Finite State Machines

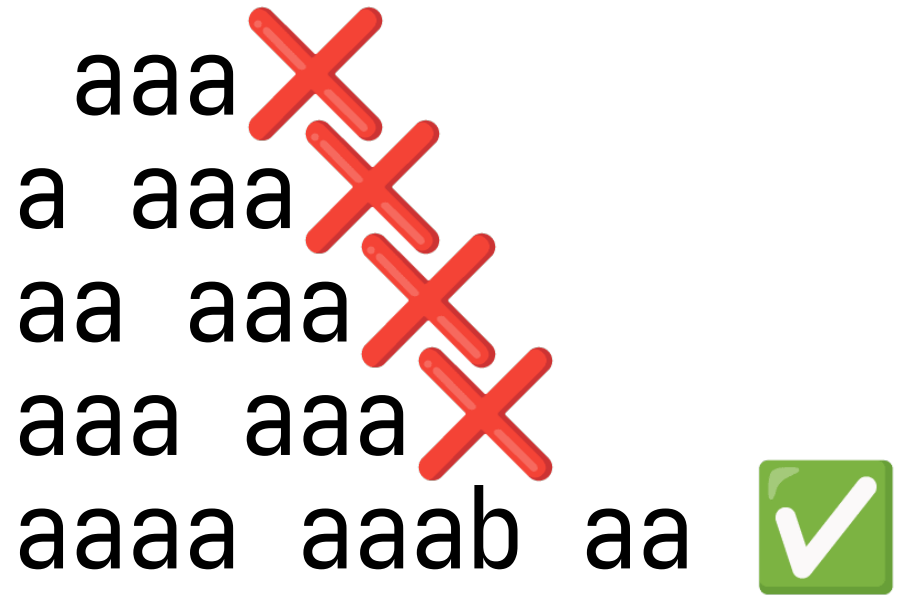
Recap: RegExp

```
<|>  :: R → R → R                  r1|r2
<+>  :: R → R → R                  r1r2
many  :: R → R                     r*
many1 :: R → R                     r+
option  :: R → R                   r?
symbol  :: Char → R                c
satisfy :: (Char → Bool) → R       \d \s \S [a-z] ...
type R = Parser Char String
```

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

Recap: RegExp performance

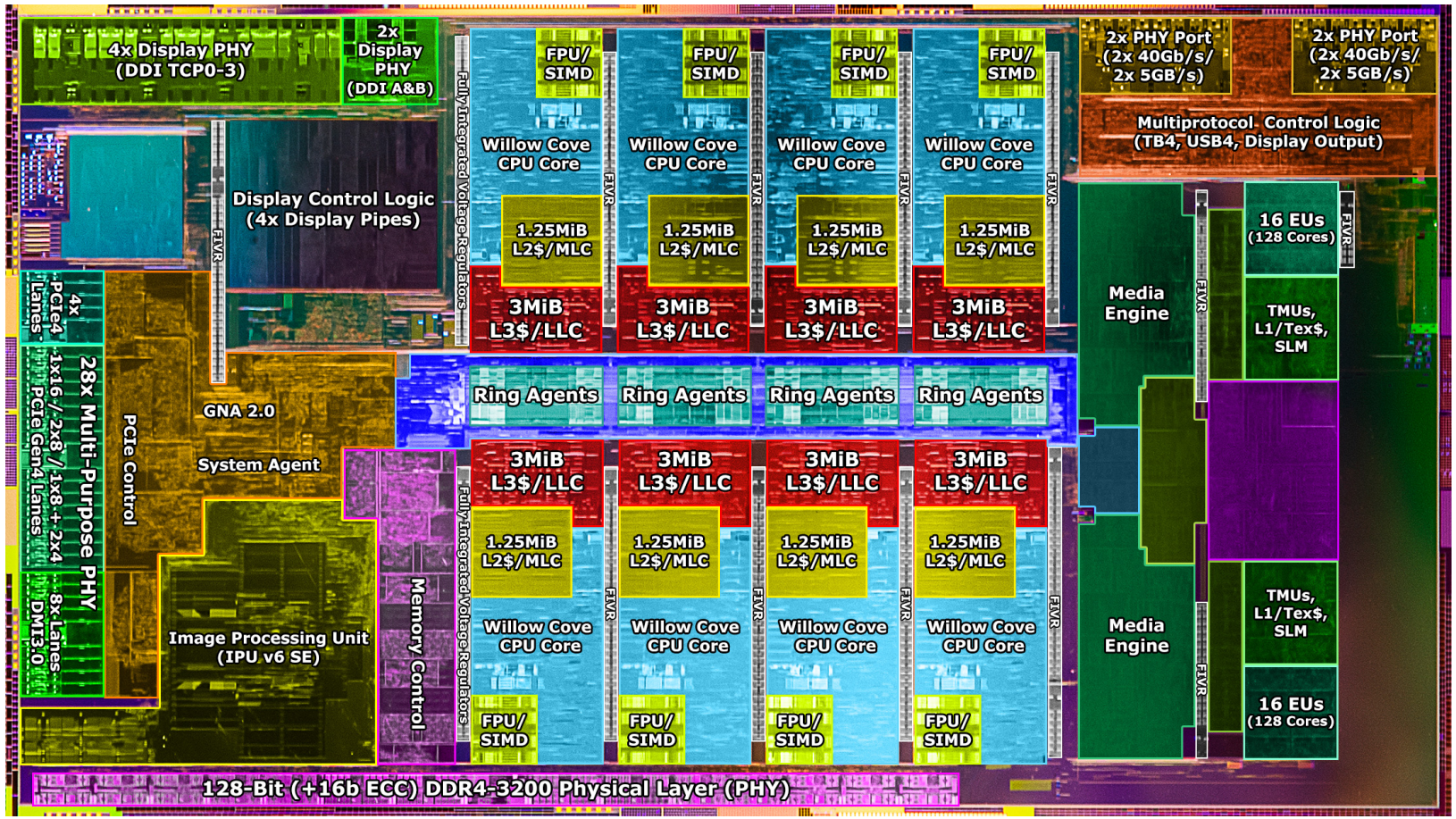
```
head $ matchRegExp "a*aaaba*$" "aaaaaaaabaa
```



Today: matching RegExp *fast*

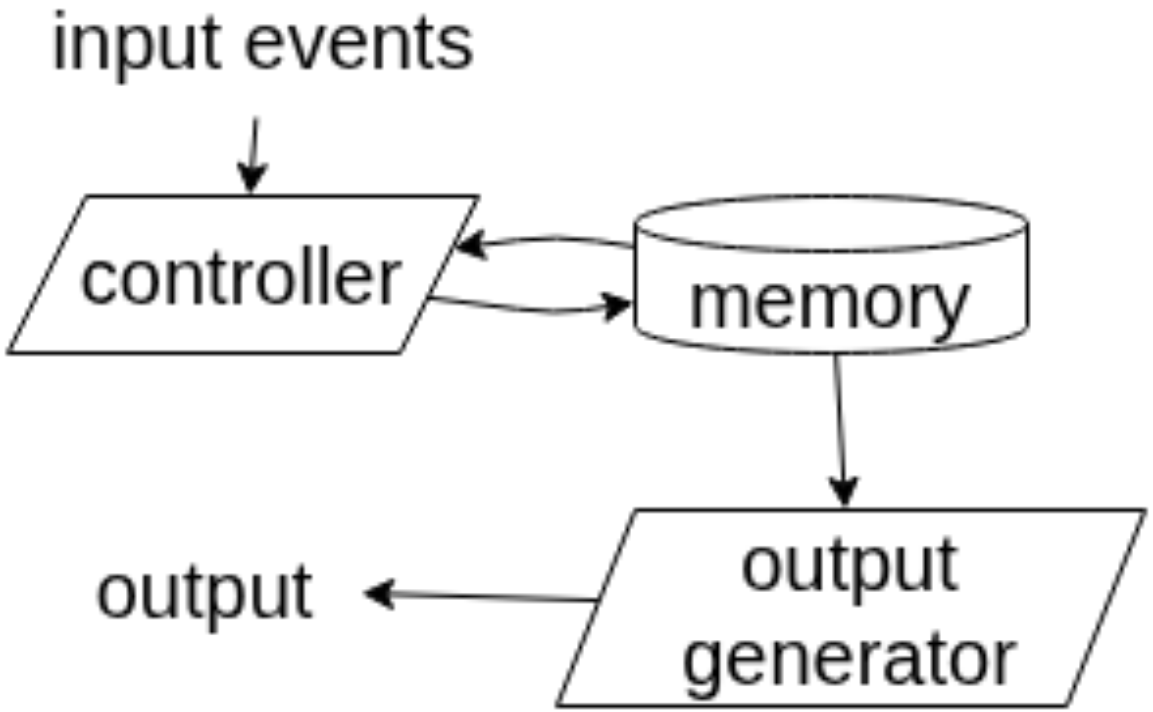
- O(length input * regexp complexity) matching time
- Algorithm from the *bottom up*

