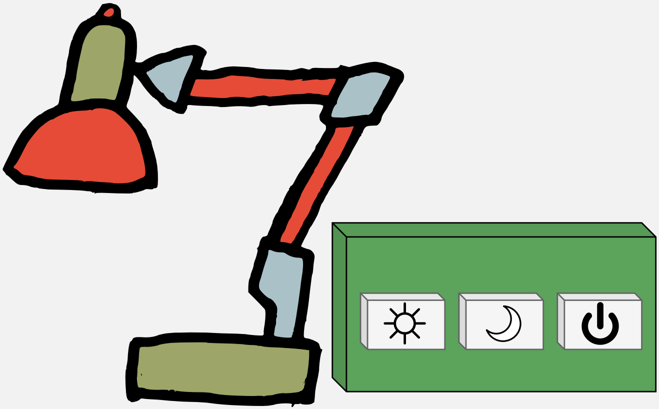
Concrete Example



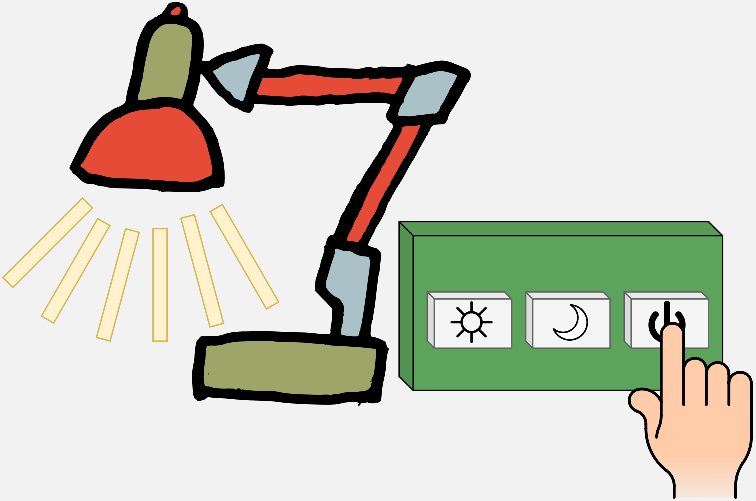
Concrete Example



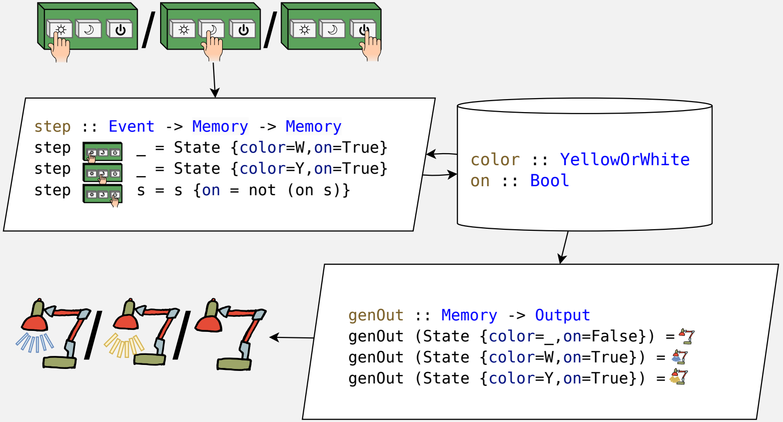
Concrete Example



Concrete Example



Moore Machine for lamp

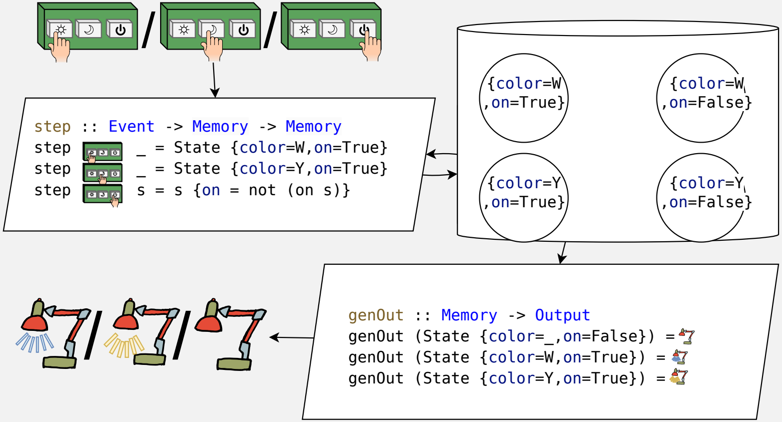


```
step :: Event -> Memory -> Memory
step       _ = State {color=W,on=True}
step       _ = State {color=Y,on=True}
step       s = s {on = not (on s)}
```

```
color :: YellowOrWhite
on :: Bool
```

```
genOut :: Memory -> Output
genOut (State {color=_,on=False}) =
genOut (State {color=W,on=True}) =
genOut (State {color=Y,on=True}) =
```

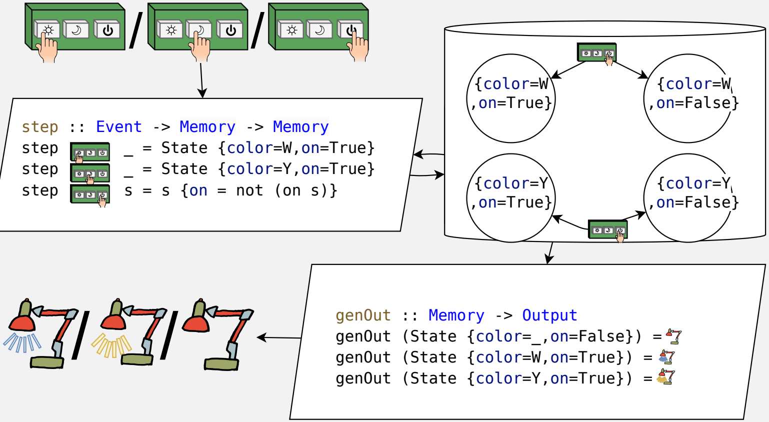
Moore Machine for lamp



```
step :: Event -> Memory -> Memory
step        _ = State {color=W,on=True}
step        _ = State {color=Y,on=True}
step        s = s {on = not (on s)}
```

```
genOut :: Memory -> Output
genOut (State {color=_,on=False}) =
genOut (State {color=W,on=True}) =
genOut (State {color=Y,on=True}) =
```

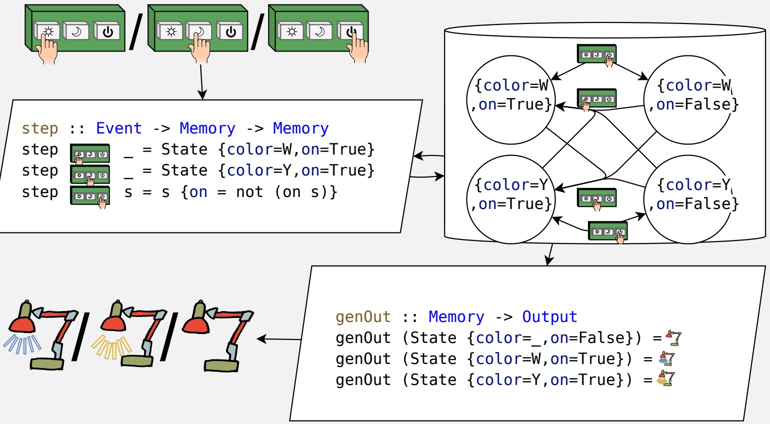
Moore Machine for lamp



```
step :: Event -> Memory -> Memory
step        _ = State {color=W,on=True}
step        _ = State {color=Y,on=True}
step        s = s {on = not (on s)}
```

```
genOut :: Memory -> Output
genOut (State {color=_,on=False}) =
genOut (State {color=W,on=True}) =
genOut (State {color=Y,on=True}) =
```

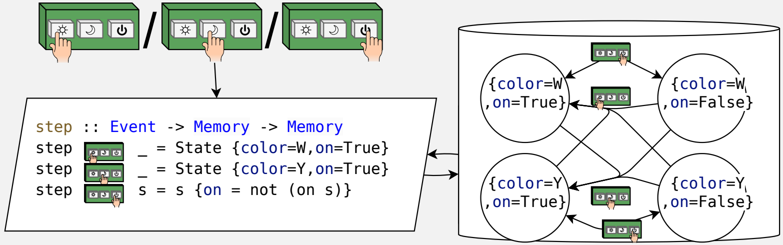
Moore Machine for lamp



```
step :: Event -> Memory -> Memory
step     _ = State {color=W,on=True}
step     _ = State {color=Y,on=True}
step     s = s {on = not (on s)}
```

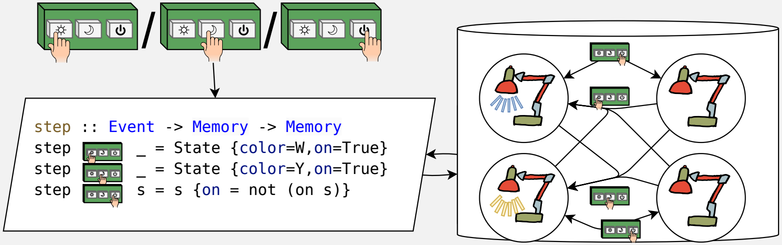
```
genOut :: Memory -> Output
genOut (State {color=_,on=False}) =
genOut (State {color=W,on=True}) =
genOut (State {color=Y,on=True}) =
```

Moore Machine for lamp



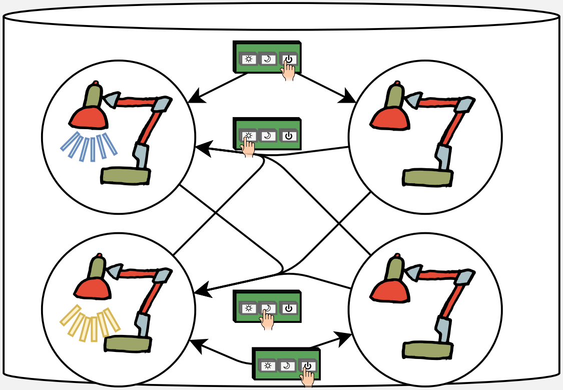
```
step :: Event -> Memory -> Memory
step       _ = State {color=W,on=True}
step       _ = State {color=Y,on=True}
step       s = s {on = not (on s)}
```

Moore Machine for lamp

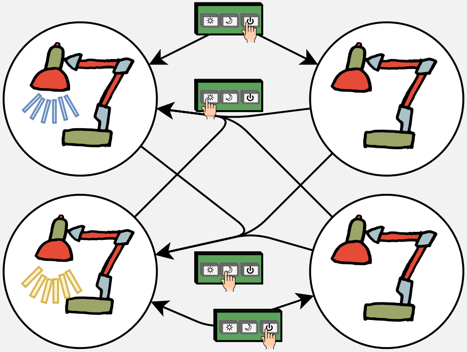


```
step :: Event -> Memory -> Memory
step      _ = State {color=W,on=True}
step      _ = State {color=Y,on=True}
step      s = s {on = not (on s)}
```

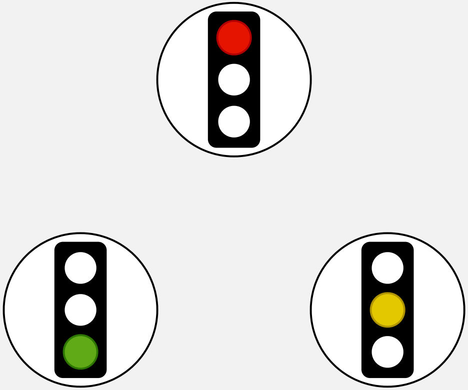
Moore Machine for lamp



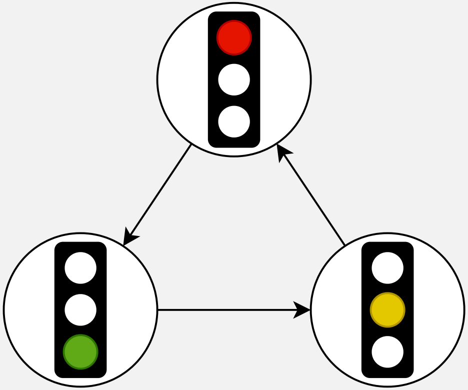
Moore Machine for lamp



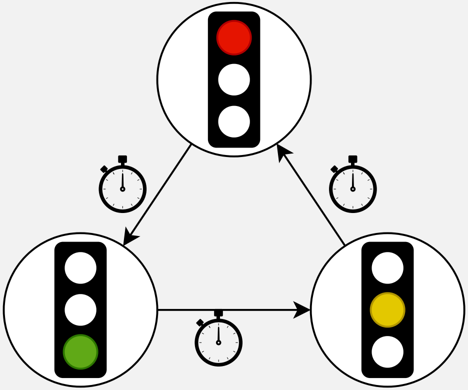
Another Example



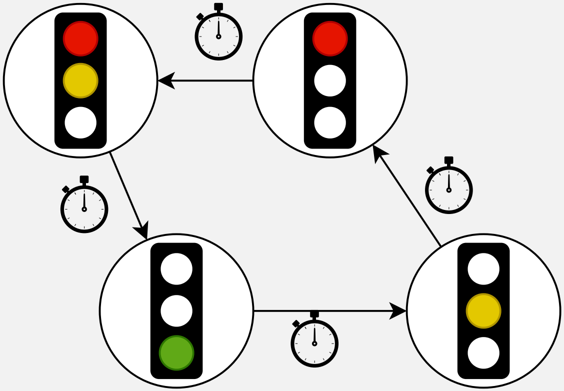
Another Example



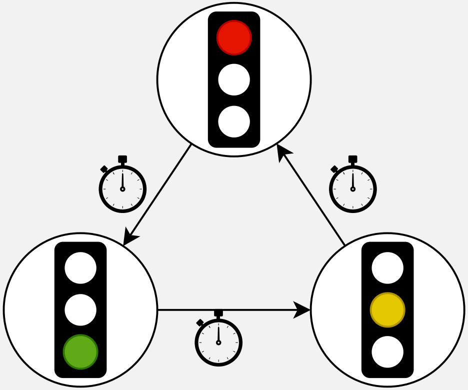
Another Example



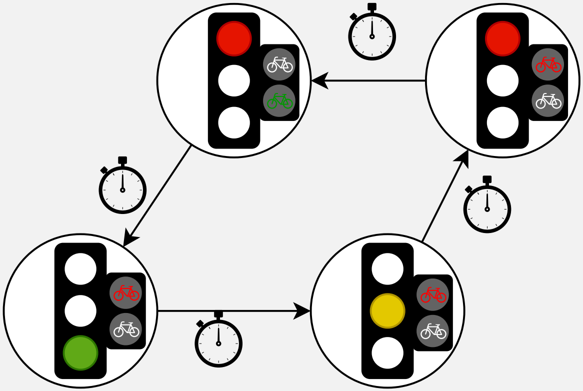
Another Example



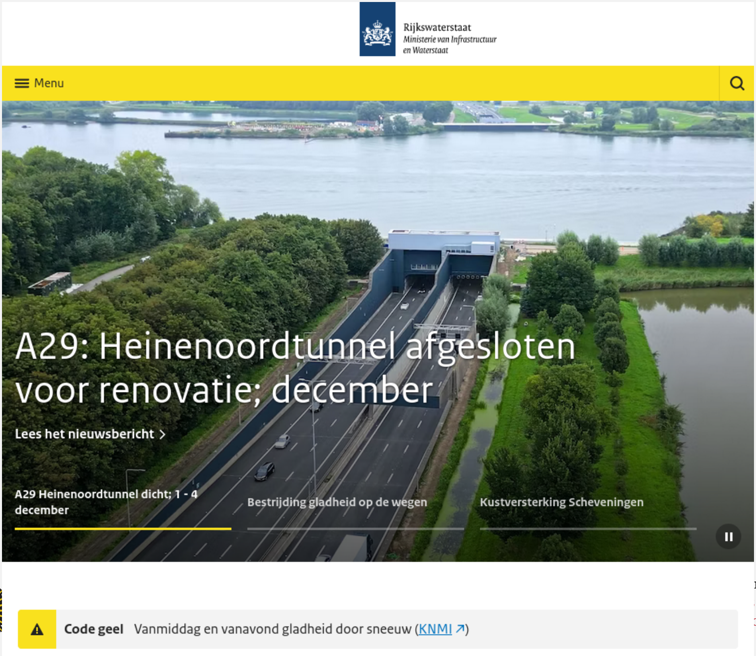
Another Example



Another Example



A Big Example


Rijkswaterstaat
Ministerie van Infrastructuur en Waterstaat

Menu

A29: Heinenoordtunnel afgesloten voor renovatie; december

Lees het nieuwsbericht >

A29 Heinenoordtunnel dicht; 1 - 4 december

Bestrijding gladheid op de wegen

Kustversterking Scheveningen

Code geel Vanmiddag en vanavond gladheid door sneeuw (KNMI ↗)

nce
ing
ces]

Moore Machines benefits

- Easy to use
- Easy to modify
- Easy to verify

Moore Machines benefits

- Easy to use
- Easy to modify
- Easy to verify
- By the inventor of Moore's law

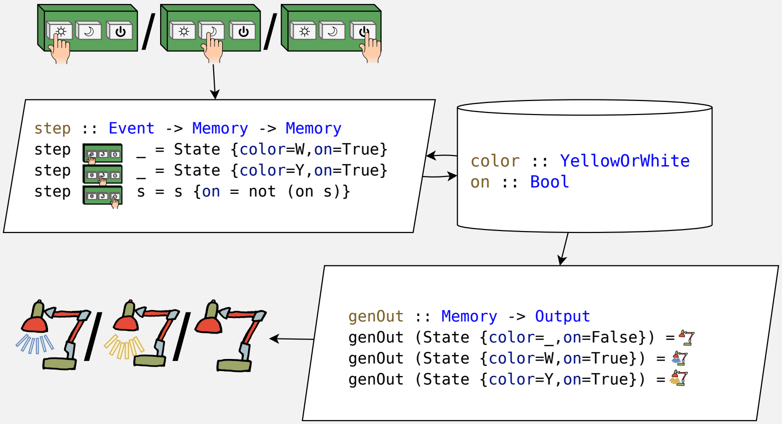
Moore Machines benefits

- Easy to use
- Easy to modify
- Easy to verify
- By the inventor of Moore's law
    - False! Edward F. Moore vs Gordon E. Moore

Moore Machines benefits

- Easy to use
- Easy to modify
- Easy to verify
- By the inventor of Moore's law
  - False! Edward F. Moore vs Gordon E. Moore
- Easy to implement 👉

Moore Machines in Haskell



```
step :: Event -> Memory -> Memory
step      _ = State {color=W,on=True}
step      _ = State {color=Y,on=True}
step      s = s {on = not (on s)}
```

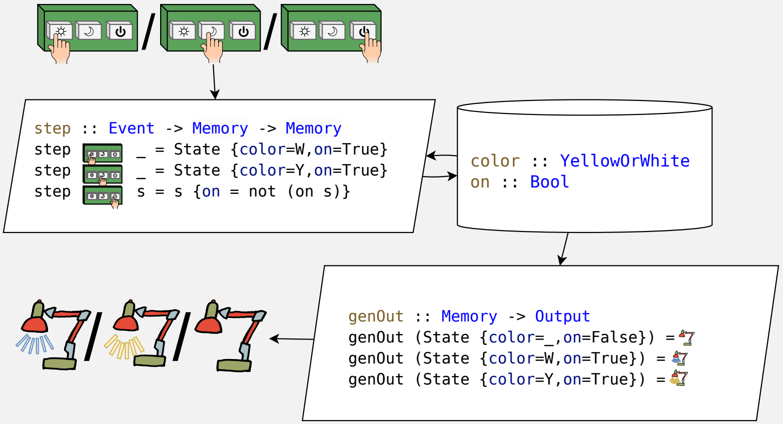
```
color :: YellowOrWhite
on :: Bool
```

```
genOut :: Memory -> Output
genOut (State {color=_,on=False}) =
genOut (State {color=W,on=True})  =
genOut (State {color=Y,on=True})  =
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Moore Machines in Haskell



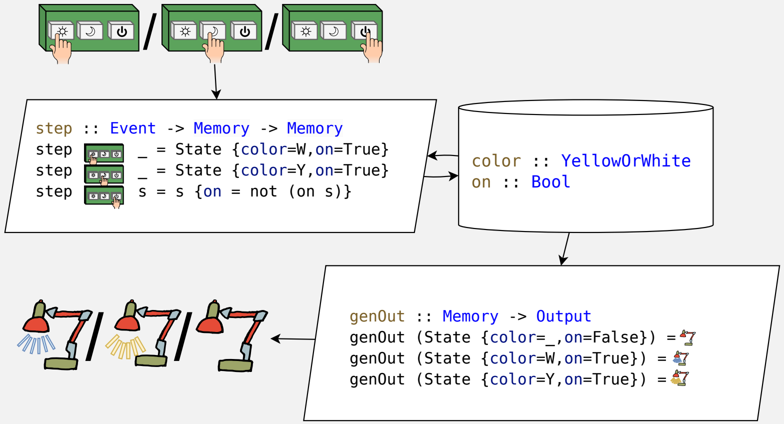
```haskell
step :: Event -> Memory -> Memory
step   _ = State {color=W,on=True}
step   _ = State {color=Y,on=True}
step   s = s {on = not (on s)}
```