Problem: computers are complicated



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

Simplification: Moore Machine



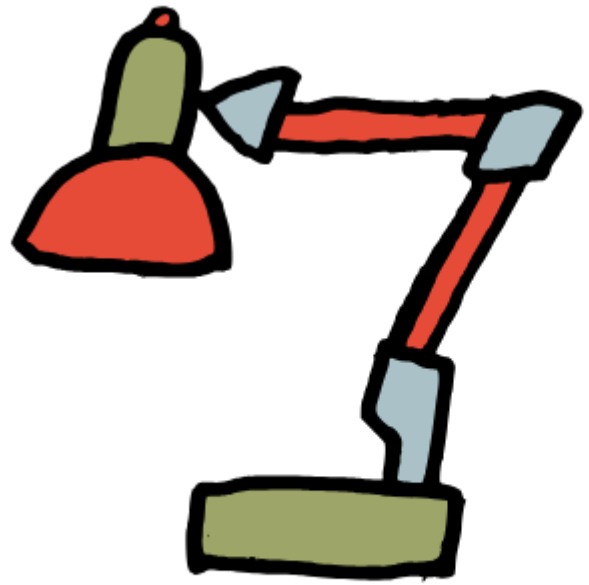
a.k.a. *Finite State Machine* (FSM)

v.s.t. *Finite State Automaton* (FSA)

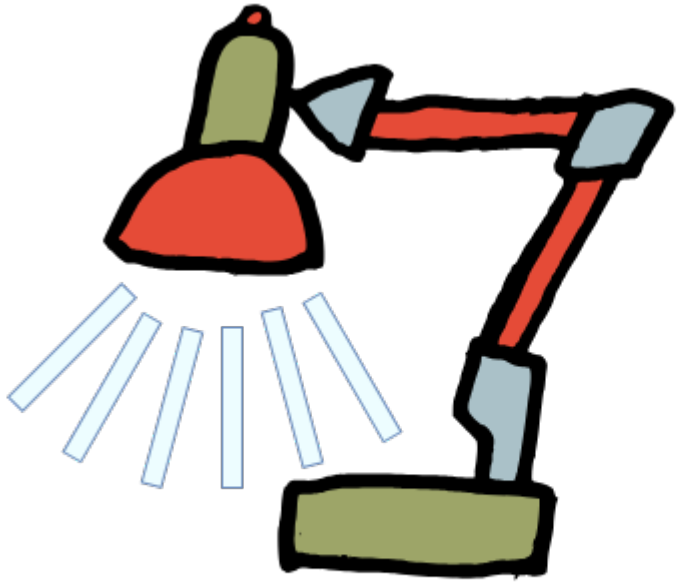
a.k.a. *Deterministic Finite Automaton* (DFA)