```haskell
color :: YellowOrWhite
on :: Bool
```

```haskell
genOut :: Memory -> Output
genOut (State {color=_,on=False}) =
genOut (State {color=W,on=True}) =
genOut (State {color=Y,on=True}) =
```

```haskell
data Moore event memory output = Moore
  { step :: event → memory → memory
  , genOut :: memory → output
  , s0 :: memory}
```

Moore Machines in Haskell



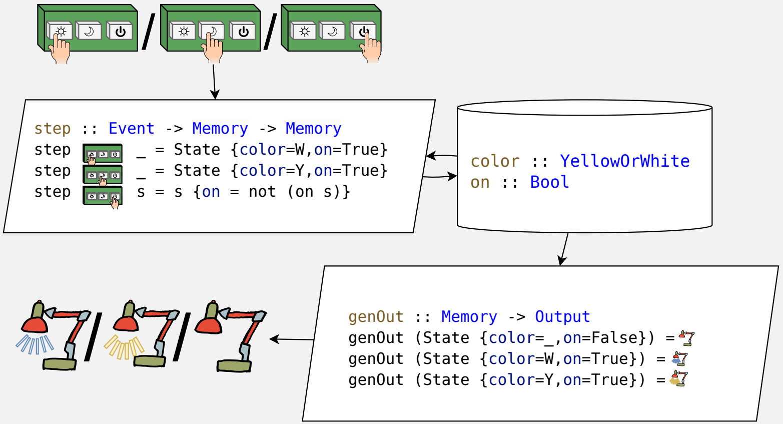
```
step :: Event -> Memory -> Memory
step      _ = State {color=W,on=True}
step      _ = State {color=Y,on=True}
step      s = s {on = not (on s)}
```

```
color :: YellowOrWhite
on :: Bool
```

```
genOut :: Memory -> Output
genOut (State {color=_,on=False}) =
genOut (State {color=W,on=True}) =
genOut (State {color=Y,on=True}) =
```

```
data Moore symbol memory output = Moore
  { step :: symbol → memory → memory
  , genOut :: memory → output
  , s0 :: memory}
```

Moore Machines in Haskell



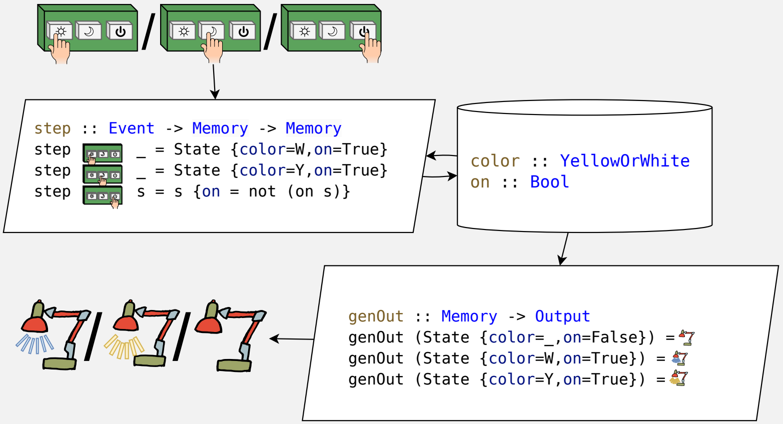
```
step :: Event -> Memory -> Memory
step        _ = State {color=W,on=True}
step        _ = State {color=Y,on=True}
step        s = s {on = not (on s)}
```

```
color :: YellowOrWhite
on :: Bool
```

```
genOut :: Memory -> Output
genOut (State {color=_,on=False}) =
genOut (State {color=W,on=True})  =
genOut (State {color=Y,on=True})  =
```

```
data Moore symbol state output = Moore
  { step :: symbol → state → state
  , genOut :: state → output
  , s0 :: state}
```

Moore Machines in Haskell



```
step :: Event -> Memory -> Memory
step    _ = State {color=W,on=True}
step    _ = State {color=Y,on=True}
step    s = s {on = not (on s)}
```

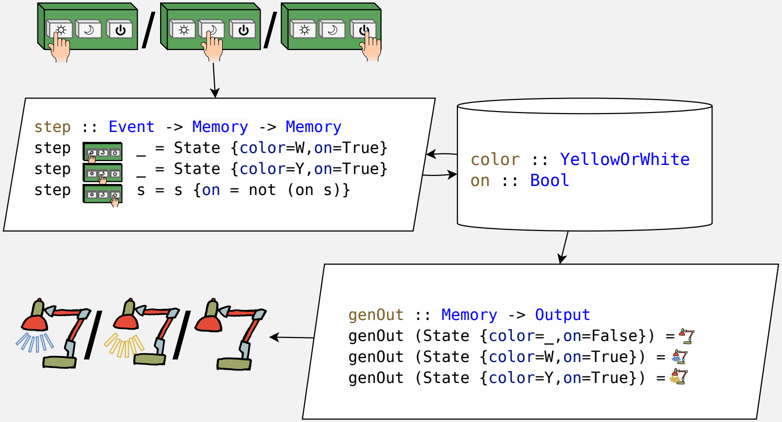
```
color :: YellowOrWhite
on :: Bool
```

```
genOut :: Memory -> Output
genOut (State {color=_,on=False}) =
genOut (State {color=W,on=True})  =
genOut (State {color=Y,on=True})  =
```

```
data Moore symbol state output = Moore
  { step :: symbol → state → state
  , genOut :: state → output
  , s0 :: state}
```

```
type DFA symbol state = Moore symbol state Bool
```

Moore Machines in Haskell

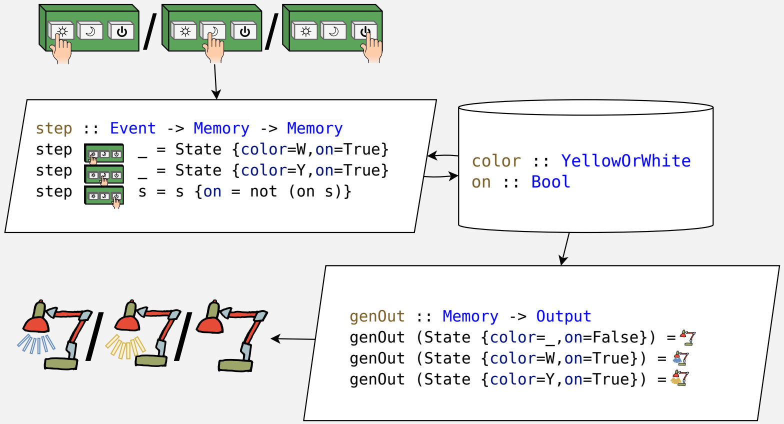


```haskell
step :: Event -> Memory -> Memory
step    _ = State {color=W,on=True}
step    _ = State {color=Y,on=True}
step    s = s {on = not (on s)}
```

```haskell
color :: YellowOrWhite
on :: Bool
```

```haskell
genOut :: Memory -> Output
genOut (State {color=_,on=False}) =
genOut (State {color=W,on=True}) =
genOut (State {color=Y,on=True}) =
```

Moore Machines in Haskell

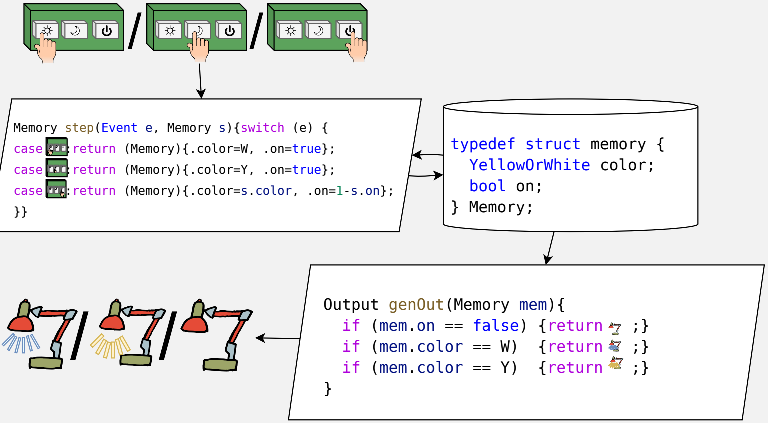


```
step :: Event -> Memory -> Memory
step      _ = State {color=W,on=True}
step      _ = State {color=Y,on=True}
step      s = s {on = not (on s)}
```

```
color :: YellowOrWhite
on :: Bool
```

```
genOut :: Memory -> Output
genOut (State {color=_,on=False}) =
genOut (State {color=W,on=True}) =
genOut (State {color=Y,on=True}) =
```

Moore Machines in C

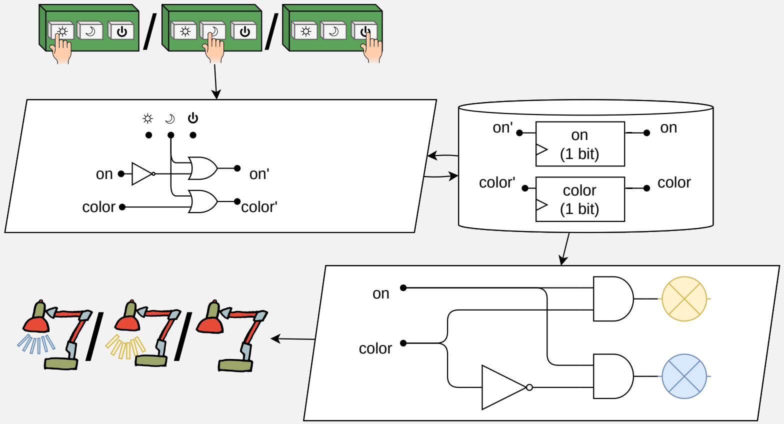


```
Memory step(Event e, Memory s){switch (e) {
case  :return (Memory){.color=W, .on=true};
case  :return (Memory){.color=Y, .on=true};
case  :return (Memory){.color=s.color, .on=1-s.on};
}}
```

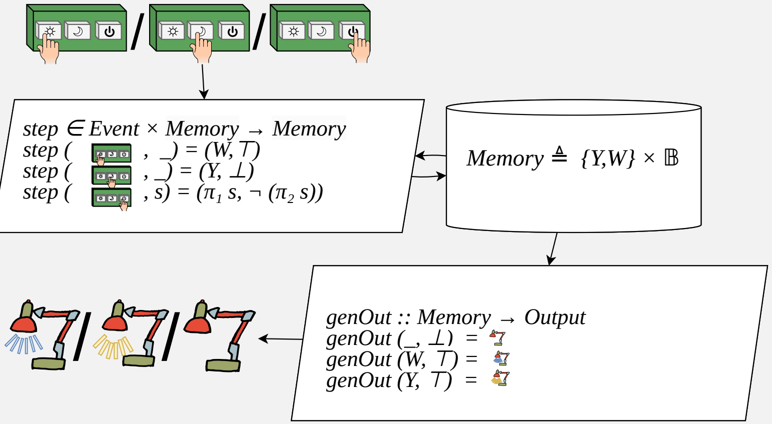
```
typedef struct memory {
  YellowOrWhite color;
  bool on;
} Memory;
```

```
Output genOut(Memory mem){
  if (mem.on == false) {return   ;}
  if (mem.color == W)  {return   ;}
  if (mem.color == Y)  {return   ;}
}
```

Moore Machines in Hardware



Moore Machines in Mathematics



$step \in Event \times Memory \rightarrow Memory$
$step\ (\quad , \_) = (W, \top)$
$step\ (\quad , \_) = (Y, \bot)$
$step\ (\quad , s) = (\pi_1\ s, \neg\ (\pi_2\ s))$

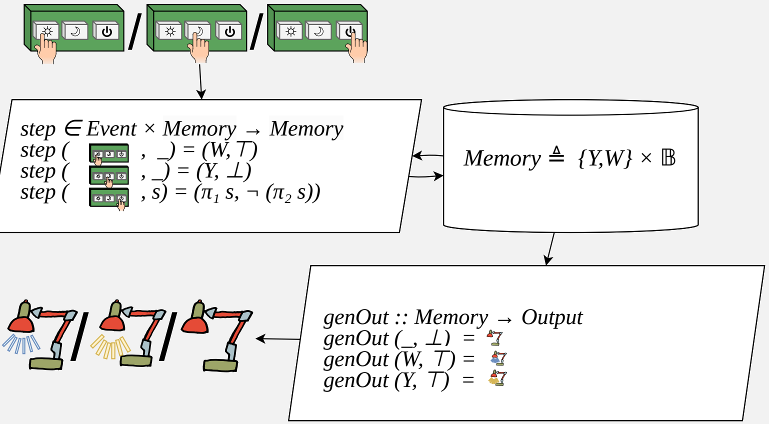
$Memory \triangleq \{Y,W\} \times \mathbb{B}$

$genOut :: Memory \rightarrow Output$
$genOut\ (\_, \bot) =$
$genOut\ (W, \top) =$
$genOut\ (Y, \top) =$

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

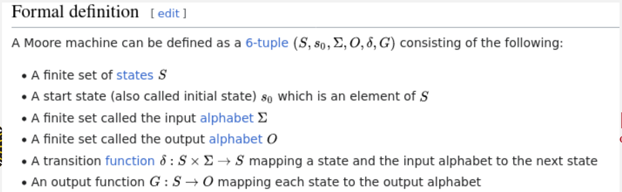
Moore Machines in Mathematics



$step \in Event \times Memory \to Memory$
$step\ (\quad , \_) = (W, \top)$
$step\ (\quad , \_) = (Y, \bot)$
$step\ (\quad , s) = (\pi_1\ s, \neg\ (\pi_2\ s))$

$Memory \triangleq \{Y,W\} \times \mathbb{B}$

$genOut :: Memory \to Output$
$genOut\ (\_, \bot) =$
$genOut\ (W, \top) =$
$genOut\ (Y, \top) =$

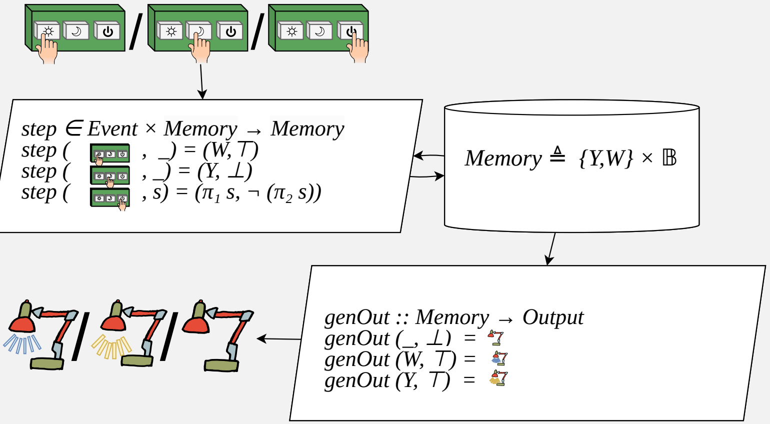


Formal definition [edit]

A Moore machine can be defined as a 6-tuple $(S, s_0, \Sigma, O, \delta, G)$ consisting of the following:

- A finite set of states S
- A start state (also called initial state) s_0 which is an element of S
- A finite set called the input alphabet Σ
- A finite set called the output alphabet O
- A transition function $\delta : S \times \Sigma \to S$ mapping a state and the input alphabet to the next state
- An output function $G : S \to O$ mapping each state to the output alphabet

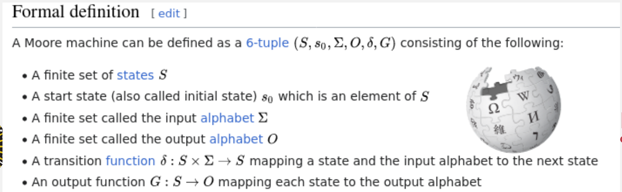
Moore Machines in Mathematics



$$\begin{aligned}
&step \in Event \times Memory \to Memory\\
&step\ (\quad, \_) = (W, \top)\\
&step\ (\quad, \_) = (Y, \bot)\\
&step\ (\quad, s) = (\pi_1\ s, \neg\ (\pi_2\ s))
\end{aligned}$$

$$Memory \triangleq \{Y,W\} \times \mathbb{B}$$

$$\begin{aligned}
&genOut :: Memory \to Output\\
&genOut\ (\_, \bot) =\\
&genOut\ (W, \top) =\\
&genOut\ (Y, \top) =
\end{aligned}$$



Formal definition [edit]

A Moore machine can be defined as a 6-tuple $(S, s_0, \Sigma, O, \delta, G)$ consisting of the following:

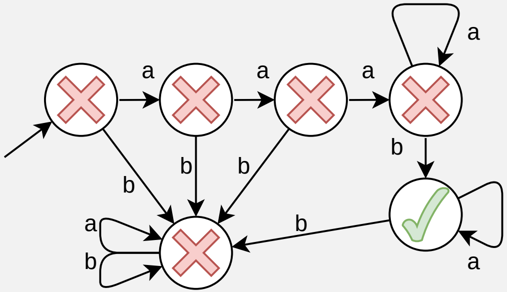
- A finite set of states S
- A start state (also called initial state) s_0 which is an element of S
- A finite set called the input alphabet Σ
- A finite set called the output alphabet O
- A transition function $\delta : S \times \Sigma \to S$ mapping a state and the input alphabet to the next state
- An output function $G : S \to O$ mapping each state to the output alphabet

Moore Machines summary

- Easy to use
- Easy to modify
- Easy to verify
- Easy to implement LaTeX

Moore Machines for RegExp Matching

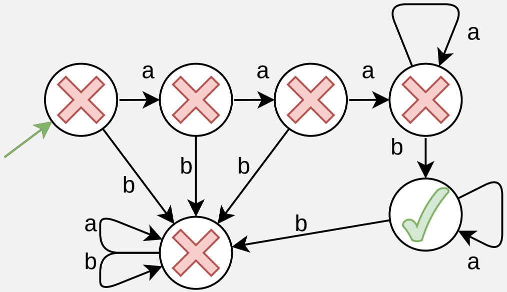
`a*aaaba*`



`aaaaaaabaabba`

Moore Machines for RegExp Matching

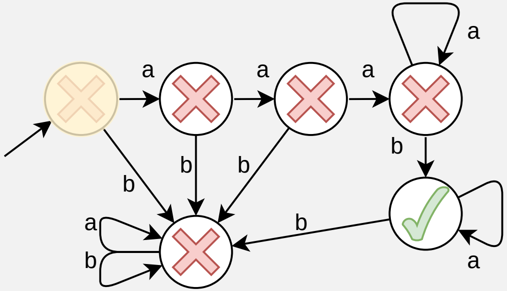
`a*aaaba*`



`aaaaaaabaabba`

Moore Machines for RegExp Matching

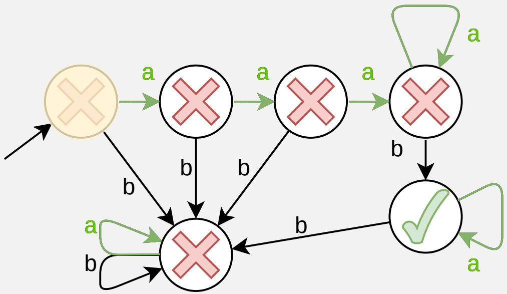
`a*aaaba*`



`|aaaaaaabaabba`

Moore Machines for RegExp Matching

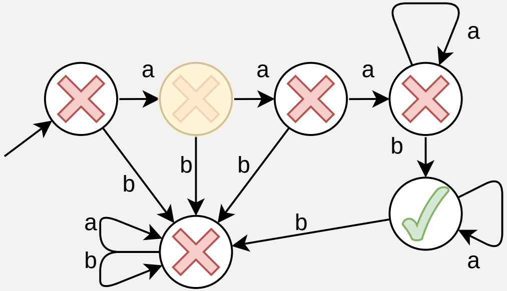
`a*aaaba*`



`[aaaaaaabaabba`

Moore Machines for RegExp Matching

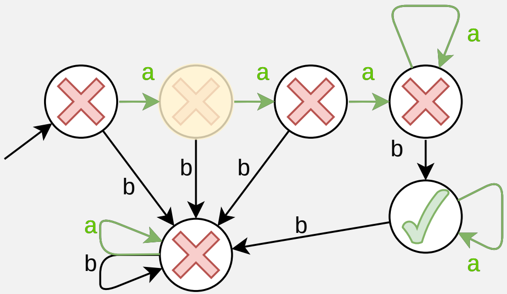
a*aaaba*



a|aaaaaabaabba

Moore Machines for RegExp Matching

a*aaaba*



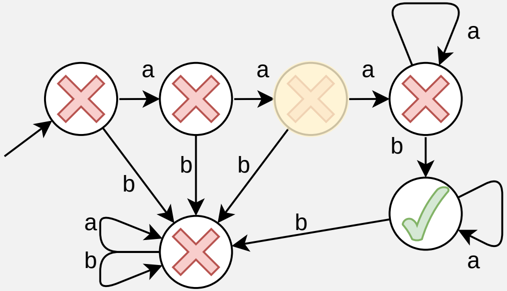
a[aaaaabaabba

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Moore Machines for RegExp Matching

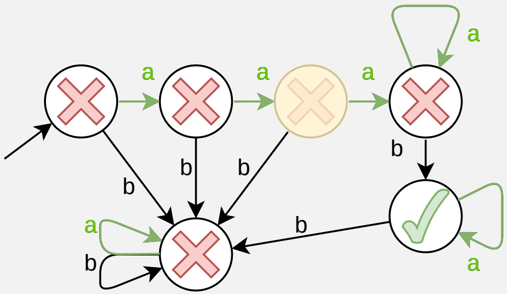
`a*aaaba*`



`aa|aaaaabaabba`

Moore Machines for RegExp Matching

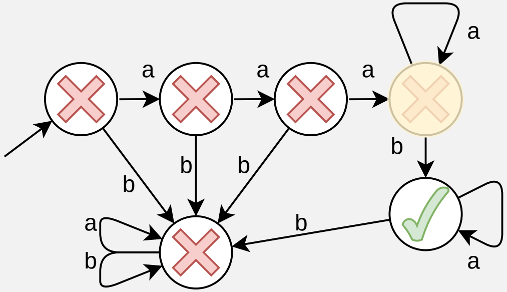
`a*aaaba*`



`aa[aaaabaabba`

Moore Machines for RegExp Matching

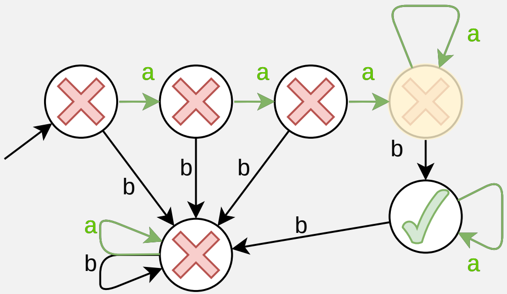
a*aaaba*



aaa|aaaabaabba

Moore Machines for RegExp Matching

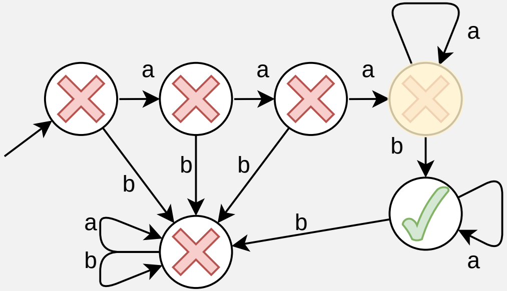
a*aaaba*



aaa[aaaabaabba

Moore Machines for RegExp Matching

`a*aaaba*`



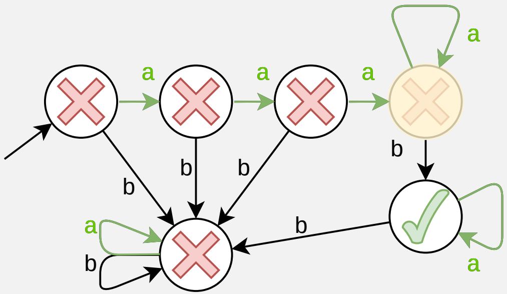
`aaaa|aaabaabba`

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Moore Machines for RegExp Matching

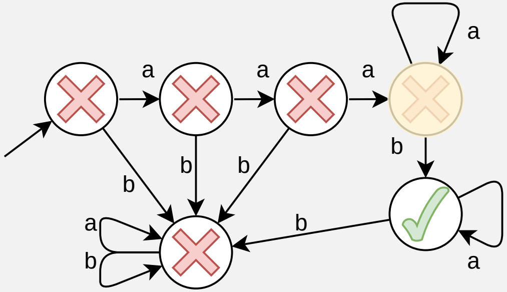
```
a*aaaba*
```