Concrete Example

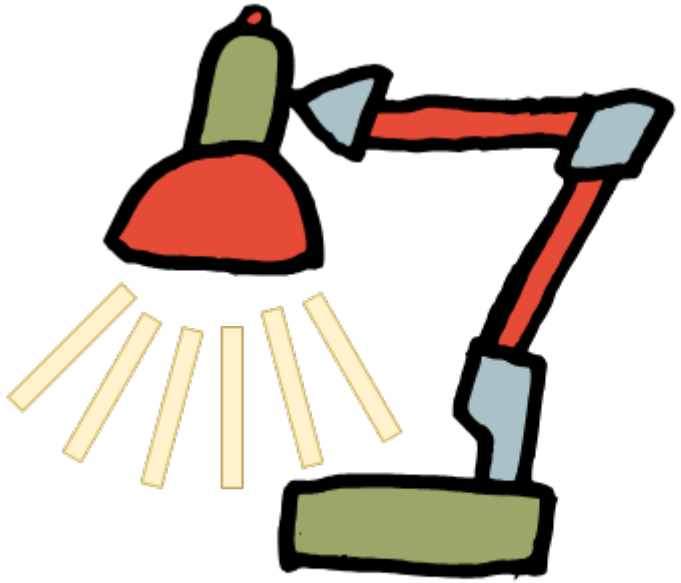
Concrete Example



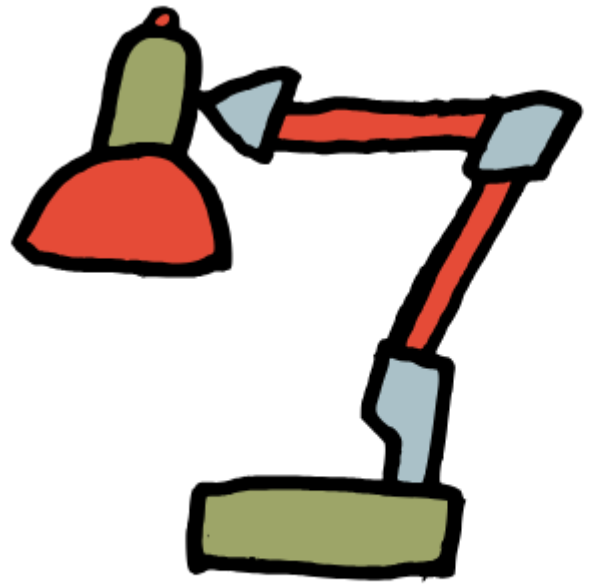
Concrete Example



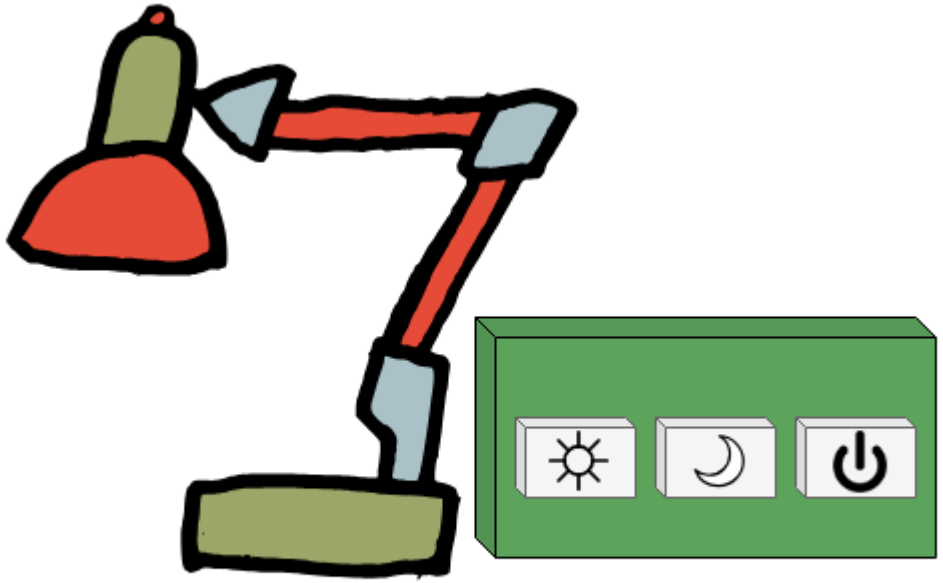
Concrete Example



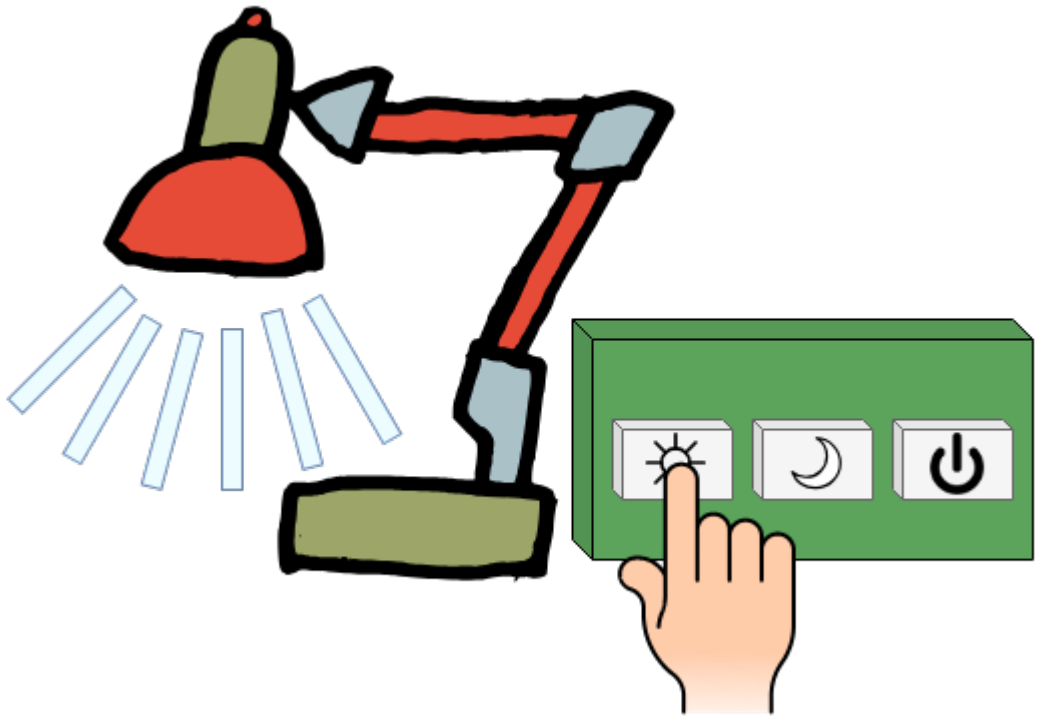
Concrete Example



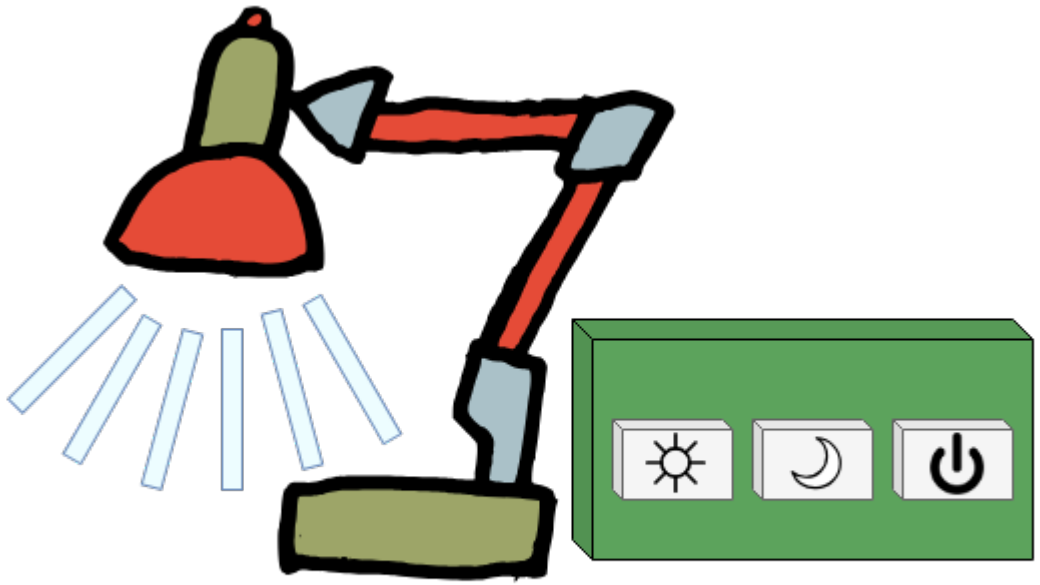
Concrete Example



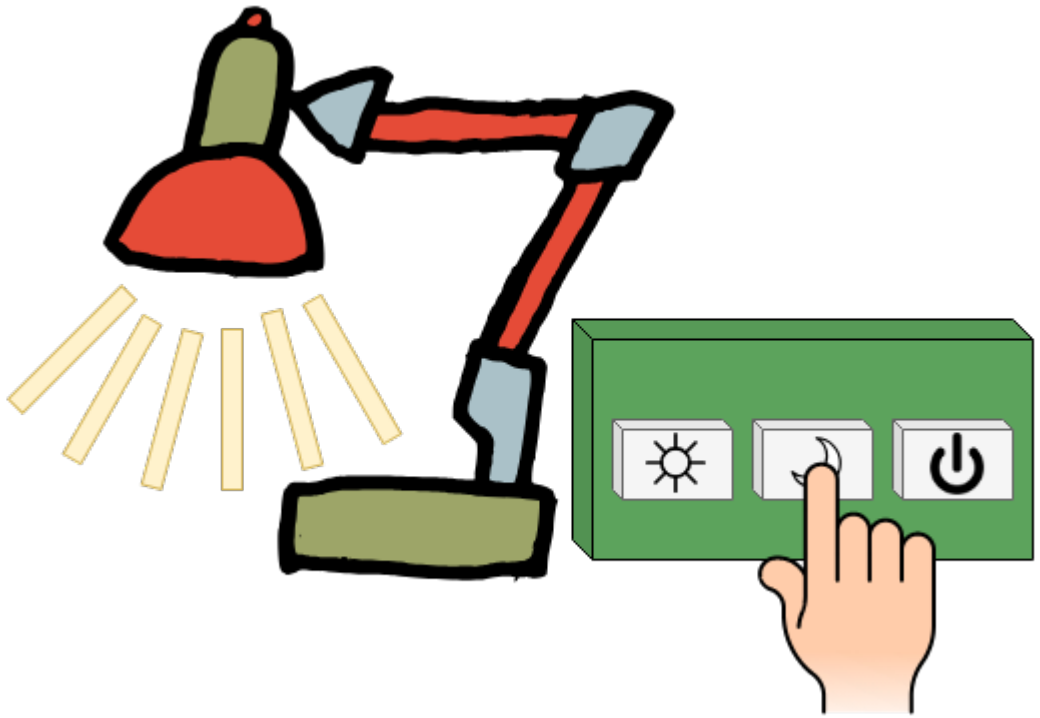
Concrete Example



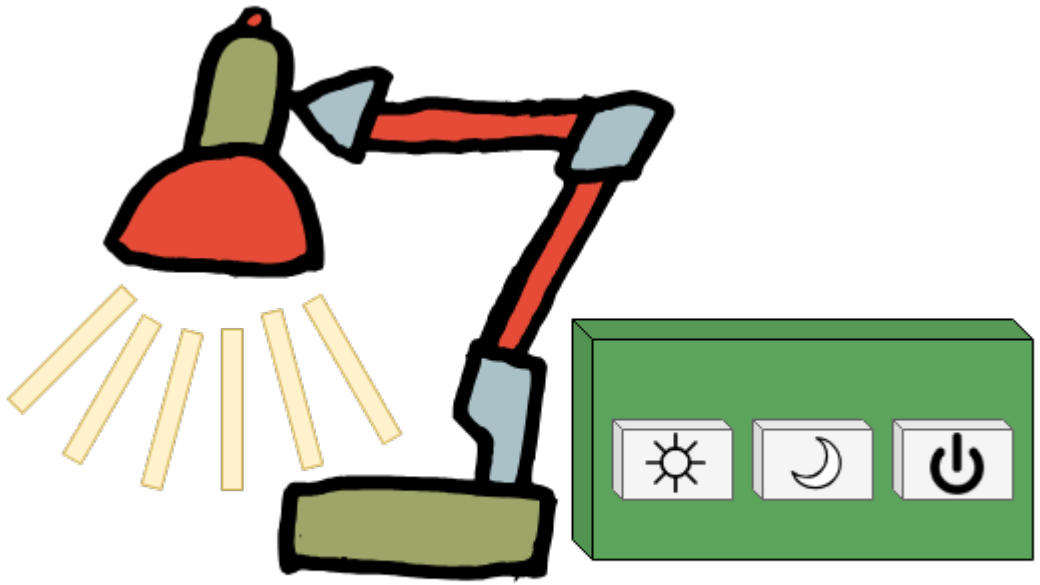
Concrete Example



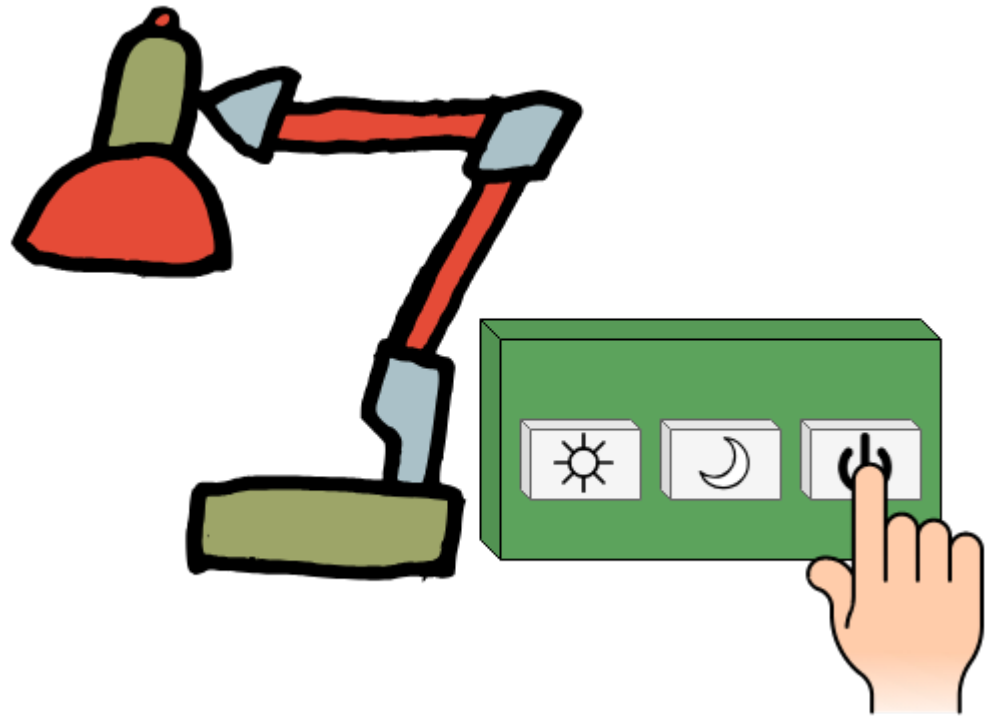
Concrete Example



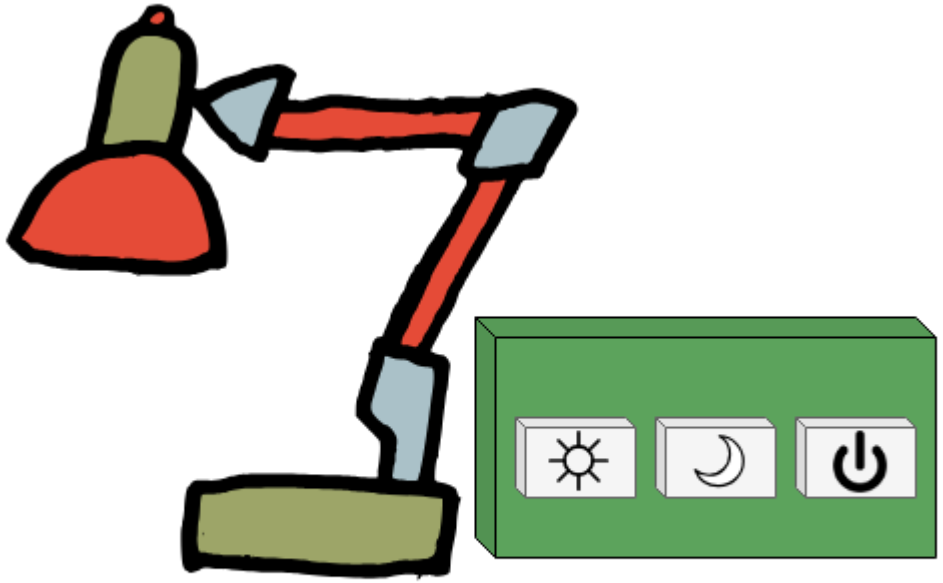
Concrete Example



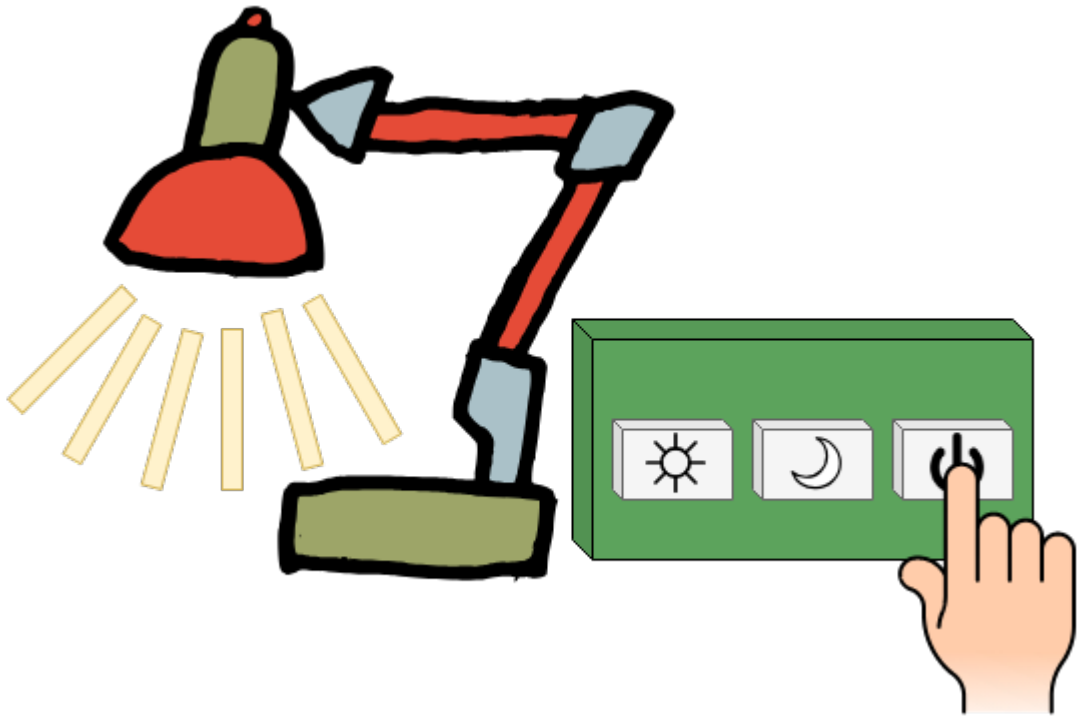
Concrete Example



Concrete Example