```
aaaa[aaabaabba
```

Moore Machines for RegExp Matching

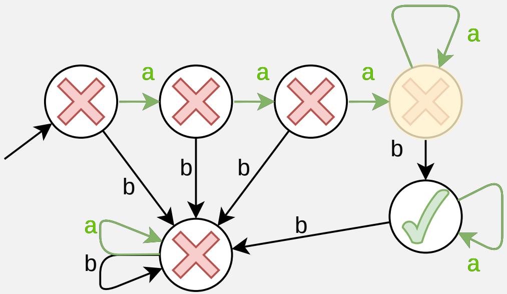
`a*aaaba*`



`aaaaa|aabaabba`

Moore Machines for RegExp Matching

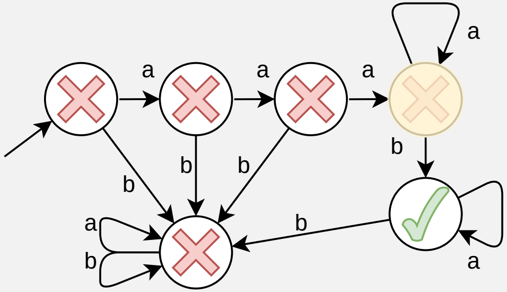
`a*aaaba*`



`aaaaa[aabaabba`

Moore Machines for RegExp Matching

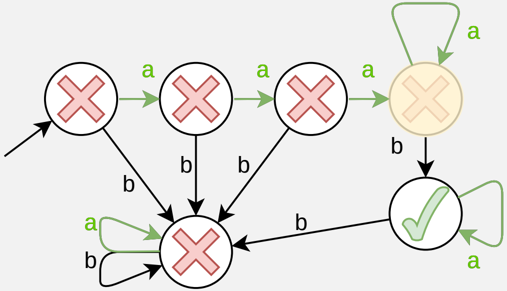
`a*aaaba*`



`aaaaaa|abaabba`

Moore Machines for RegExp Matching

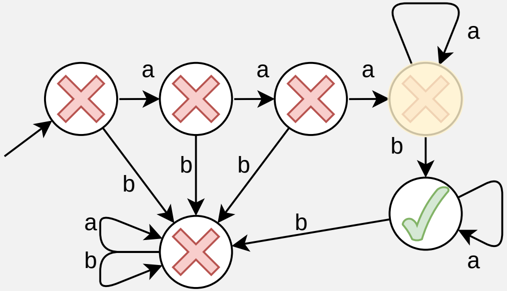
a*aaaba*



aaaaaa[abaabba

Moore Machines for RegExp Matching

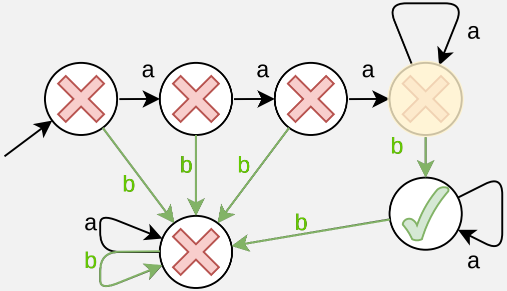
`a*aaaba*`



`aaaaaaa|baabba`

Moore Machines for RegExp Matching

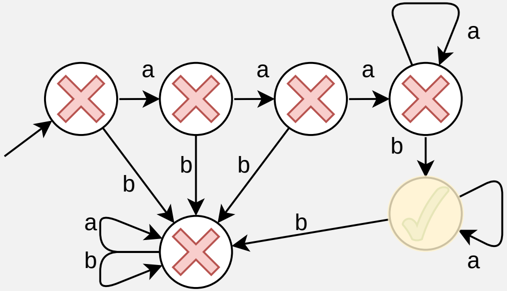
`a*aaaba*`



`aaaaaaa[baabba`

Moore Machines for RegExp Matching

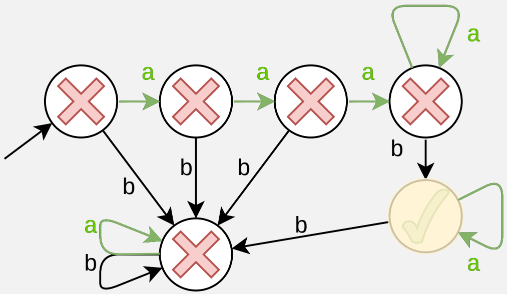
`a*aaaba*`



`aaaaaaab|aabba`

Moore Machines for RegExp Matching

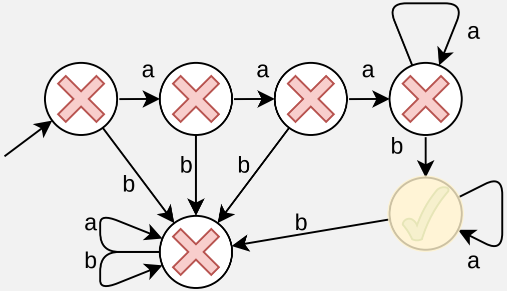
`a*aaaba*`



`aaaaaaab[aabba`

Moore Machines for RegExp Matching

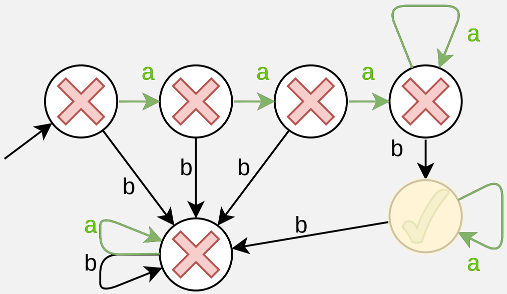
`a*aaaba*`



`aaaaaaaba|abba`

Moore Machines for RegExp Matching

```
a*aaaba*
```



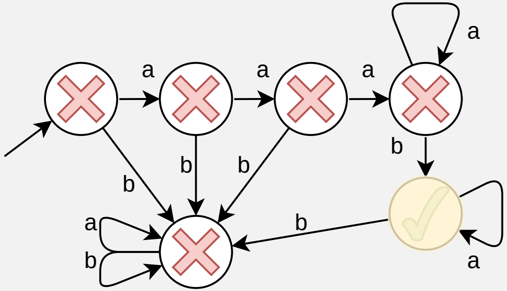
```
aaaaaaaba[abba
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Moore Machines for RegExp Matching

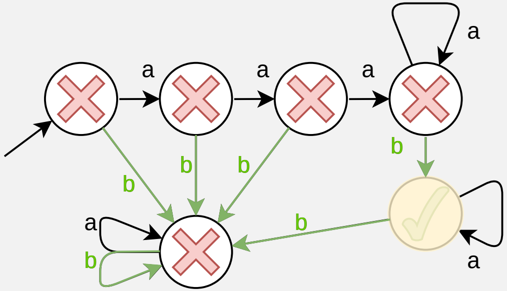
`a*aaaba*`



`aaaaaaabaa|bba`

Moore Machines for RegExp Matching

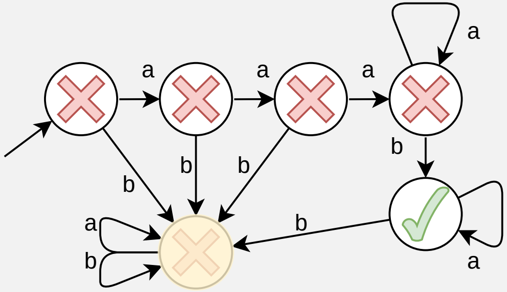
`a*aaaba*`



`aaaaaaabaa[bba`

Moore Machines for RegExp Matching

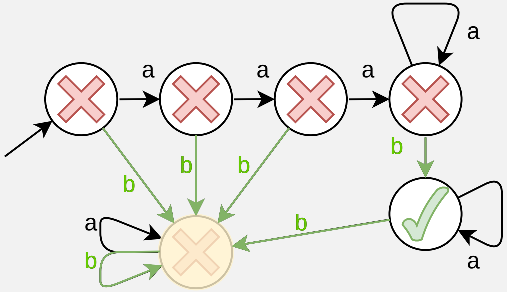
`a*aaaba*`



`aaaaaaabaab|ba`

Moore Machines for RegExp Matching

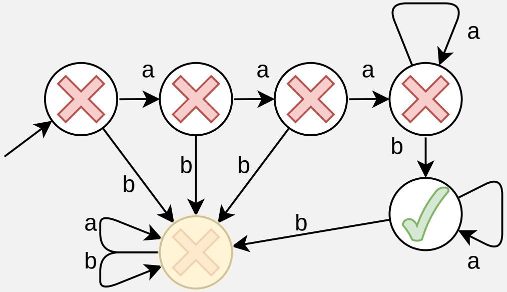
```
a*aaaba*
```



```
aaaaaaabaab[ba
```

Moore Machines for RegExp Matching

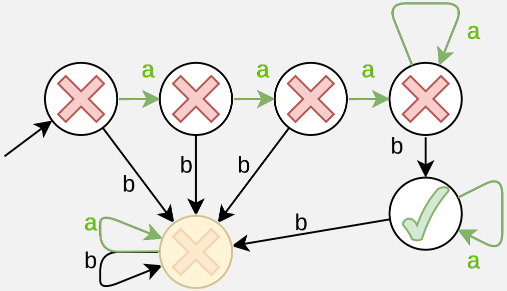
```
a*aaaba*
```



```
aaaaaaabaabb|a
```

Moore Machines for RegExp Matching

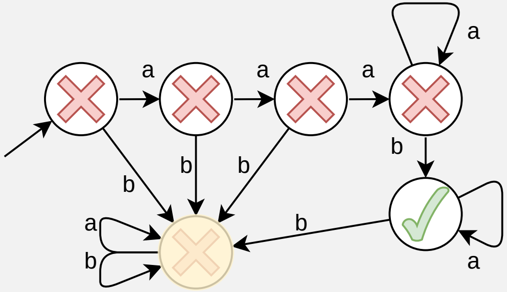
```
a*aaaba*
```



```
aaaaaaabaabb[a
```