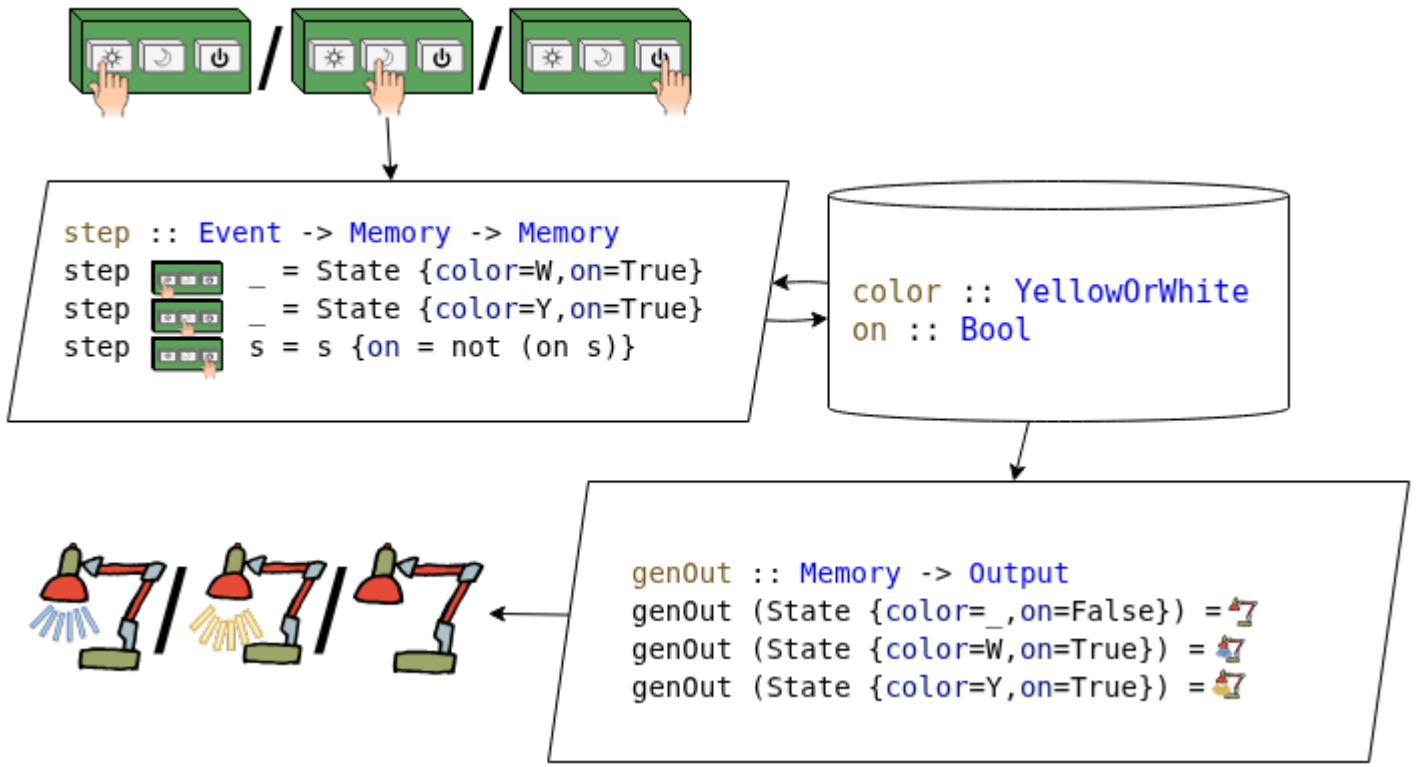


Concrete Example

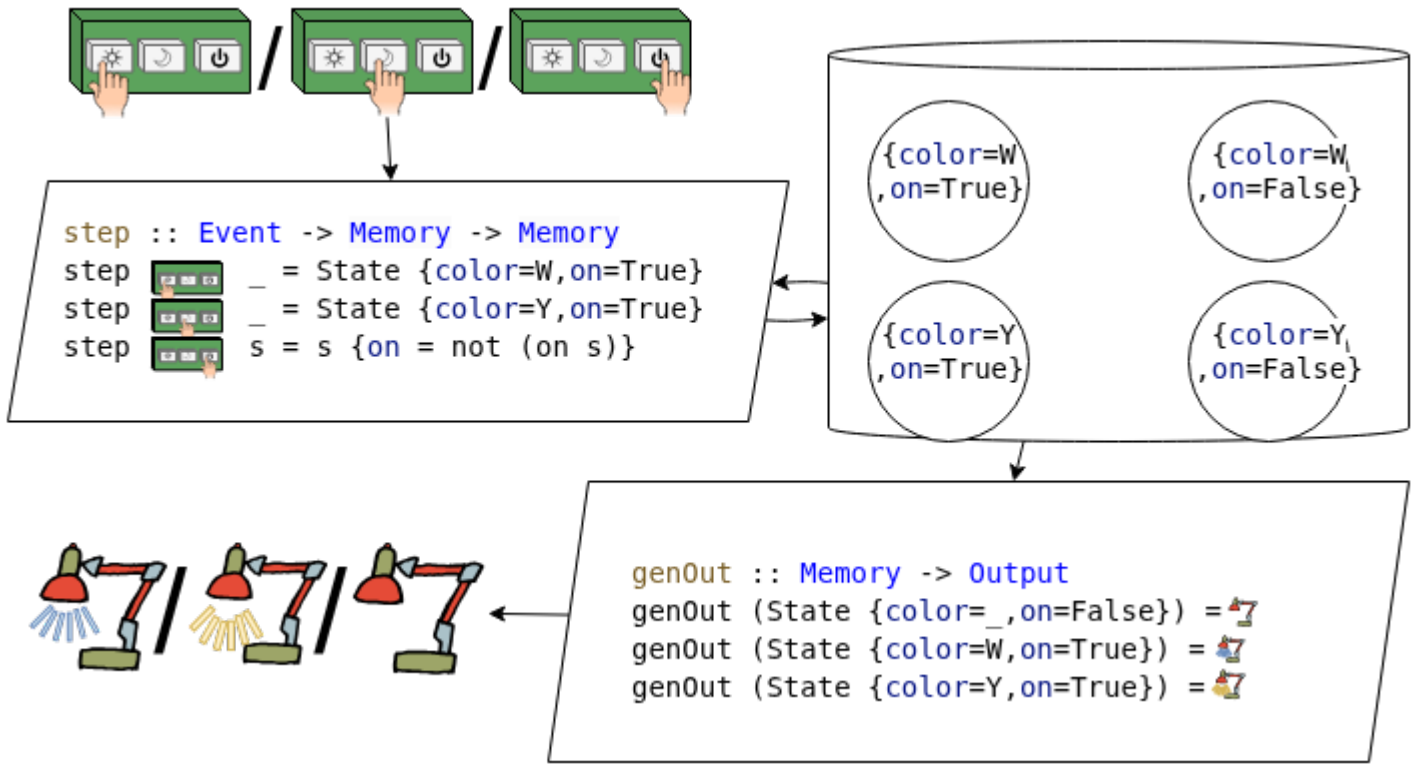


Moore Machine for lamp

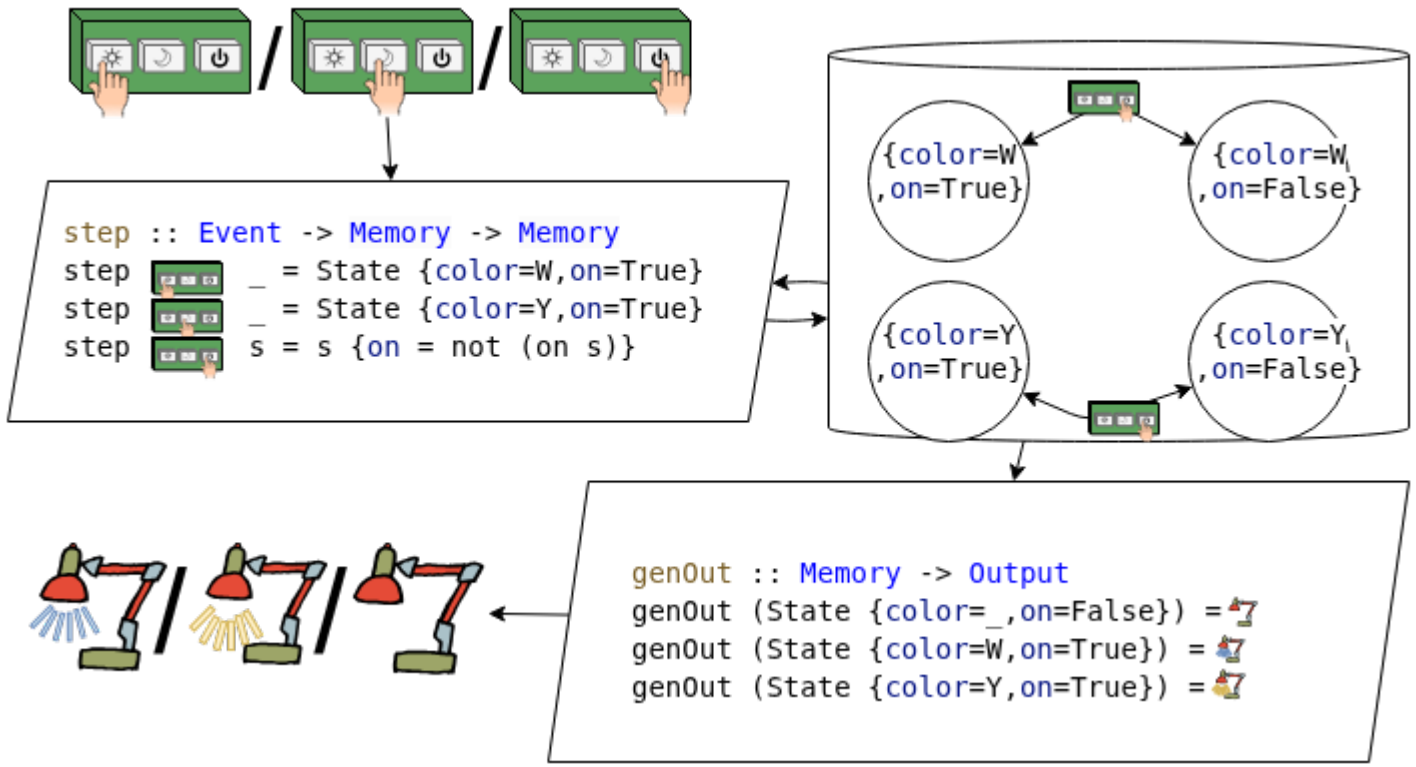
Moore Machine for lamp



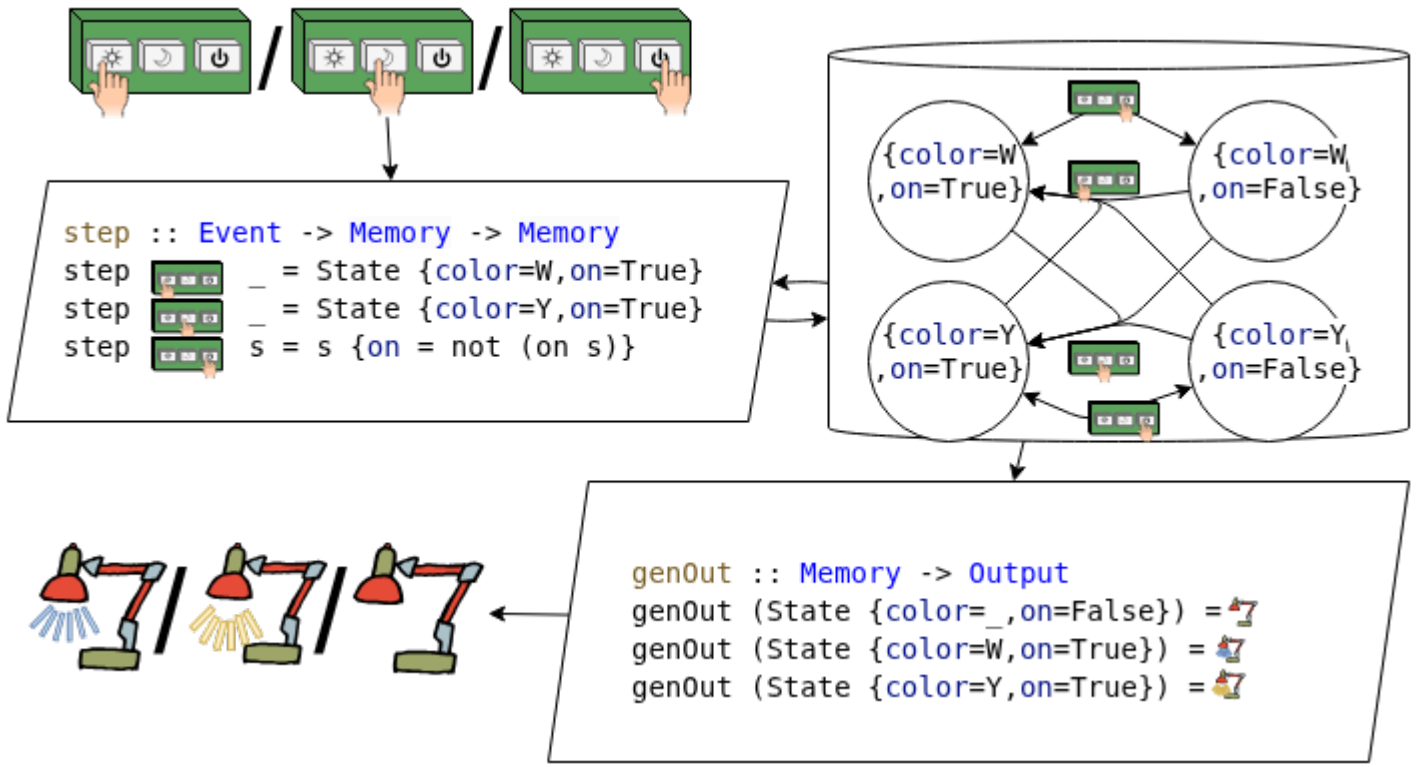
Moore Machine for lamp



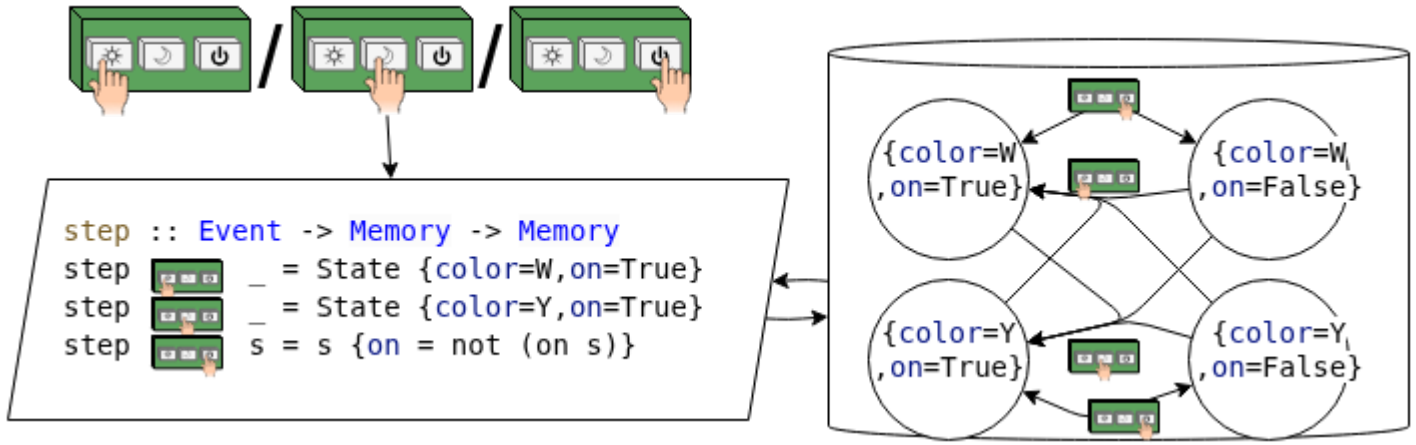
Moore Machine for lamp



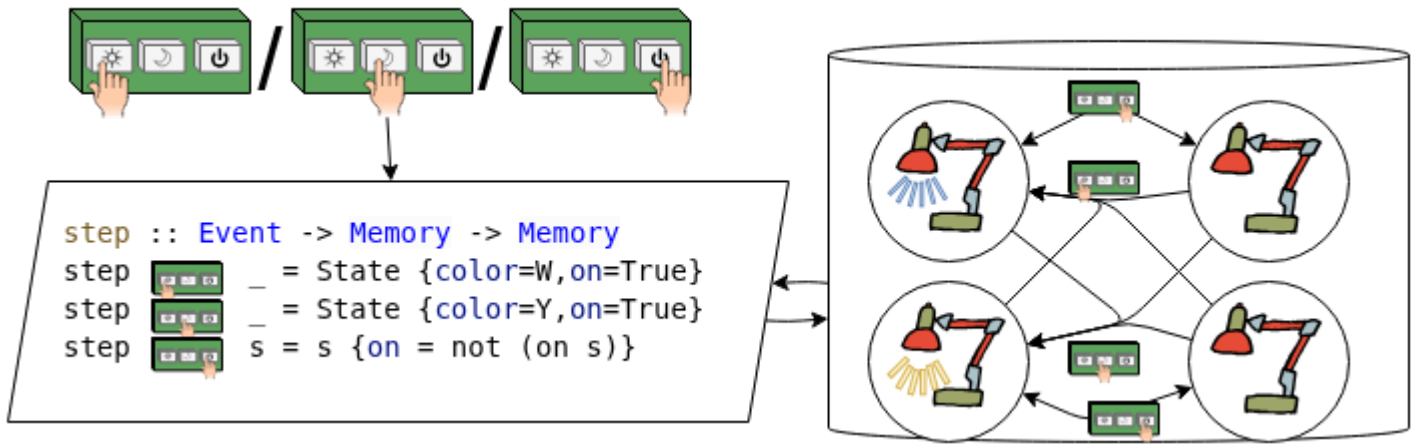
Moore Machine for lamp



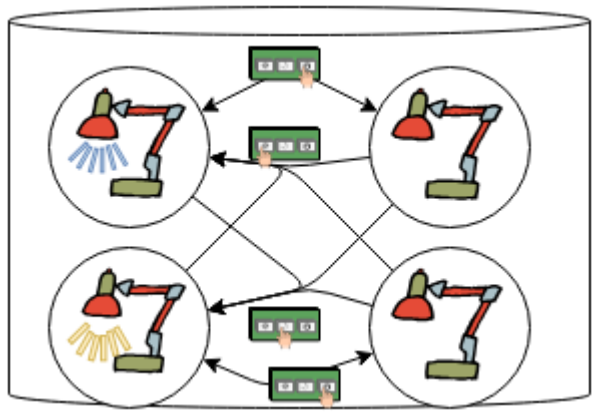
Moore Machine for lamp



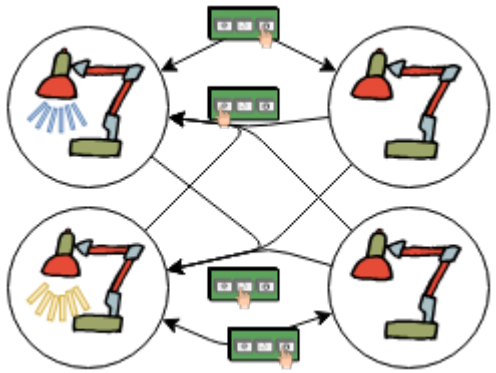
Moore Machine for lamp



Moore Machine for lamp

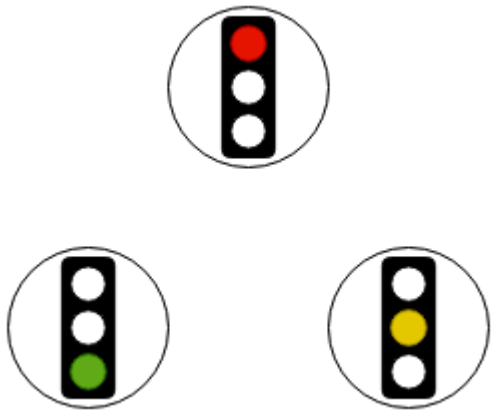


Moore Machine for lamp

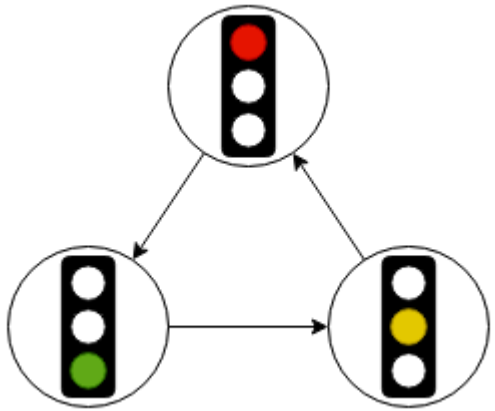


Another Example