Moore Machines for RegExp Matching

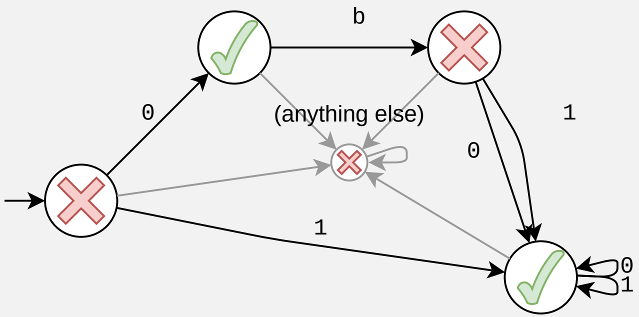
`a*aaaba*`



`aaaaaaabaabba|`

More Moore Machine Matching

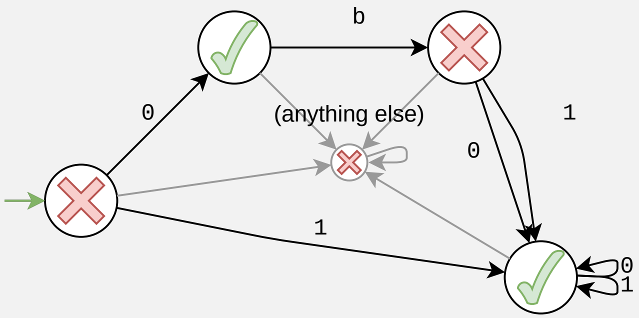
`(0b)?(0|1)+`



`0b110100,10`

More Moore Machine Matching

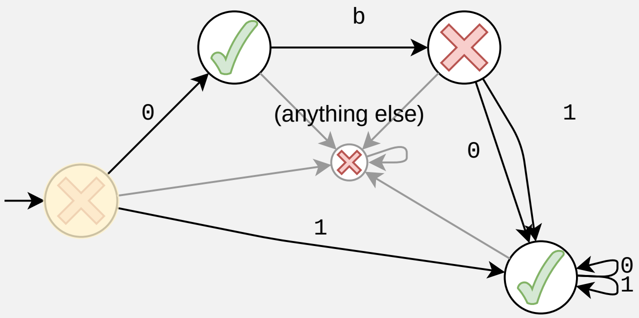
`(0b)?(0|1)+`



```
0b110100,10
```

More Moore Machine Matching

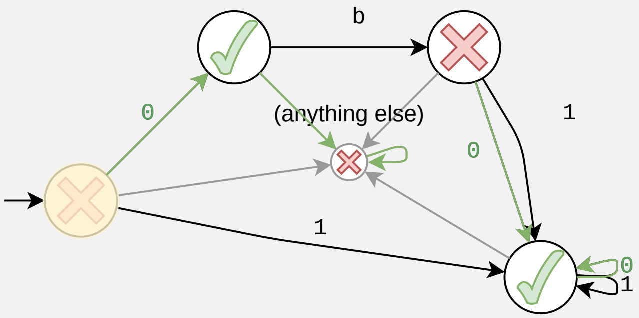
`(0b)?(0|1)+`



`|0b110100,10`

More Moore Machine Matching

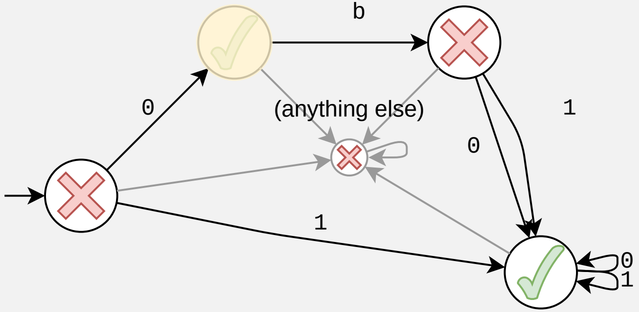
`(0b)?(0|1)+`



`[0b110100,10`

More Moore Machine Matching

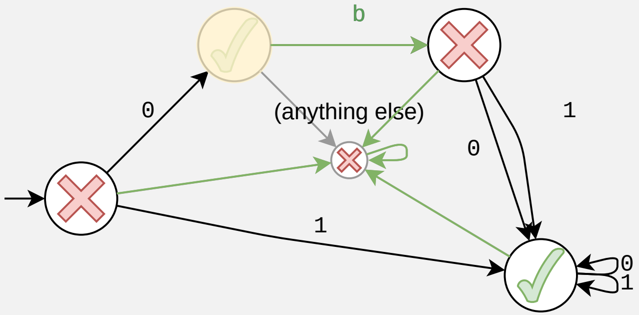
`(0b)?(0|1)+`



`0|b110100,10`

More Moore Machine Matching

`(0b)?(0|1)+`



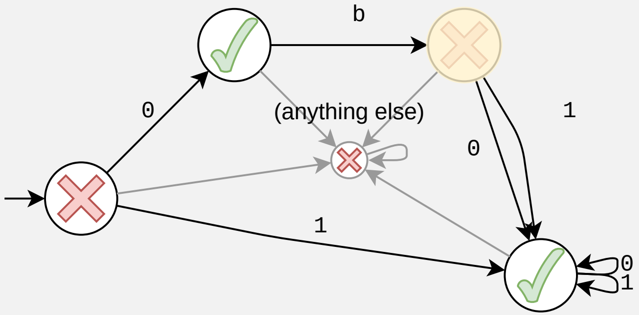
`0[b110100,10`

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

More Moore Machine Matching

`(0b)?(0|1)+`



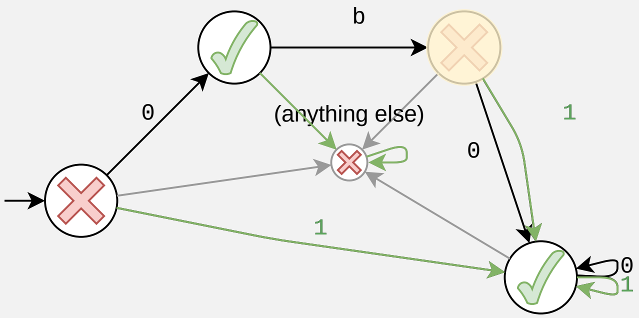
`0b|110100,10`

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

More Moore Machine Matching

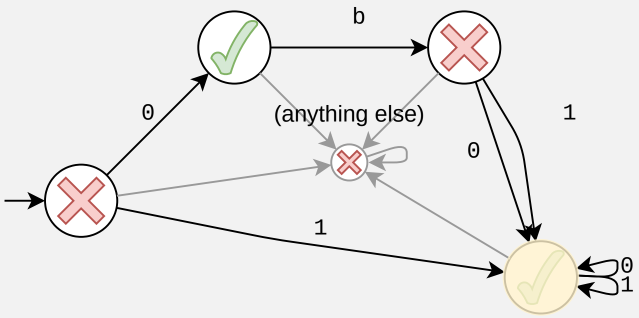
`(0b)?(0|1)+`



`0b[110100,10`

More Moore Machine Matching

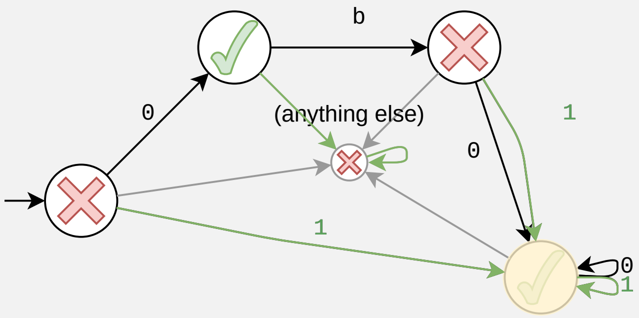
`(0b)?(0|1)+`



`0b1|10100,10`

More Moore Machine Matching

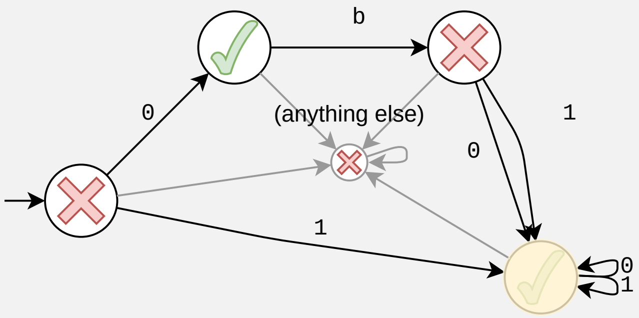
`(0b)?(0|1)+`



`0b1[10100,10`

More Moore Machine Matching

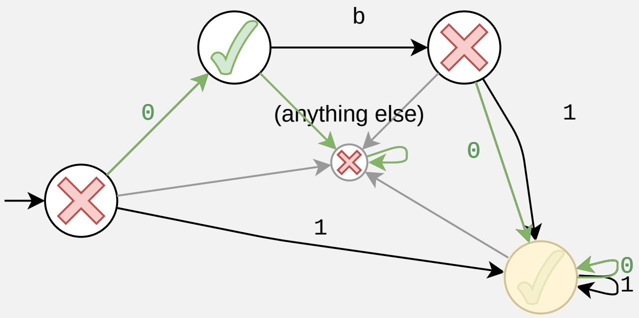
`(0b)?(0|1)+`



`0b11|0100,10`

More Moore Machine Matching

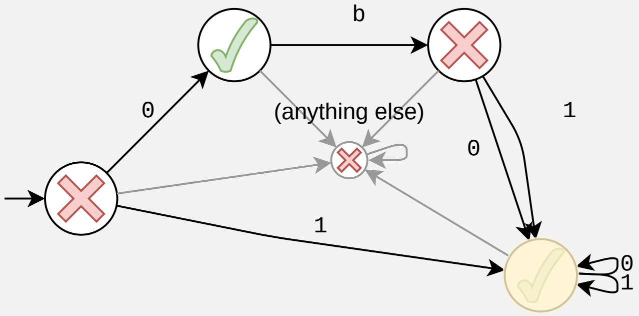
`(0b)?(0|1)+`



`0b11[0100,10`

More Moore Machine Matching

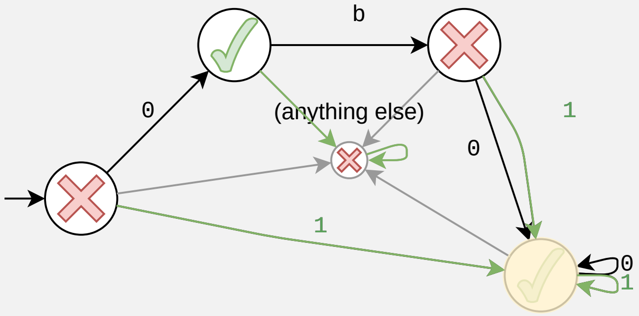
`(0b)?(0|1)+`



`0b110|100,10`

More Moore Machine Matching

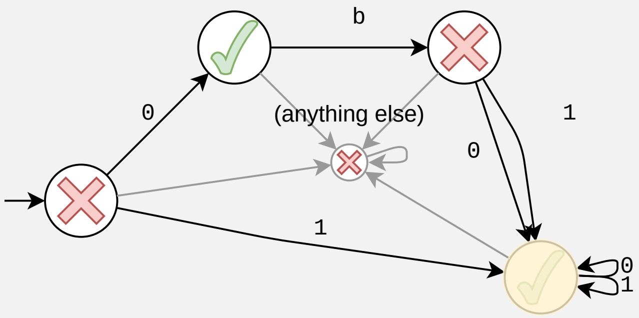
`(0b)?(0|1)+`



`0b110[100,10`

More Moore Machine Matching

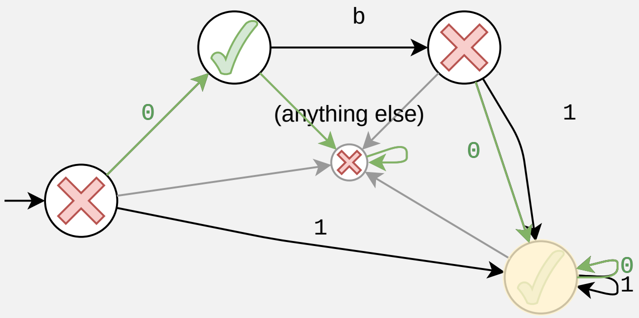
`(0b)?(0|1)+`



`0b1101|00,10`

More Moore Machine Matching

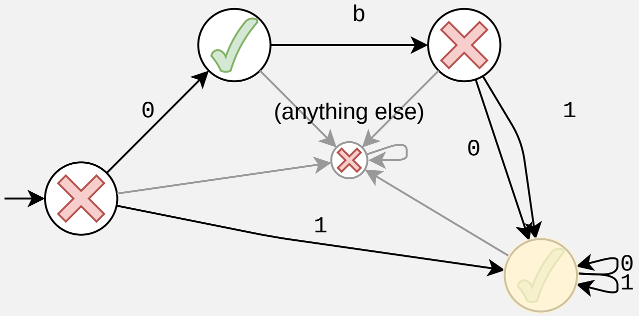
`(0b)?(0|1)+`



`0b1101[00,10`

More Moore Machine Matching

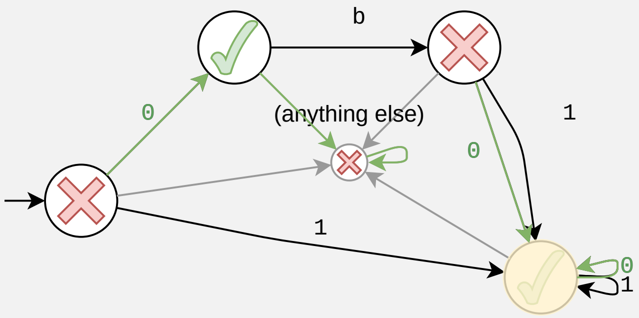
`(0b)?(0|1)+`



`0b11010|0,10`

More Moore Machine Matching

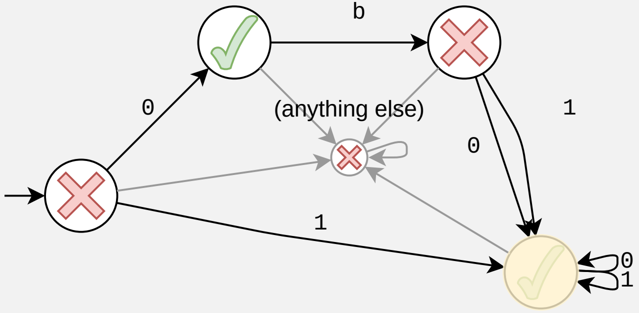
```
(0b)?(0|1)+
```



```
0b11010[0,10
```

More Moore Machine Matching

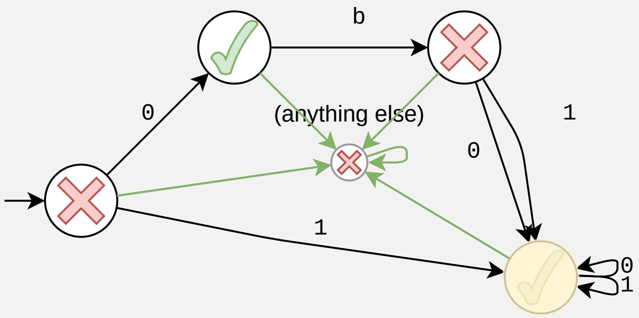
`(0b)?(0|1)+`



`0b110100|,10`

More Moore Machine Matching

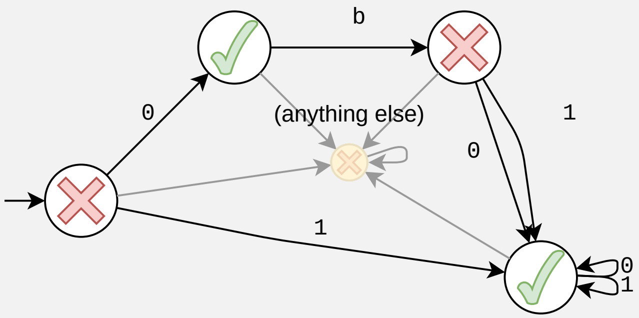
`(0b)?(0|1)+`



`0b110100[,10`

More Moore Machine Matching

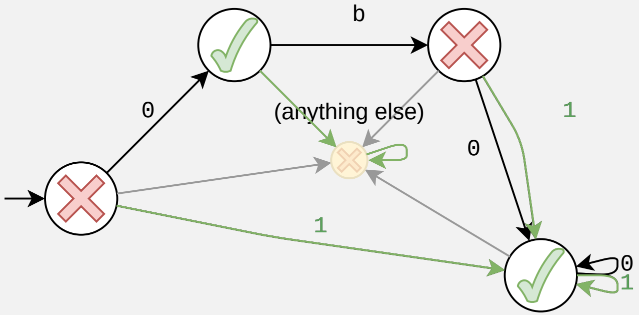
`(0b)?(0|1)+`



`0b110100,|10`

More Moore Machine Matching

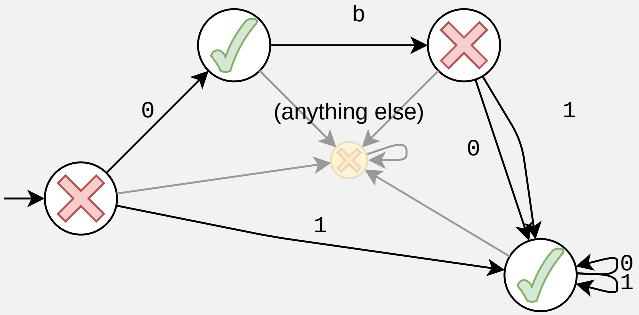
`(0b)?(0|1)+`



`0b110100,[10`

More Moore Machine Matching

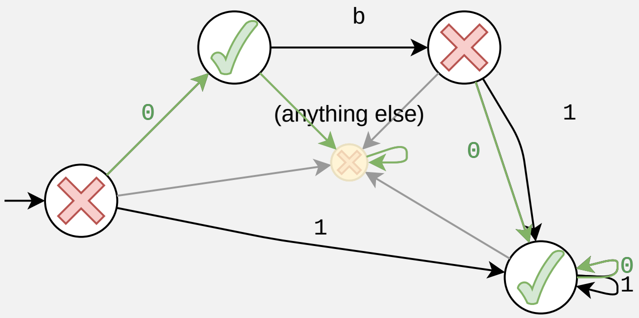
`(0b)?(0|1)+`



`0b110100,1|0`

More Moore Machine Matching

`(0b)?(0|1)+`

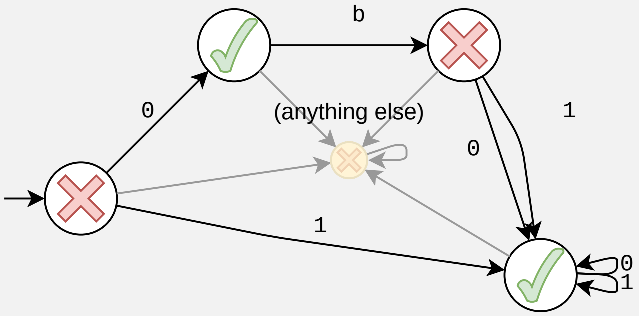


`0b110100,1[0`

Universiteit Utrecht

More Moore Machine Matching

`(0b)?(0|1)+`



`0b110100,10|`

Universiteit Utrecht

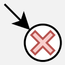
[Faculty of Science
Information and Computing
Sciences]

Making Matching Moore Machines

```
co(bra|d)
```

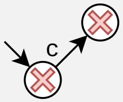
Making Matching Moore Machines

`co(bra|d)`



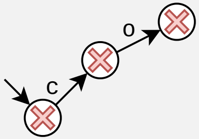
Making Matching Moore Machines

`co(bra|d)`



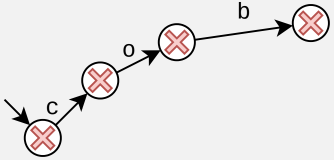
Making Matching Moore Machines

`co(bra|d)`



Making Matching Moore Machines

`co(bra|d)`

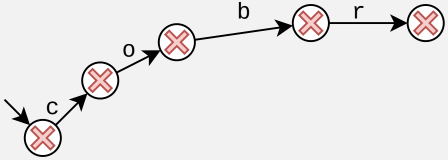


Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Making Matching Moore Machines

`co(bra|d)`

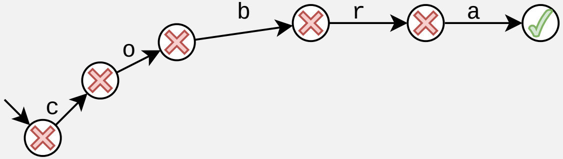


Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Making Matching Moore Machines

`co(bra|d)`

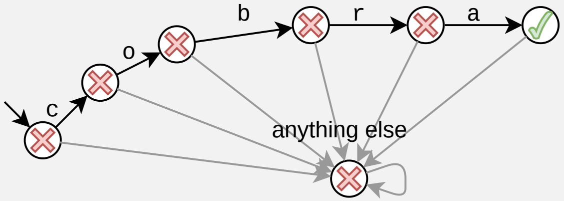


Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

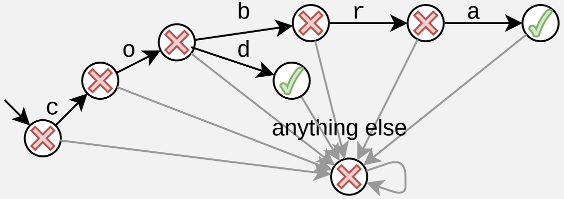
Making Matching Moore Machines

`co(bra|d)`



Making Matching Moore Machines

`co(bra|d)`

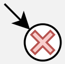


More Making Matching Moore Machines

```
gr(a|e)y|green
```

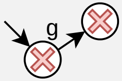
More Making Matching Moore Machines

`gr(a|e)y|green`



More Making Matching Moore Machines

`gr(a|e)y|green`

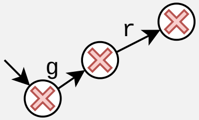


Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

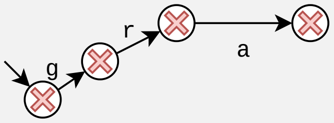
More Making Matching Moore Machines

`gr(a|e)y|green`



More Making Matching Moore Machines

`gr(a|e)y|green`

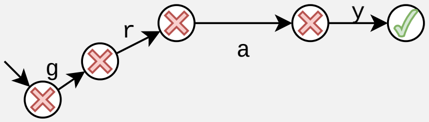


Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

More Making Matching Moore Machines

`gr(a|e)y|green`

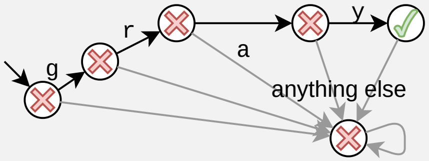


Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

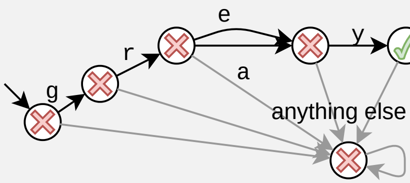
More Making Matching Moore Machines

`gr(a|e)y|green`



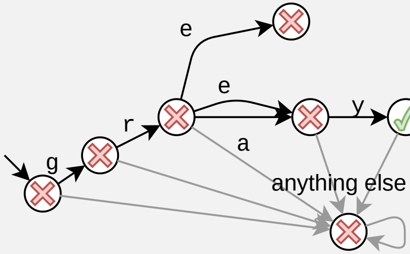
More Making Matching Moore Machines

`gr(a|e)y|green`



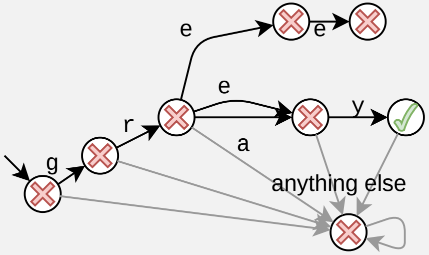
More Making Matching Moore Machines

`gr(a|e)y|green`



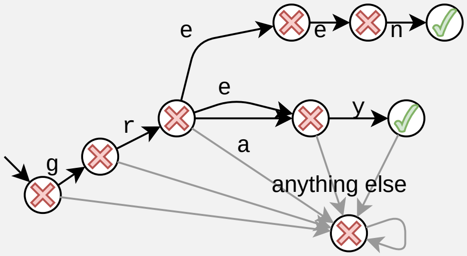
More Making Matching Moore Machines

`gr(a|e)y|green`



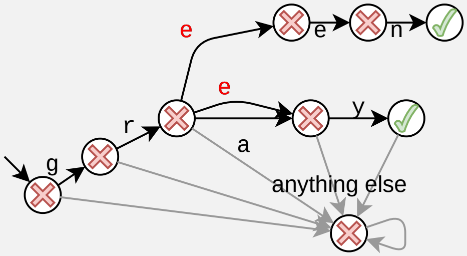
More Making Matching Moore Machines

`gr(a|e)y|green`



More Making Matching Moore Machines

`gr(a|e)y|green`

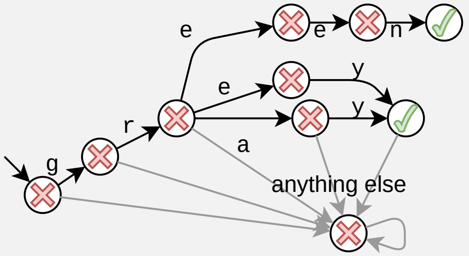


Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

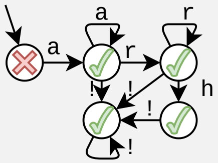
More Making Matching Moore Machines

`gr(a|e)y|green`



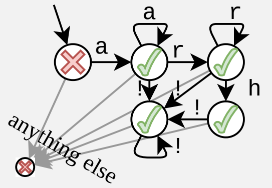
Quiz 1/4

Q: What is wrong with this Moore Machine?



Quiz 1/4

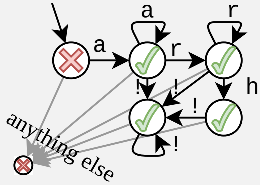
Q: What is wrong with this Moore Machine?



A: Transition Relation must be a function!

Quiz 2/4

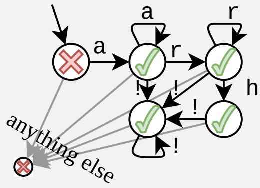
Q: Which strings are matched by this Moore Machine?



1. “arr”
2. “arrgh”
3. “arrh!!!”
4. “arhh!”
5. “aaaaa!!!”
6. “”
7. “arrh me hearties!”

Quiz 2/4

Q: Which strings are matched by this Moore Machine?



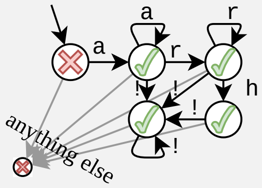
1. “`arr`” 👉
2. “`arrgh`”
3. “`arrh!!!`”👉
4. “`arhh!`”
5. “`aaaaa!!!`”👉
6. “”
7. “`arrh me hearties!`”

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Quiz 3/4

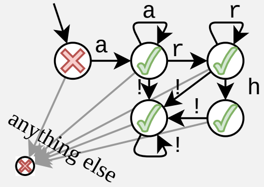
Q: What is the RegExp for this Moore Machine?