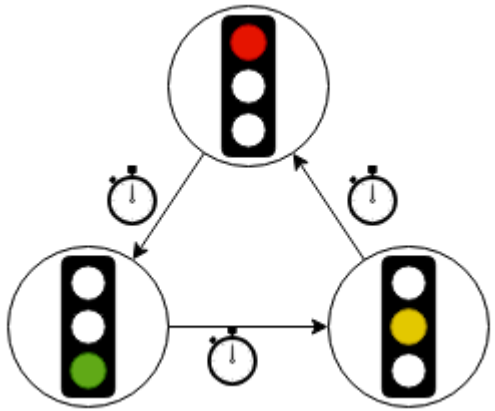
Another Example



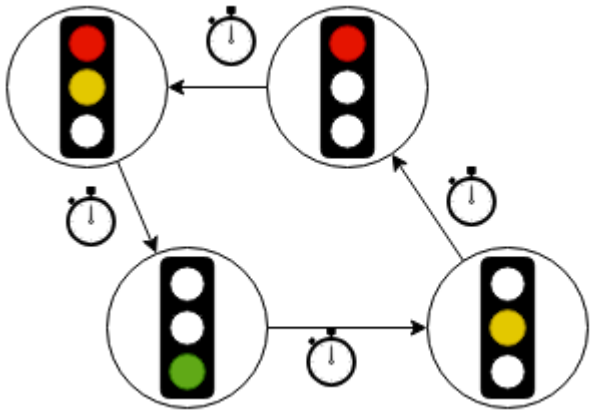
Another Example



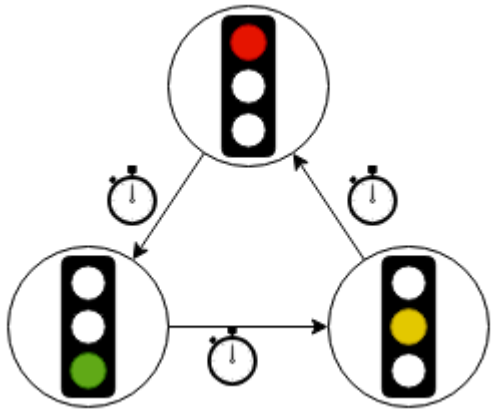
Another Example



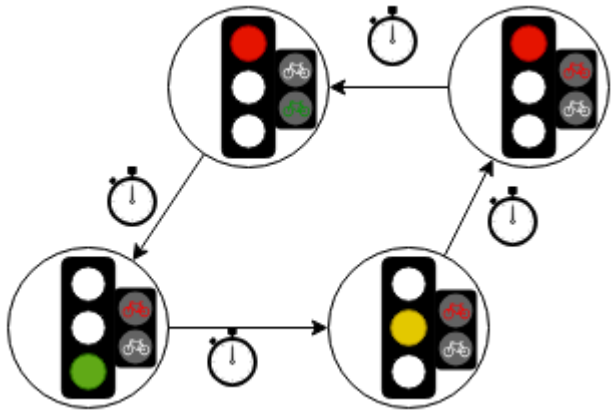
Another Example



Another Example

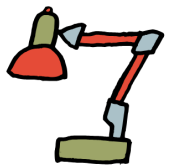


Another Example

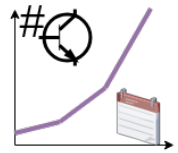