1. `arrh!`
2. `aa+r+h!+`
3. `aa*r*h?!*`
4. `a+r*h?!*`
5. `a*r?h!*?`
6. `a*r?h?(!+)?`

Quiz 3/4

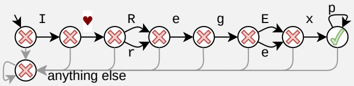
Q: What is the RegExp for this Moore Machine?

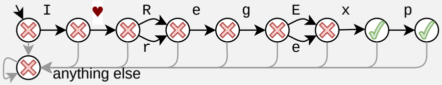


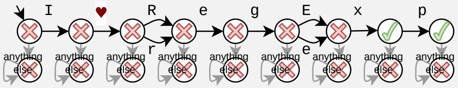
1. `arrh!`
2. `aa+r+h!+`
3. `aa*r*h?!*`👉
4. `a+r*h?!*`👉
5. `a*r?h!*?`
6. `a*r?h?(!+)?`👉

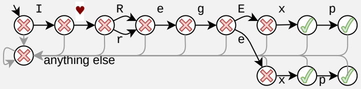
Quiz 4/4

Q: Which Moore Machine matches `I❤(R|r)eg(E|e)xp?` ?

1. 

2. 

3. 

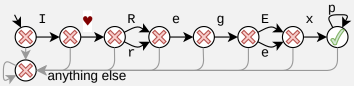
4. 

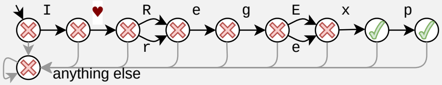
Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

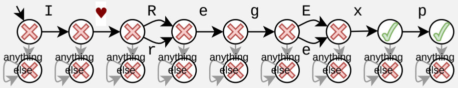
Quiz 4/4

Q: Which Moore Machine matches `I❤️(R|r)eg(E|e)xp?` ?

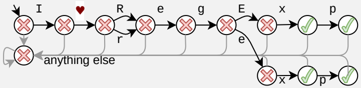
1. 

2. 

👉

3. 

👉

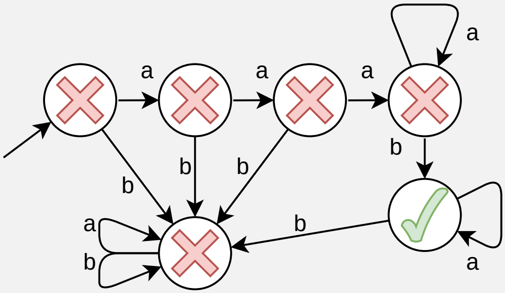
4. 

👉

Universiteit Utrecht

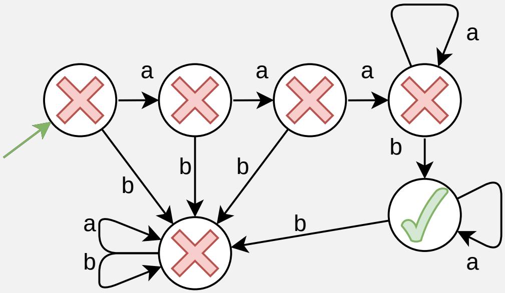
[Faculty of Science
Information and Computing
Sciences]

Recap



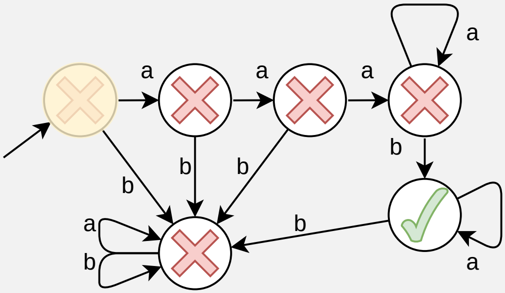
`a*aaaba*` ~~~~~>

Recap



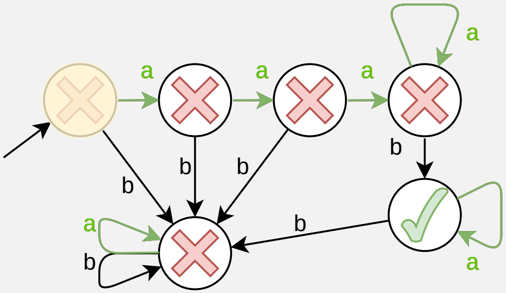
`a*aaaba*` ~~~~~>

Recap



`a*aaaba*` ~~~~~>

Recap

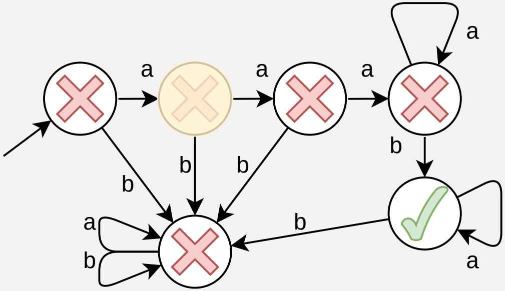


a*aaaba* ~~~~~>

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Recap

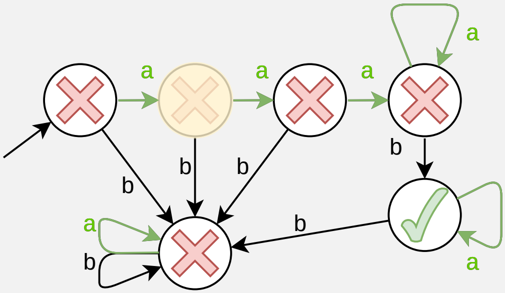


a*aaaba* ~~~~~>

Universiteit Utrecht

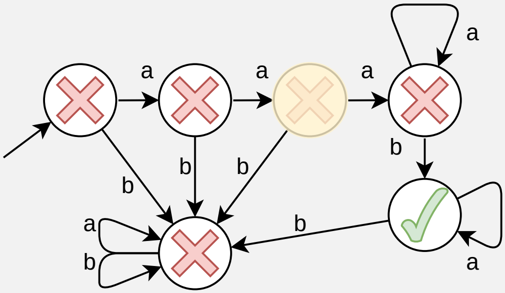
[Faculty of Science
Information and Computing
Sciences]

Recap



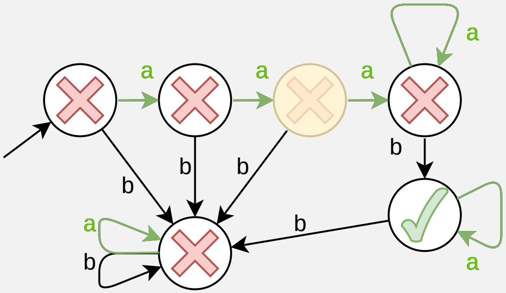
`a*aaaba*` ~~~~~>

Recap



`a*aaaba*` ~~~~~>

Recap

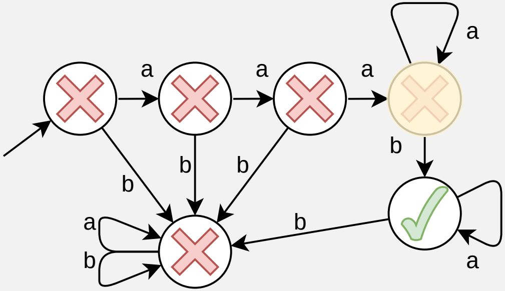


`a*aaaba*` ~~~~~>

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Recap

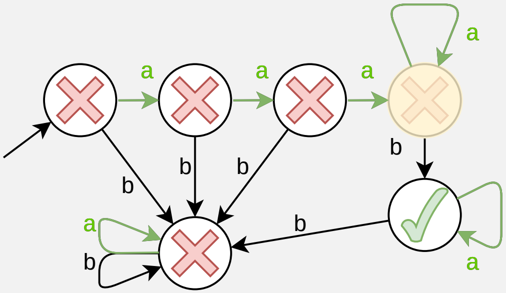


`a*aaaba*` ~~~~~>

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Recap

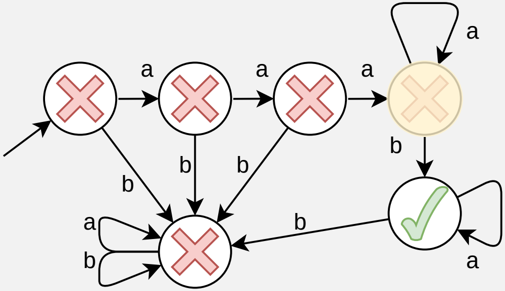


`a*aaaba*` ~~~~~>

Universiteit Utrecht

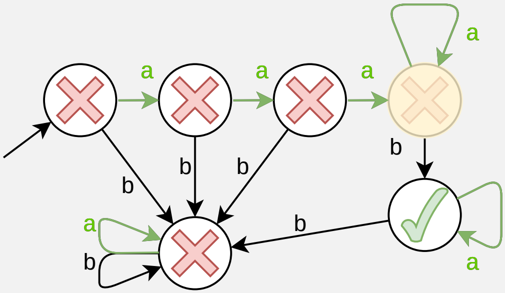
[Faculty of Science
Information and Computing
Sciences]

Recap



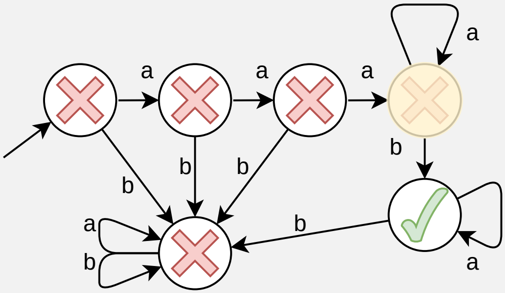
`a*aaaba*` ~~~~~>

Recap



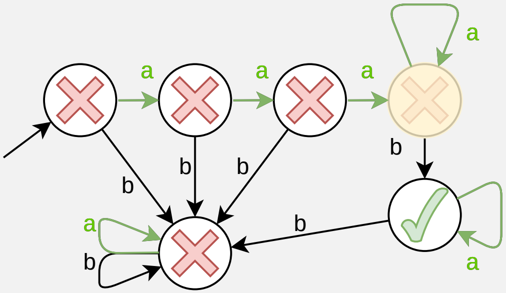
`a*aaaba*` ~~~~~>

Recap



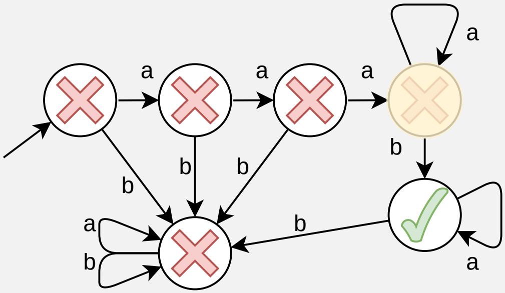
`a*aaaba*` ~~~~~>

Recap



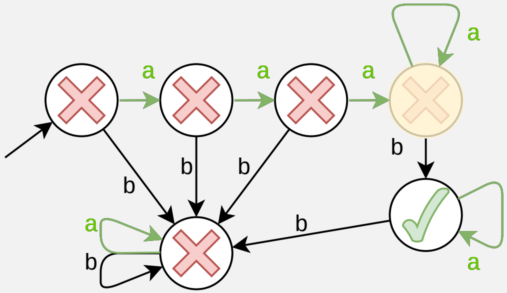
`a*aaaba* ~~~~~>`

Recap



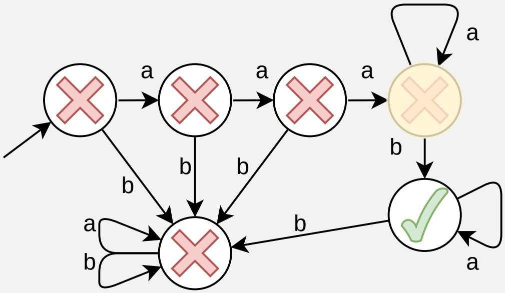
a*aaaba* ~~~~~>

Recap



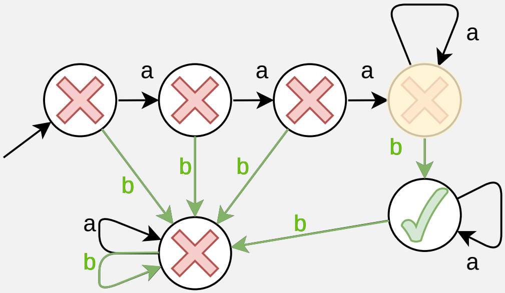
`a*aaaba*` ~~~~~>

Recap



a*aaaba* ~~~~~>

Recap

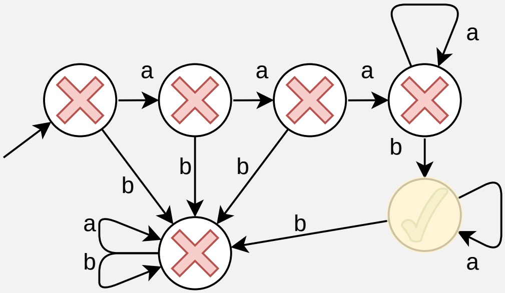


a*aaaba* ~~~~~>

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Recap

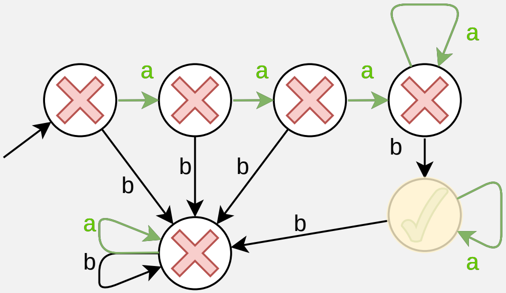


a*aaaba* ~~~~~>

Universiteit Utrecht

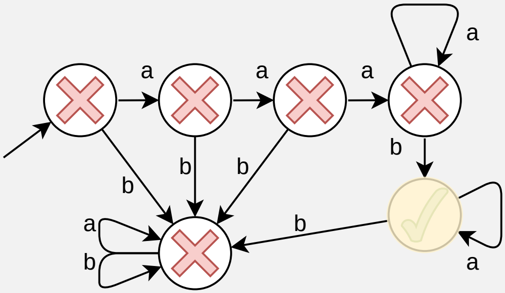
[Faculty of Science
Information and Computing
Sciences]

Recap



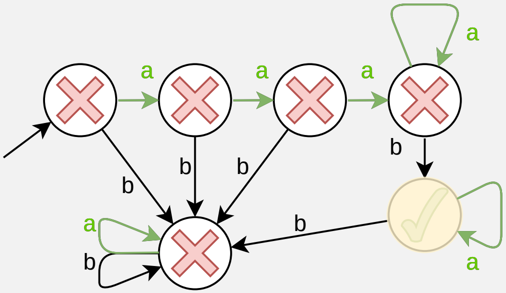
a*aaaba* ~~~~~>

Recap



`a*aaaba*` ~~~~~>

Recap

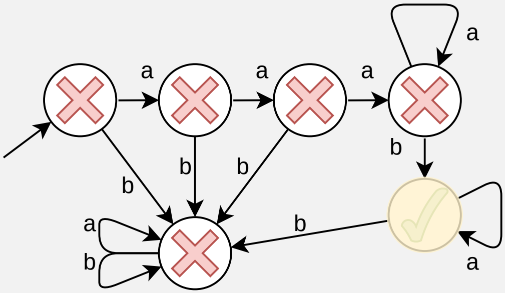


`a*aaaba*` ~~~~~>

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

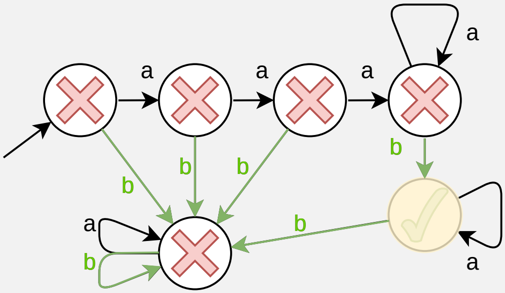
Recap



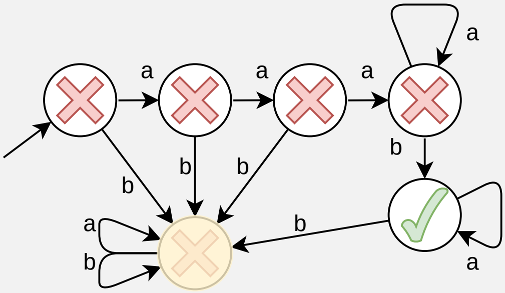
a*aaaba* ~~~~~>

Recap

`a*aaaba*` ~~~~~>

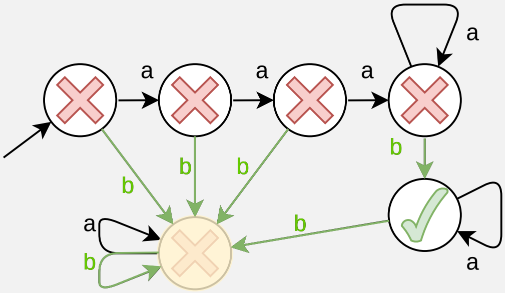


Recap



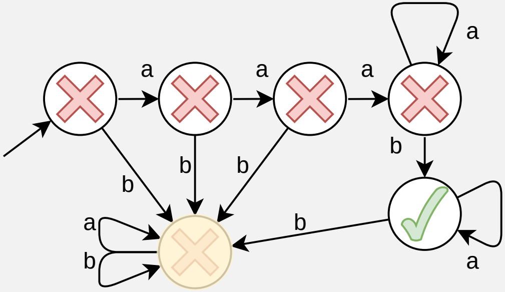
`a*aaaba*` ~~~~~>

Recap



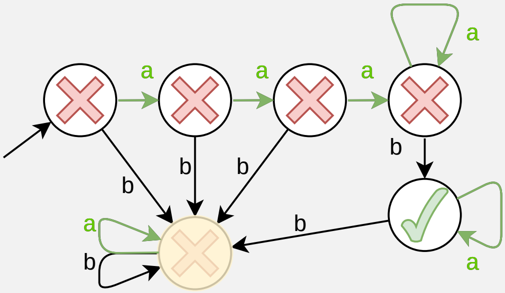
`a*aaaba* ~~~~~>`

Recap



a*aaaba* ~~~~~>

Recap

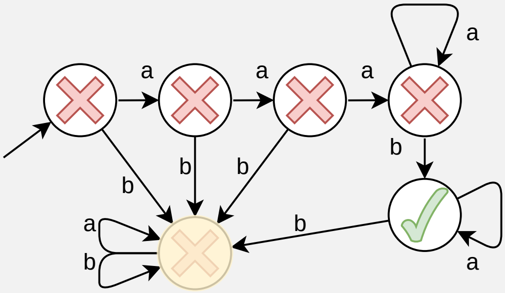


`a*aaaba*` ~~~~~>

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Recap

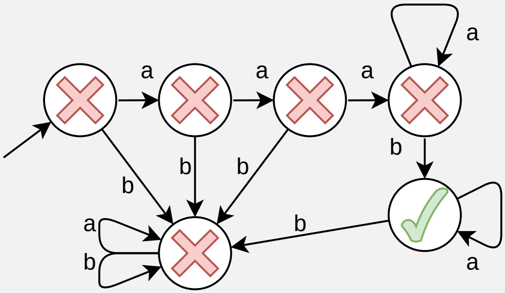


`a*aaaba*` ~~~~~>

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Recap



`a*aaaba*` ~~~~~>

Recap

`a*aaaba*` ~~~~~>

Recap

`a*aaaba* ~~~~~>`

Generally?

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step genOut s0) = ???
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = ???
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = runFrom s0 where
  runFrom :: state → [inp] → state
  runFrom st sys = ???
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = runFrom s0 where
  runFrom :: state → [inp] → state
  runFrom st [] = ???
  runFrom st (i:is) = ???
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = runFrom s0 where
  runFrom :: state → [inp] → state
  runFrom st [] = st
  runFrom st (i:is) = ???
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = runFrom s0 where
  runFrom :: state → [inp] → state
  runFrom st [] = st
  runFrom st (i:is) = runFrom ??? ???
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = runFrom s0 where
  runFrom :: state → [inp] → state
  runFrom st [] = st
  runFrom st (i:is) = runFrom (step i st) is
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = runFrom s0 where
  runFrom :: state → [inp] → state
  runFrom st [] = st
  runFrom st (i:is) = runFrom (step i st) is
```

🕵 Look familiar?

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = runFrom s0 where
  runFrom :: state → [inp] → state
  runFrom st is = foldr step st is
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = runFrom s0 where
  runFrom :: state → [inp] → state
  runFrom = foldr step
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = foldr step s0
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = foldr step s0
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = foldr step s0
```

Running DFAs, generally

```
runDFA :: DFA symbol state → [symbol] → state
runDFA = runMoore
```

Running Moore Machines, generally

```
runMoore :: Moore inp state out → [inp] → state
```

```
runMoore (Moore step _ s0) = foldr step s0
```

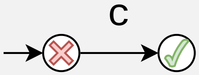
Running DFAs, generally

```
runDFA :: DFA symbol state → [symbol] → state
runDFA = runMoore
```

```
matchesDFA :: DFA symbol state → [symbol] → Bool
matchesDFA dfa = genOutput dfa . runDFA dfa
```

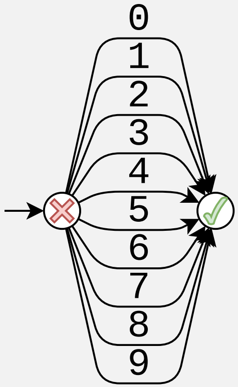
Compiling RegExp to DFA, generally

c



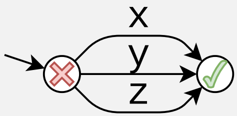
Compiling RegExp to DFA, generally

`\d`



Compiling RegExp to DFA, generally

`[x-z]`

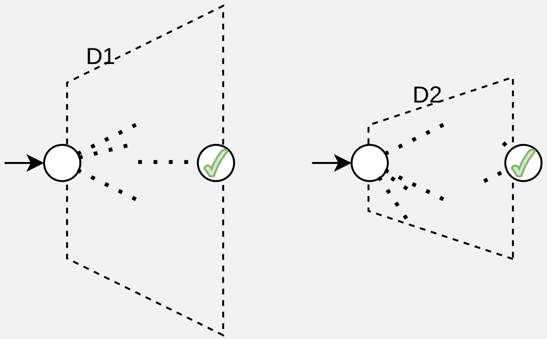


Compiling RegExp to DFA, generally

$\mathtt{r}_1\mathtt{r}_2$

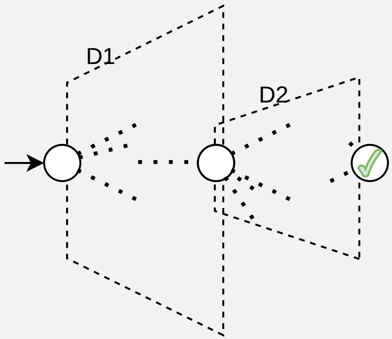
Compiling RegExp to DFA, generally

$\mathtt{r}_1\mathtt{r}_2$



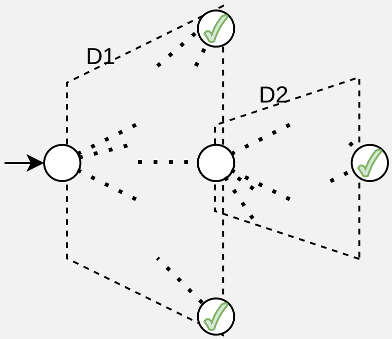
Compiling RegExp to DFA, generally

$r_1 r_2$



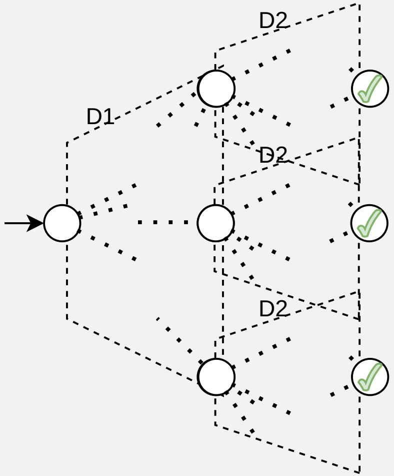
Compiling RegExp to DFA, generally

r_1r_2



Compiling RegExp to DFA, generally

r_1r_2



Universiteit Utrecht

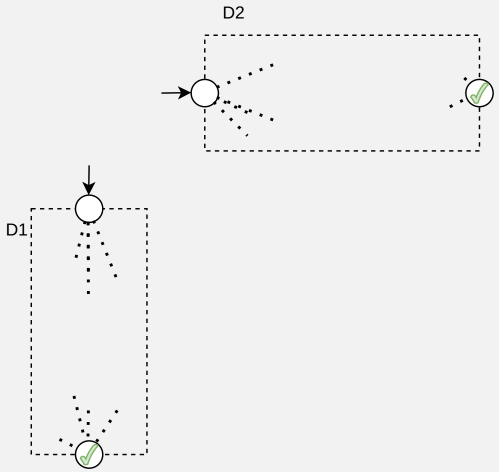
[Faculty of Science
Information and Computing
Sciences]

Compiling RegExp to DFA, generally

$\mathtt{r}_1 | \mathtt{r}_2$

Compiling RegExp to DFA, generally

$r_1 | r_2$

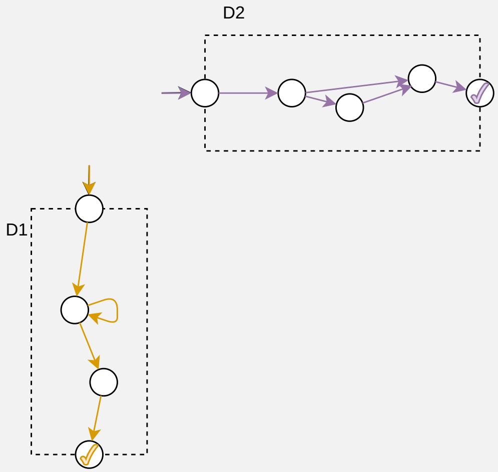


Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

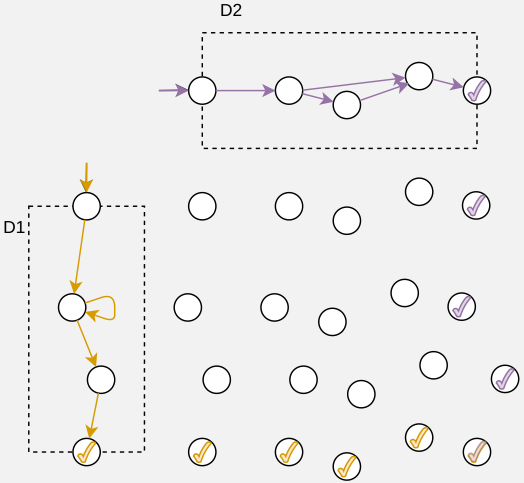
Compiling RegExp to DFA, generally

$r_1 | r_2$



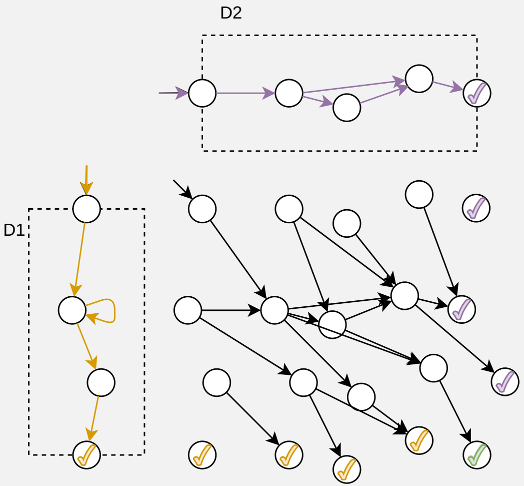
Compiling RegExp to DFA, generally

$r_1 | r_2$



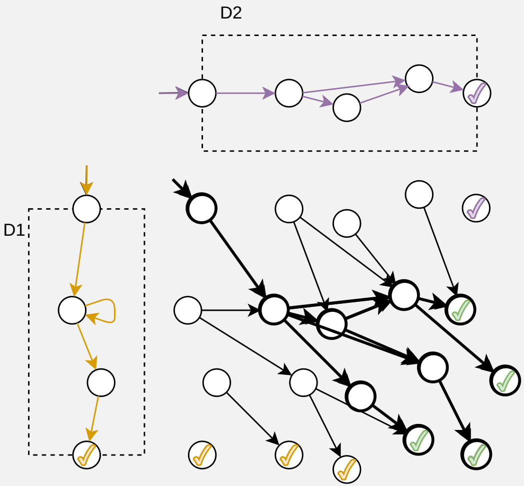
Compiling RegExp to DFA, generally

`r`$_1$ | `r`$_2$



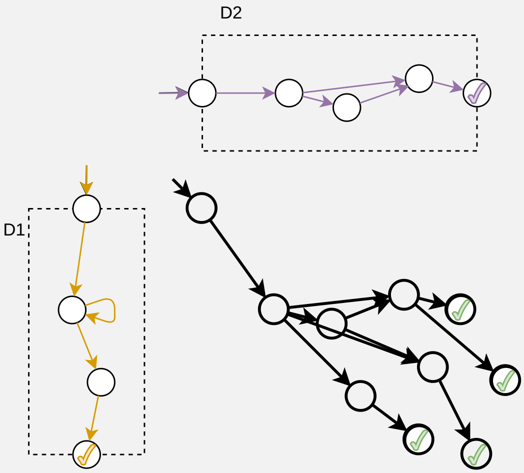
Compiling RegExp to DFA, generally

$r_1 | r_2$



Compiling RegExp to DFA, generally

`r`$_1$`|r`$_2$



Compiling RegExp to DFA, generally

`r+`

Compiling RegExp to DFA, generally

`r+`

🧠

Compiling RegExp to DFA, generally

`r+`

❓

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Compiling RegExp to DFA, generally

`r*`

❓

Compiling RegExp to DFA, generally

`r?`

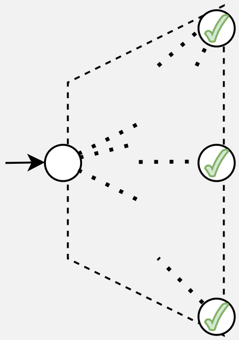
❓

Compiling RegExp to DFA progress

- ✅ `c`
- ✅ `\d`
- ✅ `[x-z]`
- 💣 r_1r_2
- 💣 $r_1|r_2$
- ❓ `r+`
- ❓ `r*`
- ❓ `r?`

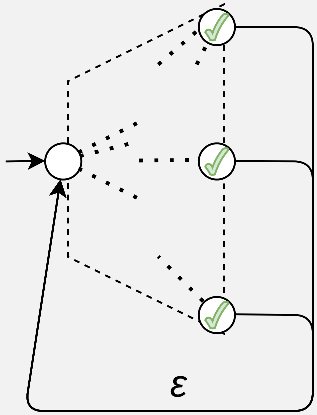
Compiling RegExp to NFAε

`r+`



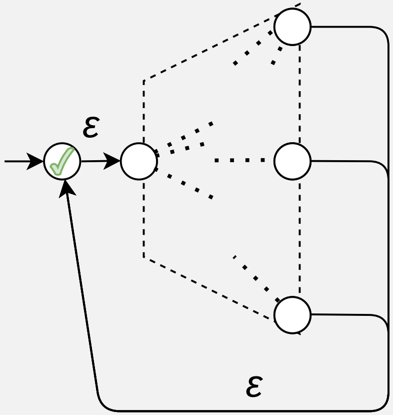
Compiling RegExp to NFAε

`r+`



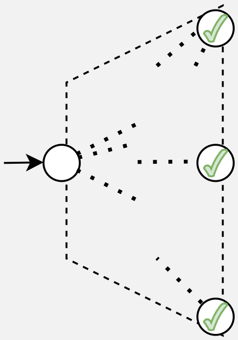
Compiling RegExp to NFAε

`r*`



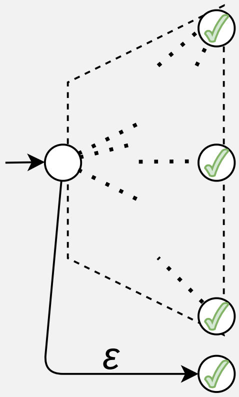
Compiling RegExp to NFAε

`r?`



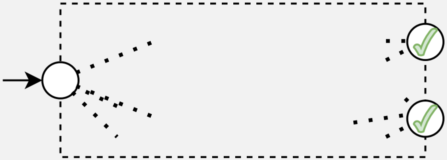
Compiling RegExp to NFAε

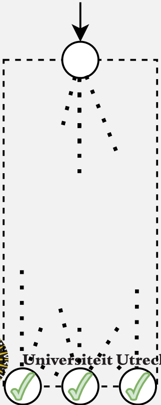
`r?`



Compiling RegExp to NFAε

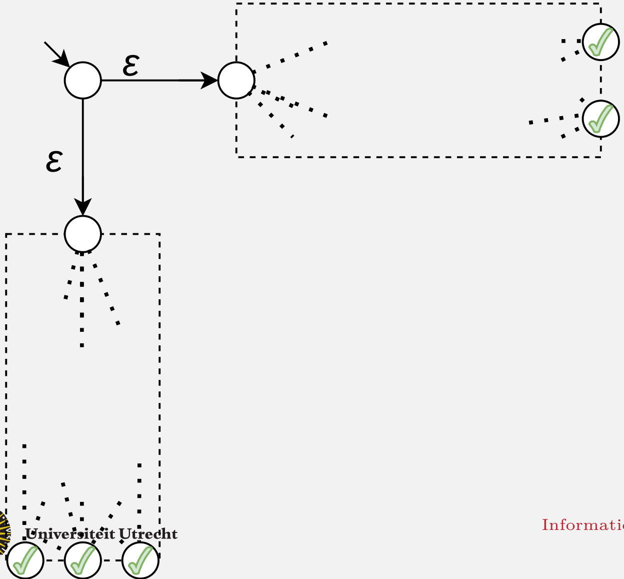
`r`$_1$|`r`$_2$





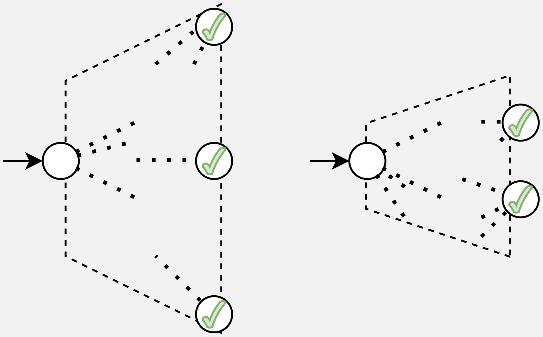
Compiling RegExp to NFAε

$r_1 | r_2$



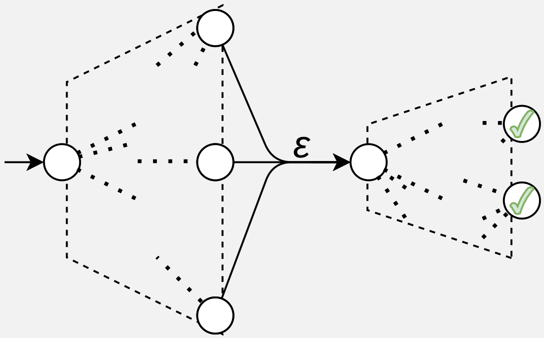
Compiling RegExp to NFAε

$r_1 r_2$



Compiling RegExp to NFAε

r_1r_2



Compiling RegExp to NFAε progress

- ✅ `c`
- ✅ `\d`
- ✅ `[x-z]`
- ✅ $\texttt{r}_1\texttt{r}_2$
- ✅ $\texttt{r}_1|\texttt{r}_2$
- ✅ `r+`
- ✅ `r*`
- ✅ `r?`

Running an NFAε

```
runDFA :: DFA symbol state → [symbol] → state
runDFA = runMoore
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε = ???
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

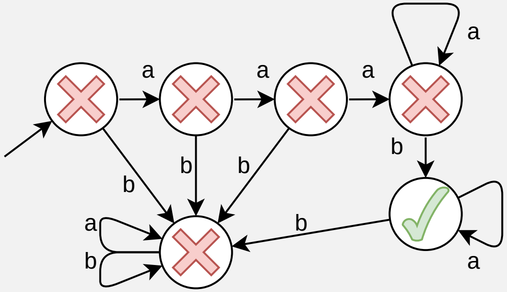
```
a*aaaba*
```

Universiteit Utrecht

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*



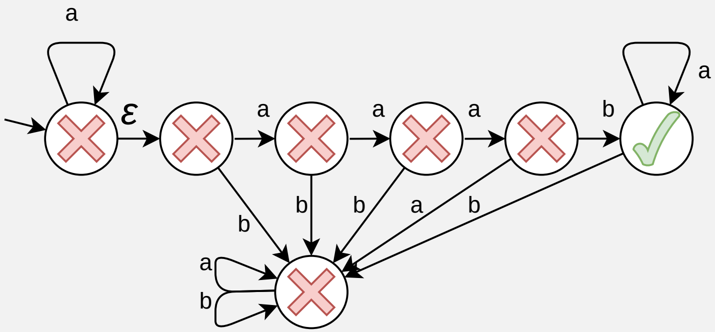
Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*



Universiteit Utrecht

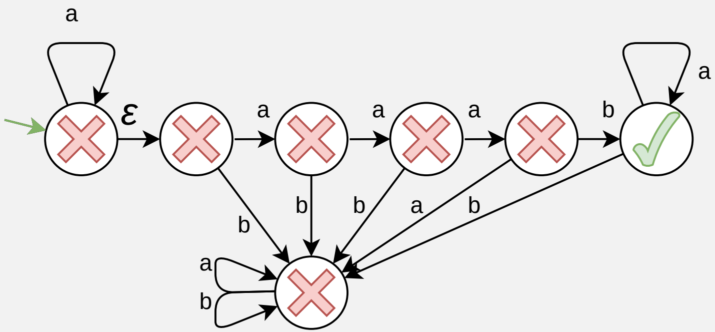
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

```
a*aaaba*
```

```
aaaaab
```

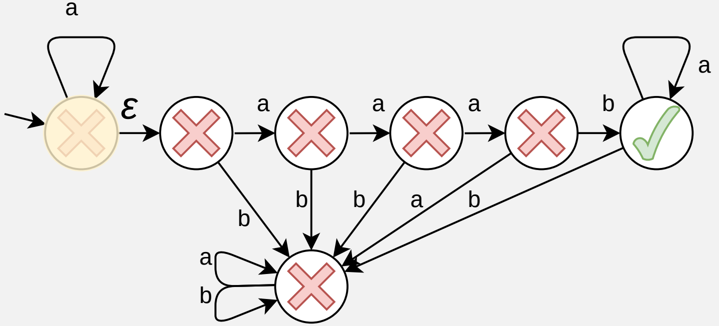


Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

]aaaaab

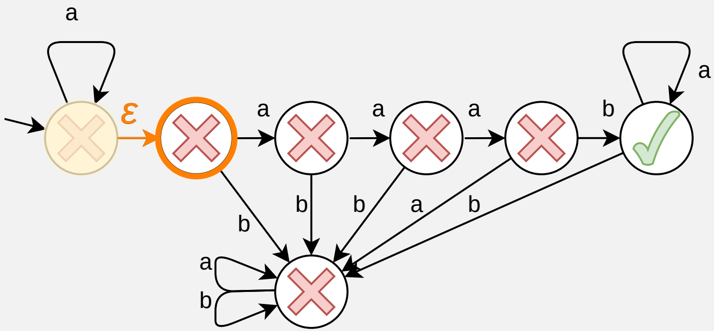


Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

!aaaaab



Universiteit Utrecht

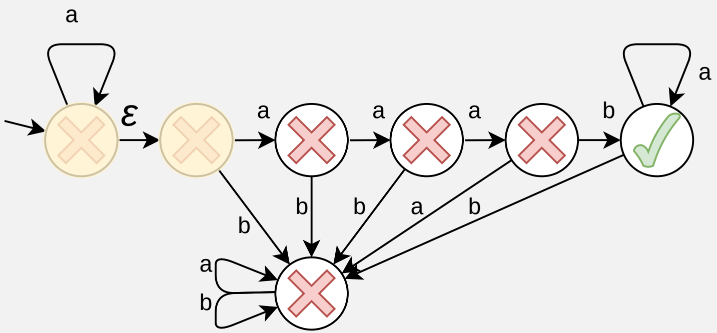
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

|aaaaab



Universiteit Utrecht

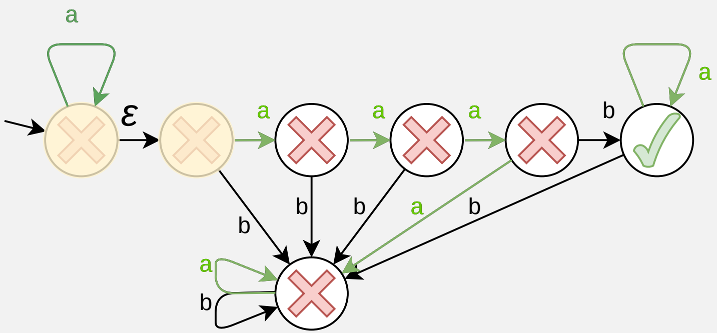
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

[aaaaab

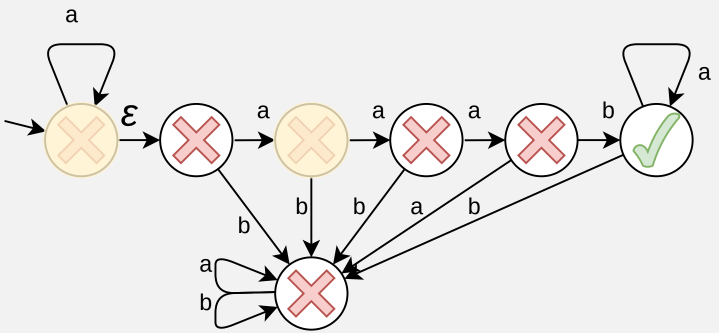


Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

a]aaaab



Universiteit Utrecht

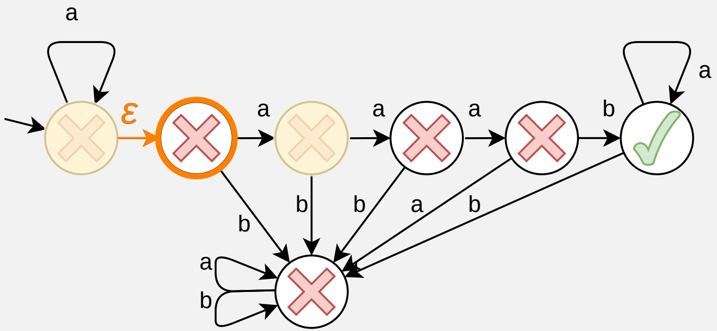
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

a!aaaab



Universiteit Utrecht

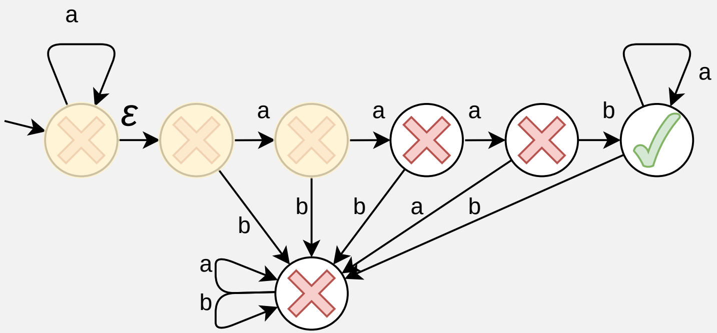
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

a|aaaab



Universiteit Utrecht

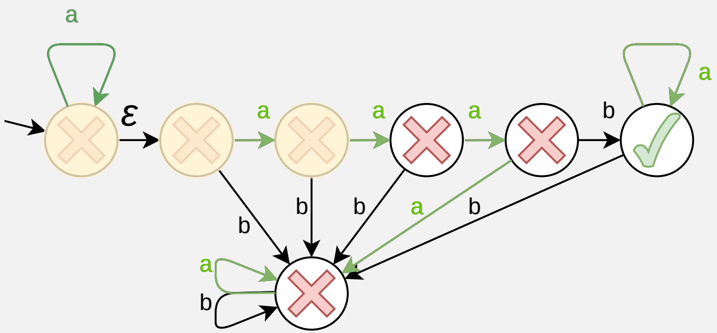
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

a[aaaab

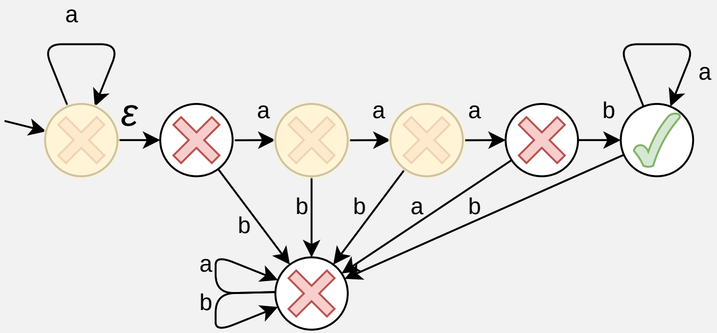


Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aa]aaab

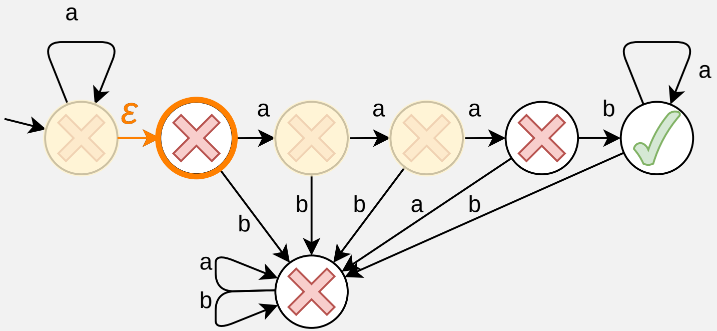


Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aa!aaab



Universiteit Utrecht

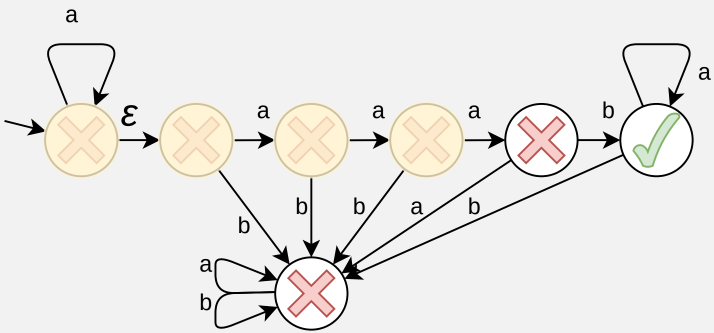
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aa|aaab



Universiteit Utrecht

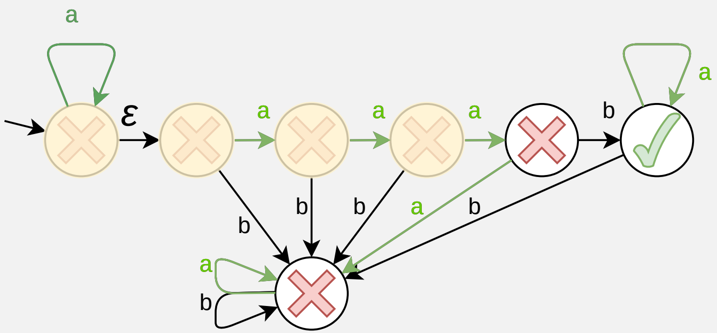
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aa[aaab

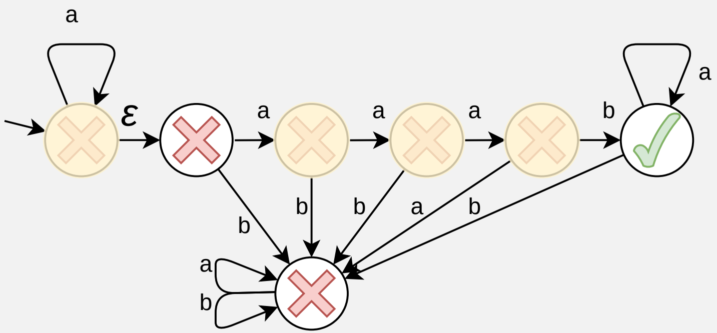


Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaa]aab

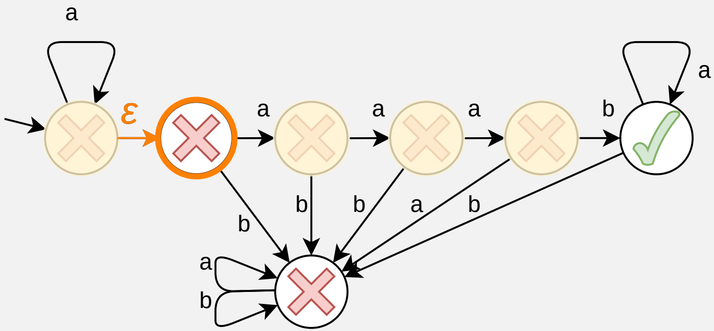


Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

```
a*aaaba*
```

```
aaa!aab
```

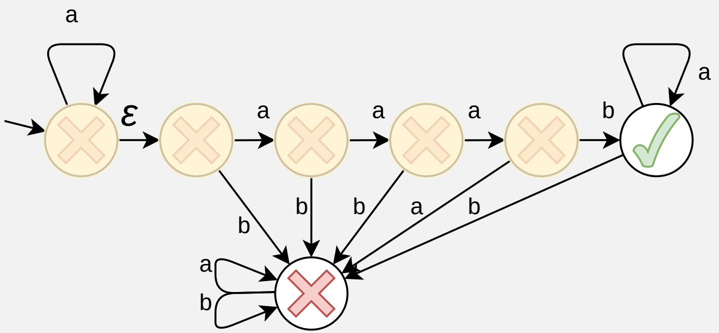


Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaa|aab



Universiteit Utrecht

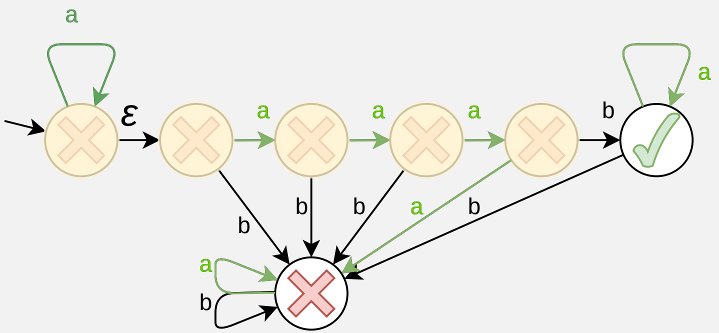
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaa[aab



Universiteit Utrecht

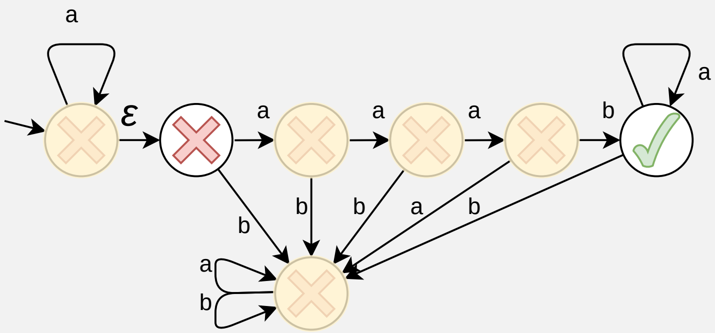
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaaa]ab



Universiteit Utrecht

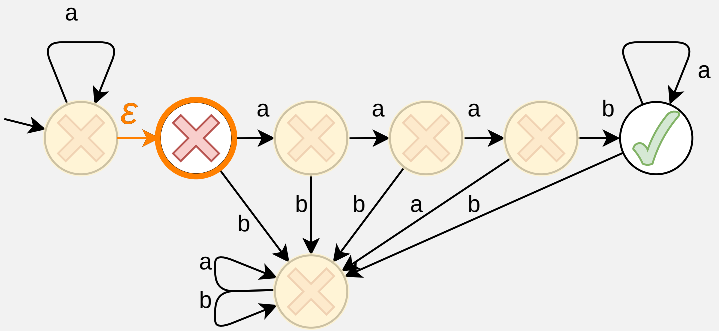
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

```
a*aaaba*

aaaa!ab
```



Universiteit Utrecht

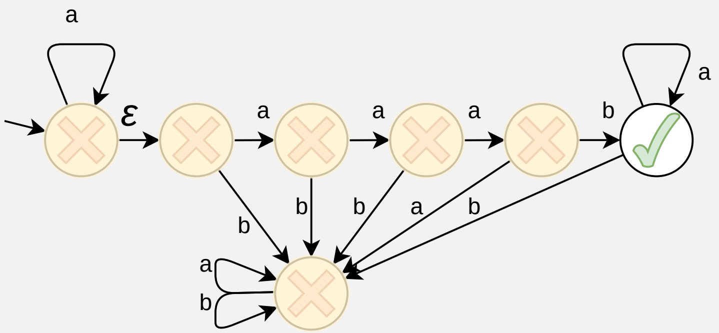
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaaa|ab



Universiteit Utrecht

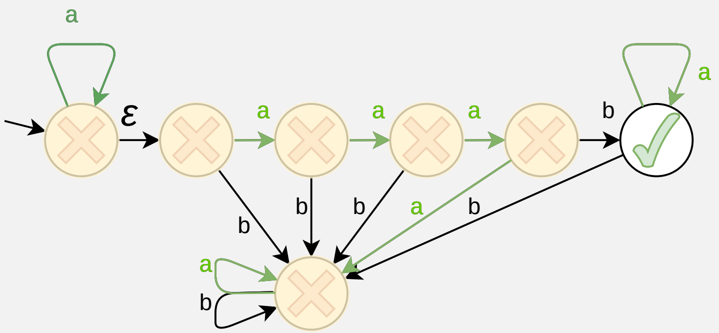
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaaa[ab

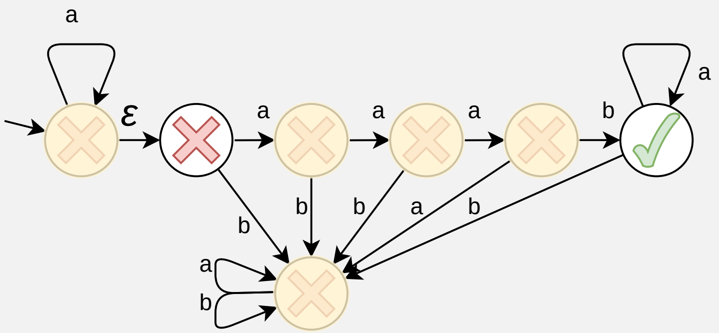


Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaaaa]b



Universiteit Utrecht

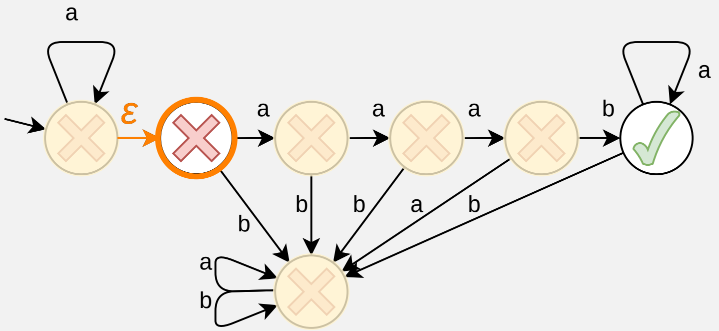
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaaaa!b

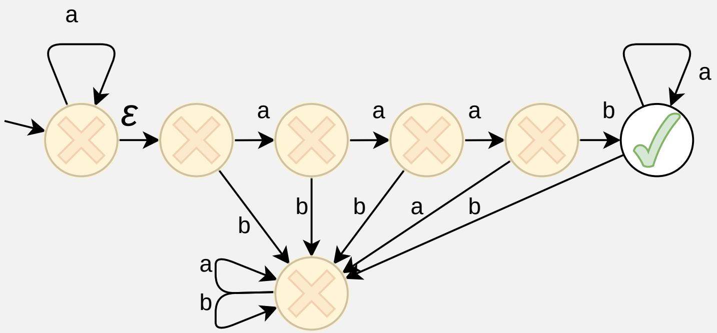


Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaaaa|b



Universiteit Utrecht

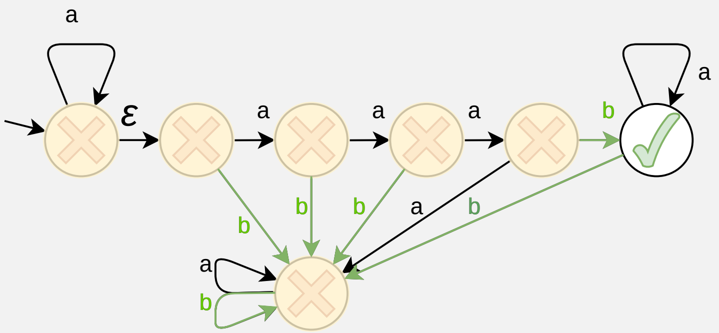
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaaaa[b



Universiteit Utrecht

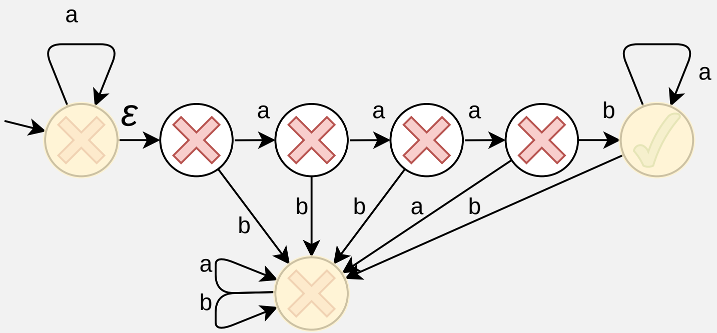
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaaaab]



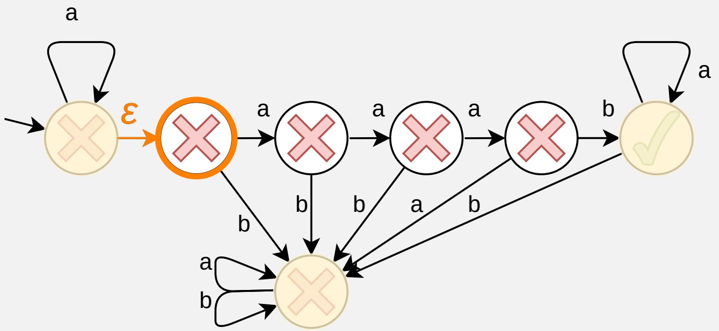
Universiteit Utrecht

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaaaab!

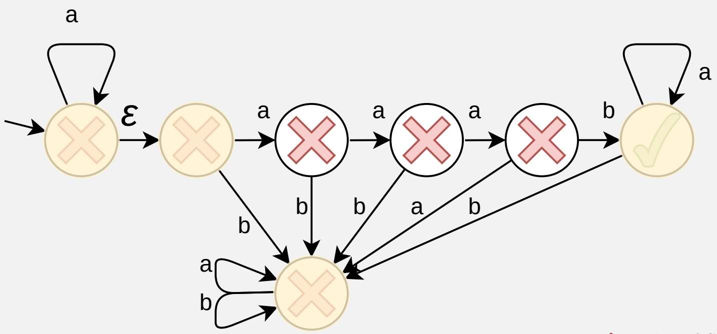


Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) = ???
```

a*aaaba*

aaaaab|



Universiteit Utrecht

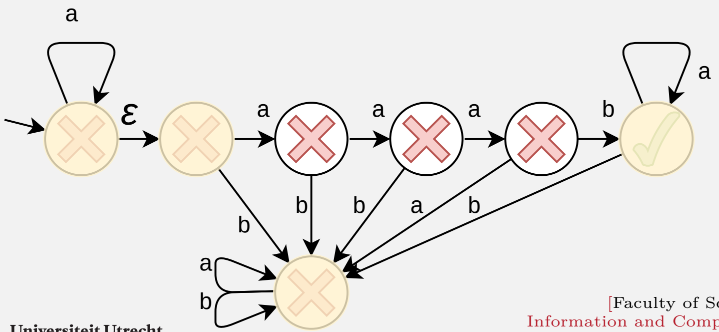
[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr ???₁ ???₂
```

a*aaaba*

aaaaab|



Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

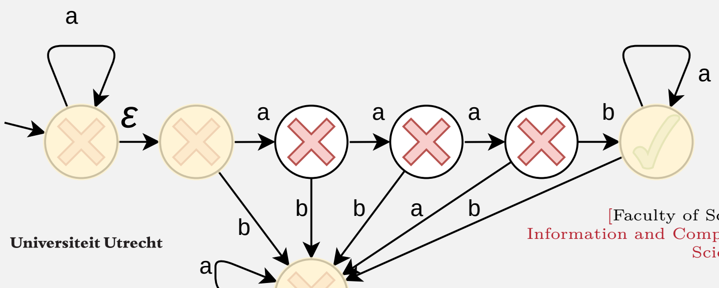
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr ???₁ ???₂
```

```
???₁ :: Set state
???₂ :: symbol → Set state → Set state
```

a*aaaba*

aaaaab|



Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr ???_1 ???_2
```

```
???_1 :: Set state👉
???_2 :: symbol → Set state → Set state
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr ???₁ ???₂
```

```
???₁ :: Set state
???₂ :: symbol → Set state → Set state
```

```
runNFAε nfa [] == -- states reachable without input
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr ???₁ ???₂
```

```
???₁ :: Set state
???₂ :: symbol → Set state → Set state
```

```
runNFAε nfa [] == -- states reachable without input
runNFAε nfa [] == foldr ???₁ ???₂ []
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr ???₁ ???₂
```

```
???₁ :: Set state
???₂ :: symbol → Set state → Set state
```

```
runNFAε nfa [] == -- states reachable without input
runNFAε nfa [] == ???₁
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr ???₁ ???₂
```

```
???₁ :: Set state
???₂ :: symbol → Set state → Set state
```

```
???₁ == -- states reachable without input
```

Running an NFAε

```haskell
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr ???₁ ???₂
```

```haskell
???₁ :: Set state
???₂ :: symbol → Set state → Set state
```

```haskell
???₁ == -- states reachable by ε-transitions only
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr ???₁ ???₂
```

```
???₁ :: Set state
???₂ :: symbol → Set state → Set state
```

```
???₁ == -- states reachable by 0 or more ε-transitions
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr ???₁ ???₂
```

```
???₁ :: Set state
???₂ :: symbol → Set state → Set state
```

```
???₁ == reachable εsteps ???₃
reachable :: Set (state,state) → state → Set state
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr ???₁ ???₂
```

```
???₁ :: Set state
???₂ :: symbol → Set state → Set state
```

```
???₁ == reachable εsteps (s0 nfa)
reachable :: Set (state,state) → state → Set state
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa)) ???₂
```

```
???₂ :: symbol → Set state → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa)) ???₂
```

```
???₂ :: symbol → Set state → Set state👉
```

```
reachable :: Set (state,state) → state → Set state
```

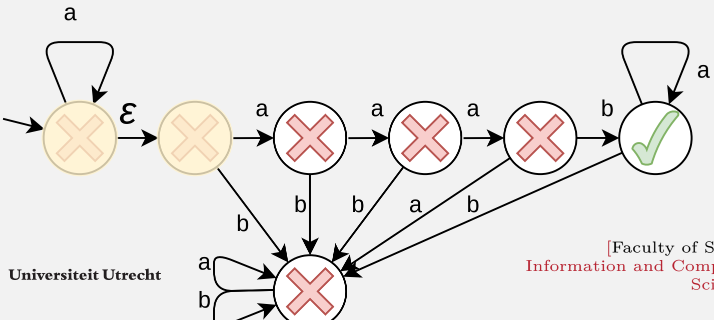
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa)) ???₂
```

```
???₂ :: symbol → Set state → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

```
|aaaaab
```



Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

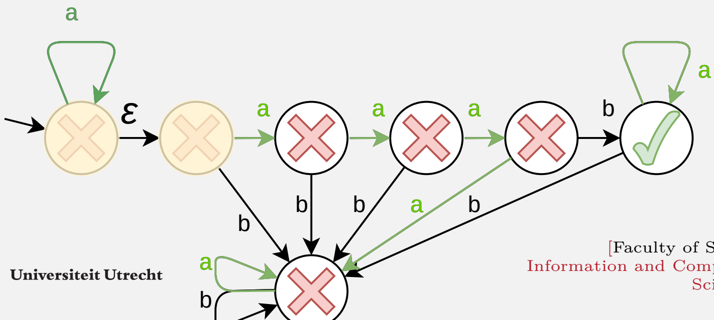
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa)) ???₂
```

```
???₂ :: symbol → Set state → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

```
[aaaaab
```



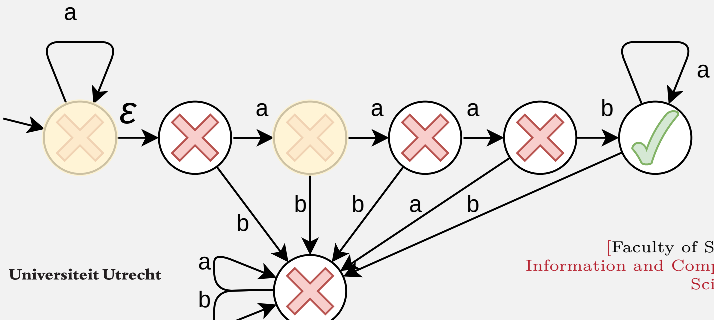
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa)) ???₂
```

```
???₂ :: symbol → Set state → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

a]aaaab



Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

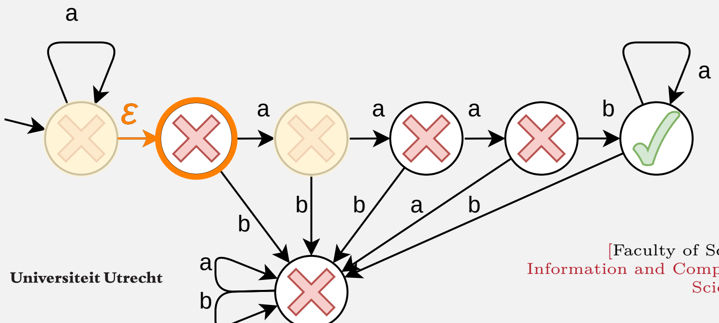
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa)) ???₂
```

```
???₂ :: symbol → Set state → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

```
a!aaaab
```



Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

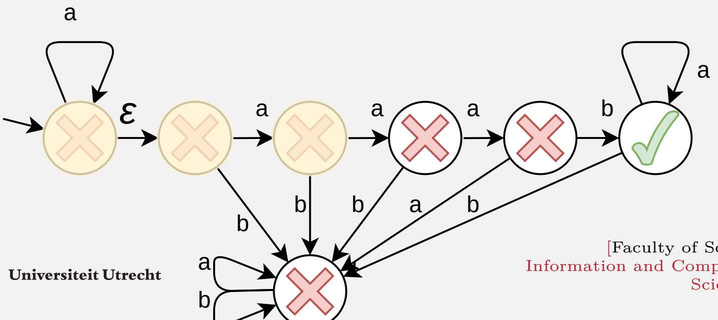
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa)) ???₂
```

```
???₂ :: symbol → Set state → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

a|aaaab



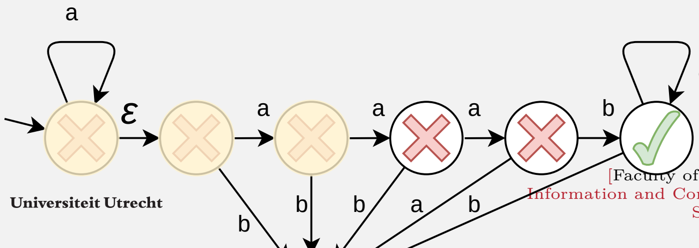
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa)) ???₂
```

```
???₂ = \sy → ???₅ . ???₄
???₄ :: Set state → Set state
???₅ :: Set state → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

```
a|aaaab
```



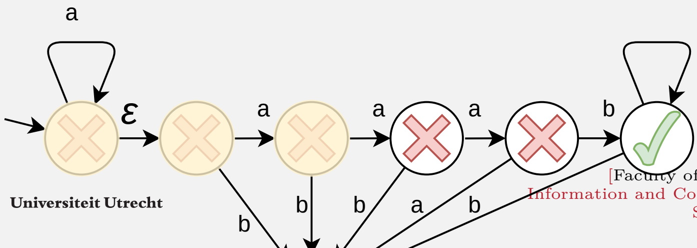
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa))
  (\sy → ???₅ . ???₄ )
```

```
???₄ :: Set state → Set state
???₅ :: Set state → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

a|aaaab



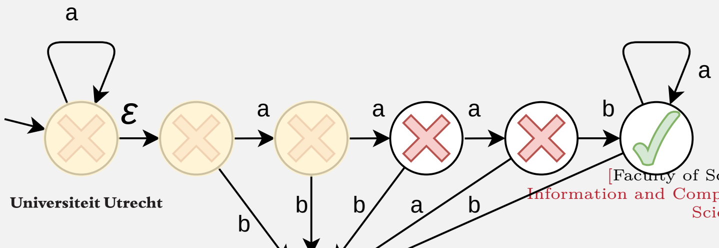
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa))
  (\sy → ???₅ . ???₄ )
```

```
???₄ :: Set state → Set state👈
???₅ :: Set state → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

a|aaaab



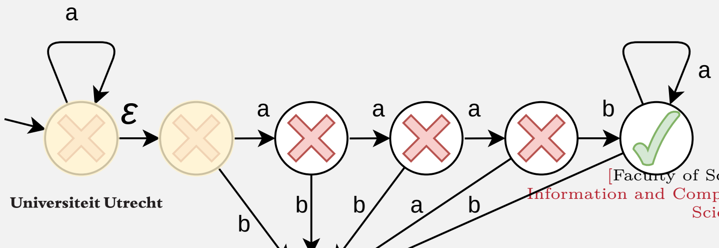
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa))
  (\sy → ???₅ . ???₄ )
```

```
???₄ :: Set state → Set state👉
???₅ :: Set state → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

```
|aaaaab
```



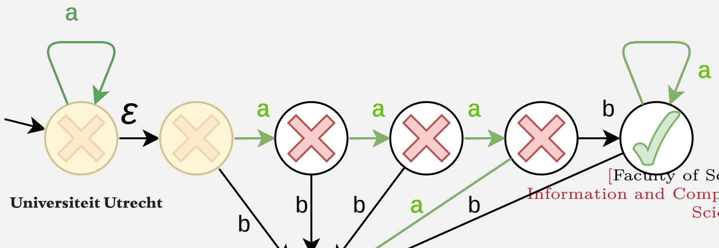
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa))
  (\sy → ???₅ . ???₄ )
```

```
???₄ :: Set state → Set state👉
???₅ :: Set state → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

```
[aaaaab
```



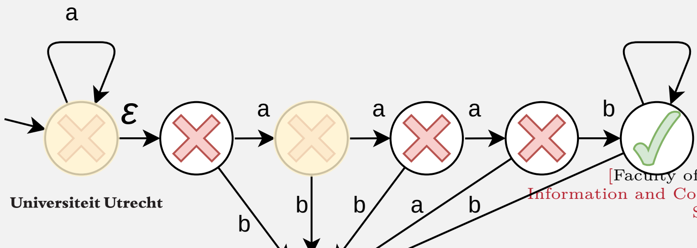
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa))
  (\sy → ???₅ . ???₄ )
```

```
???₄ = Set.map (step nfa sy)
???₅ :: Set state → Set state👉
```

```
reachable :: Set (state,state) → state → Set state
```

```
a]aaaab
```



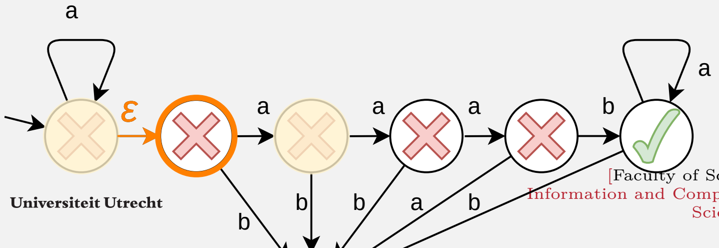
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa))
  (\sy → ???₅ . ???₄ )
```

```
???₄ = Set.map (step nfa sy)
???₅ :: Set state → Set state👉
```

```
reachable :: Set (state,state) → state → Set state
```

```
a!aaaab
```



Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

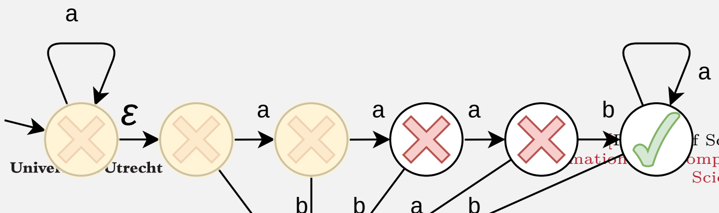
Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa))
  (\sy → ???₅ . ???₄ )
```

```
???₄ = Set.map (step nfa sy)
???₅ = ???₆ . Set.map (reachable εsteps)
???₆ :: Set (Set state) → Set state
```

```
reachable :: Set (state,state) → state → Set state
```

a|aaaab



Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa))
  (\sy → ???₅ . ???₄ )
```

```
???₄ = Set.map (step nfa sy)
???₅ = ???₆ . Set.map (reachable εsteps)
???₆ :: Set (Set state) → Set state

reachable :: Set (state,state) → state → Set state
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa))
  (\sy → ???₅ . ???₄ )
```

```
???₄ = Set.map (step nfa sy)
???₅ = ???₆ . Set.map (reachable εsteps)
???₆ = Set.unions
```

```
reachable :: Set (state,state) → state → Set state
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
  foldr (reachable εsteps (s0 nfa))
  (\sy → ???5 . ???4 )
```

```
???4 = Set.map (step nfa sy)
???5 = Set.unions . Set.map (reachable εsteps)
```

```
reachable :: Set (state,state) → state → Set state
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
    foldr (reachable εsteps (s0 nfa))
  (\sy → ???₅ . Set.map (step nfa sy))
```

```
???₅ = Set.unions . Set.map (reachable εsteps)
```

```
reachable :: Set (state,state) → state → Set state
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
    foldr (reachable εsteps (s0 nfa))
  (\sy → (Set.unions . Set.map (reachable εsteps))
     . Set.map (step nfa sy))
```

```
reachable :: Set (state,state) → state → Set state
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
    foldr (reachable εsteps (s0 nfa))
  (\sy → Set.unions . Set.map
    (reachable εsteps . step nfa sy))
```

```
reachable :: Set (state,state) → state → Set state
```

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
    foldr (reachable εsteps (s0 nfa))
  (\sy → Set.unions . Set.map
    (reachable εsteps . step nfa sy))
```

```
reachable :: Set (state,state) → state → Set state
```

Universiteit Utrecht

Running an NFAε

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
    foldr (reachable εsteps (s0 nfa))
   (\sy → Set.unions . Set.map
     (reachable εsteps . step nfa sy))
```

```
reachable :: Set (state,state) → state → Set state👈
```

Exercise

Putting together the parts

✅ `runNFAε :: NFAε sy st → [sy] → Set st`

Putting together the parts

- ✅ `runNFAε :: NFAε sy st → [sy] → Set st`
- ✅ `r2n :: RegExp → NFAε Char Label -- (by example)`

Putting together the parts

✅ `runNFAε :: NFAε sy st → [sy] → Set st`

✅ `r2n :: RegExp → NFAε Char Label -- (by example)`

```
matchesRegExp :: RegExp → String → Bool
matchesRegExp r s = any isAccepting $
  runNFAε (r2n r) s
```

Putting together the parts

✅ `runNFAε :: NFAε sy st → [sy] → Set st`

✅ `r2n :: RegExp → NFAε Char Label -- (by example)`

```
matchesRegExp :: RegExp → String → Bool
matchesRegExp r s = any isAccepting $
  runNFAε (r2n r) s
```

Done?

Putting together the parts

✅ `runNFAε :: NFAε sy st → [sy] → Set st`

✅ `r2n :: RegExp → NFAε Char Label -- (by example)`

```haskell
matchesRegExp :: RegExp → String → Bool
matchesRegExp r s = any isAccepting $
  runNFAε (r2n r) s
```

Done?

- No!

Running an NFAε better

```
runNFAε :: NFAε sy      st  → [sy] → Set st
runDFA  :: DFA  sy      st  → [sy] →     st
```

Running an NFAε better

```
runNFAε :: NFAε sy       st  → [sy] → Set st

runDFA  :: DFA  sy (Set st) → [sy] → Set st
```

Running an NFAε better

```
runNFAε :: NFAε sy       st  → [sy] → Set st

runDFA  :: DFA  sy (Set st) → [sy] → Set st

n2d       :: NFAε sy       st
           → DFA  sy (Set st)
```

Running an NFAε better

```
runNFAε :: NFAε sy       st  → [sy] → Set st

runDFA  :: DFA  sy (Set st) → [sy] → Set st

n2d       :: NFAε sy       st
            → DFA  sy (Set st)

runNFAε = runDFA . n2d
```

The subset construction

```
n2d :: NFAε sy st → DFA sy (Set st)
```

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
n2d = ???
```

Universiteit Utrecht

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = ???₁
  , step = ???₂
  , genOut = ???₃ }
```

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = ???₁
  , step = ???₂
  , genOut = ???₃ }
```

$???_1$ `:: Set state`

$???_2$ `:: symbol → Set state → Set state`

$???_3$ `:: Set state → Bool`

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = ???₁
  , step = ???₂
  , genOut = ???₃ }
```

```
???₁ :: Set state

???₂ :: symbol → Set state → Set state

???₃ :: Set state → Bool

genOut :: state → Bool
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = ???₁
  , step = ???₂
  , genOut = ???₃ }
```

```
???₁ :: Set state

???₂ :: symbol → Set state → Set state

???₃ = any genOut

genOut :: state → Bool
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = ???₁
  , step = ???₂
  , genOut = ???₃ }
```

```
???₁ :: Set state

???₂ :: symbol → Set state → Set state

???₃ = any genOut
```

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = ???₁
  , step = ???₂
  , genOut = any genOut }
```

```
???₁ :: Set state
```

```
???₂ :: symbol → Set state → Set state
```

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = ???₁
  , step = ???₂
  , genOut = any genOut }
```

```
???₁ :: Set state
```

```
???₂ :: symbol → Set state → Set state
```

🕵 Look familiar?

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = ???₁
  , step = ???₂
  , genOut = any genOut }
```

```
???₁ :: Set state

???₂ :: symbol → Set state → Set state
```

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
    foldr ???₁ ???₂
???₁ :: Set state
???₂ :: symbol → Set state → Set state
```

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = ???₁
  , step = ???₂
  , genOut = any genOut }
```

```
???₁ :: Set state
```

```
???₂ :: symbol → Set state → Set state
```

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
    foldr ???₁ ???₂
???₁ = reachable εsteps (s0 nfa)
???₂ = \sy → Set.unions . Set.map
  (reachable εsteps . step nfa sy)
```

Universiteit Utrecht

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = ???₁
  , step = ???₂
  , genOut = any genOut }
```

```
???₁ = reachable εsteps (s0 nfa)
```

```
???₂ :: symbol → Set state → Set state
```

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
    foldr ???₁ ???₂
???₁ = reachable εsteps (s0 nfa)
???₂ = \sy → Set.unions . Set.map
  (reachable εsteps . step nfa sy)
```

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = reachable εsteps (s0 nfa)
  , step = ???₂
  , genOut = any genOut }
```

```
???₂ :: symbol → Set state → Set state
```

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
    foldr ???₁ ???₂
???₁ = reachable εsteps (s0 nfa)
???₂ = \sy → Set.unions . Set.map
  (reachable εsteps . step nfa sy)
```

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = reachable εsteps (s0 nfa)
  , step = ???₂
  , genOut = any genOut }
```

```
???₂ = \sy → Set.unions . Set.map
  (reachable εsteps . step nfa sy)
```

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
    foldr ???₁ ???₂
???₁ = reachable εsteps (s0 nfa)
???₂ = \sy → Set.unions . Set.map
  (reachable εsteps . step nfa sy)
```

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = reachable εsteps (s0 nfa)
  , step = \sy → Set.unions . Set.map
       (reachable εsteps . step nfa sy)
  , genOut = any genOut }
```

```
runNFAε :: NFAε symbol state → [symbol] → Set state
runNFAε (NFAε step εsteps genOut s0) =
    foldr ???₁ ???₂
???₁ = reachable εsteps (s0 nfa)
???₂ = \sy → Set.unions . Set.map
  (reachable εsteps . step nfa sy)
```

The subset construction

```
n2d :: NFAε symbol state → DFA symbol (Set state)
```

```
n2d (NFAε step εsteps genOut s0) = Moore
  { s0 = reachable εsteps (s0 nfa)
  , step = \sy → Set.unions . Set.map
      (reachable εsteps . step nfa sy)
  , genOut = any genOut }
```

Recap

- RegExp $\to$ NFAε
- NFAε $\to$ DFA

Recap

- RegExp → NFAε
- NFAε → DFA
- RegExp 🚀

Recap

- RegExp $\rightarrow$ NFAε
- NFAε $\rightarrow$ DFA
- RegExp 🚀 ❓

Performance

If `n = length input` and `m = length regexp`, then...

- $O(nm)$ time

Performance

If `n = length input` and `m = length regexp`, then...

- $O(nm)$ time
- Our 'new' algorithm is not so hot
  - Invented 1959: doi.org/10.1147/rd.32.0114
  - Compiled 1964: doi.org/10.1145/363347.363387

Performance

If `n = length input` and `m = length regexp`, then...

- $O(nm)$ time
- Our 'new' algorithm is not so hot
  - Invented 1959: doi.org/10.1147/rd.32.0114
  - Compiled 1964: doi.org/10.1145/363347.363387
- Best known algorithm (2009):
  - $O(n)$ space
  - $O(nm\frac{\log\log n}{\log^{\frac{3}{2}} n} + n + m)$ time
  - doi.org/10.1007/978-3-642-02927-1_16

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Summary

- RegExp 🚀
- RegExp $\rightarrow$ NFAε
- NFAε $\rightarrow$ DFA
- DFA $\rightarrow$
- Not the only way
  - DFA product (e.g. $\mathtt{r}_1|\mathtt{r}_2$)
  - “Brzozowski derivative”