A Big Example

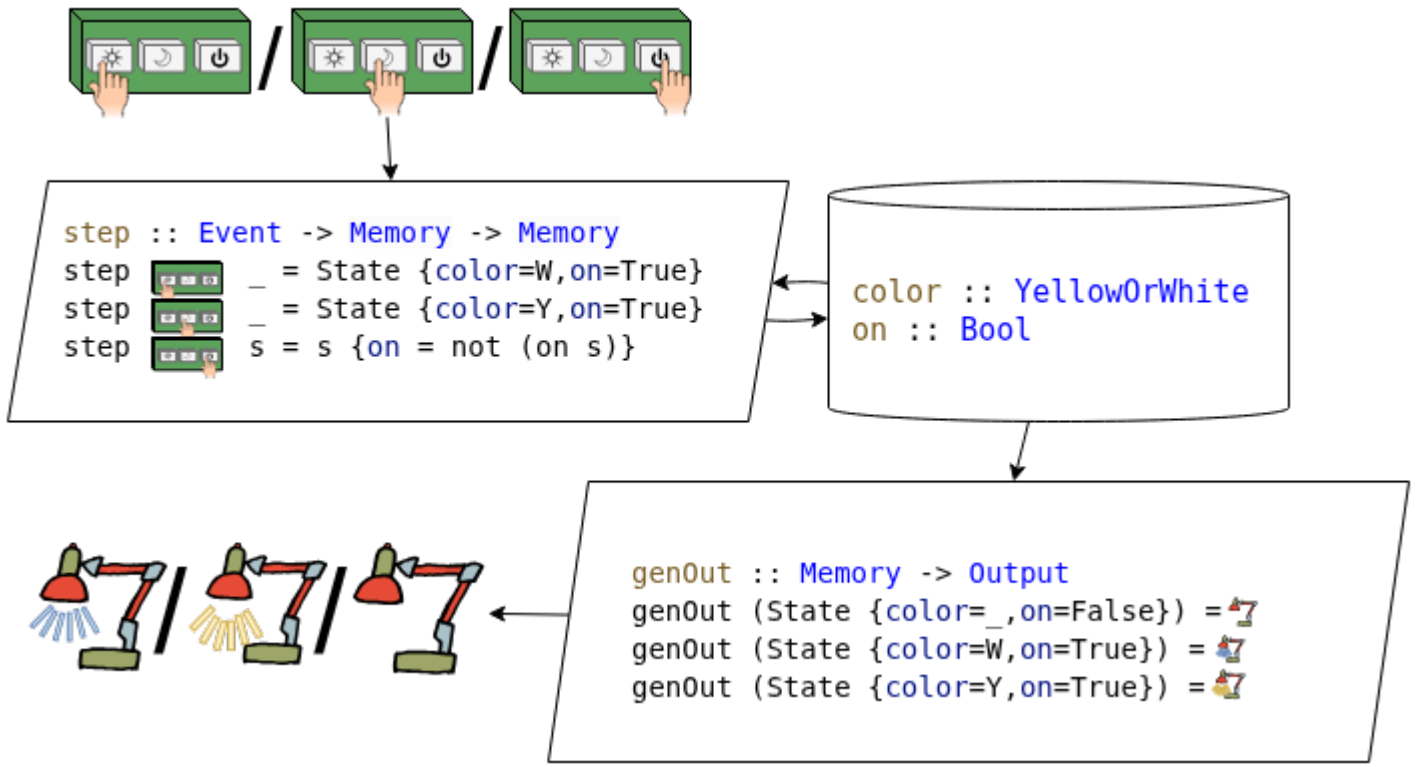
Moore Machines benefits

- Easy to write 
- Easy to modify 
- Easy to verify 

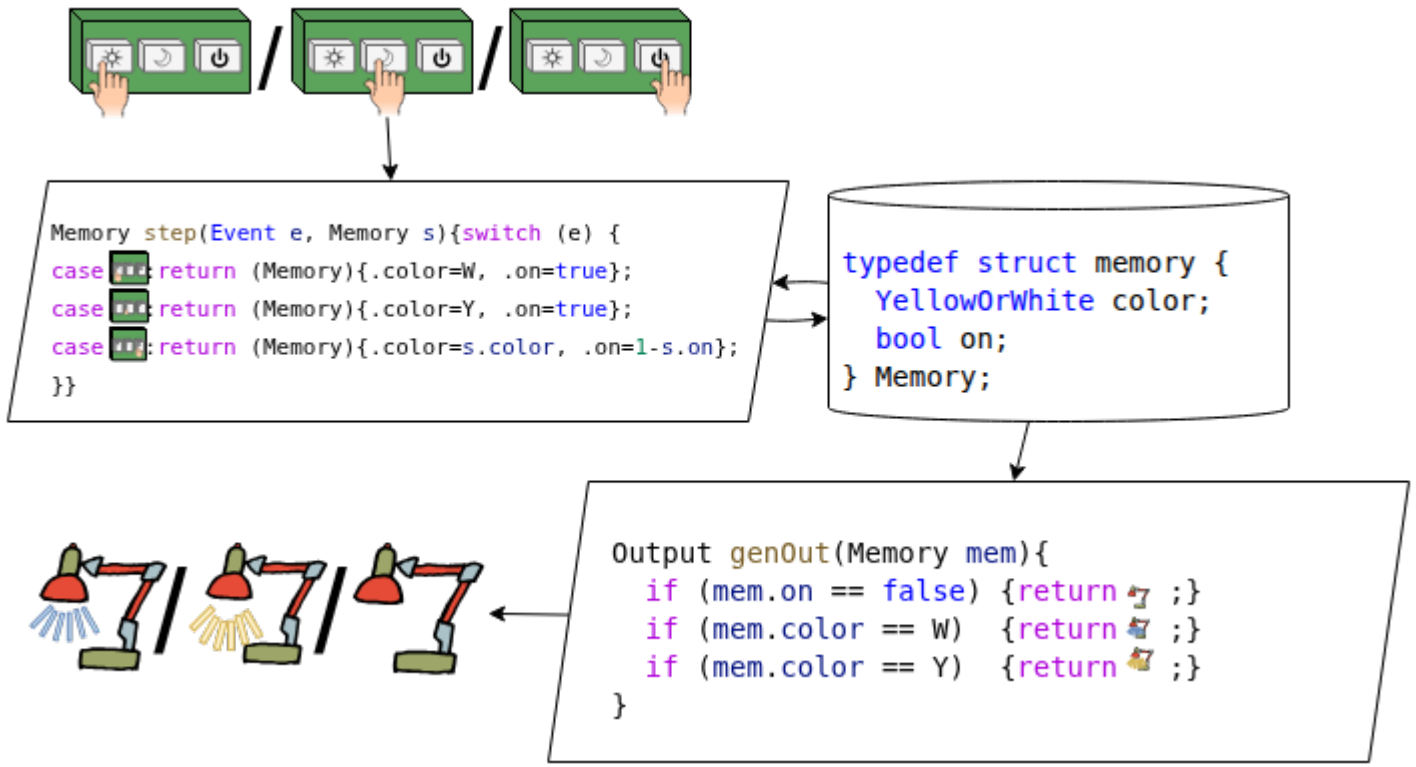
More Moore Machines benefits

- 
  By the inventor of Moore’s law
  - *False!* Edward F. Moore vs Gordon E. Moore
- Easy to implement 👈

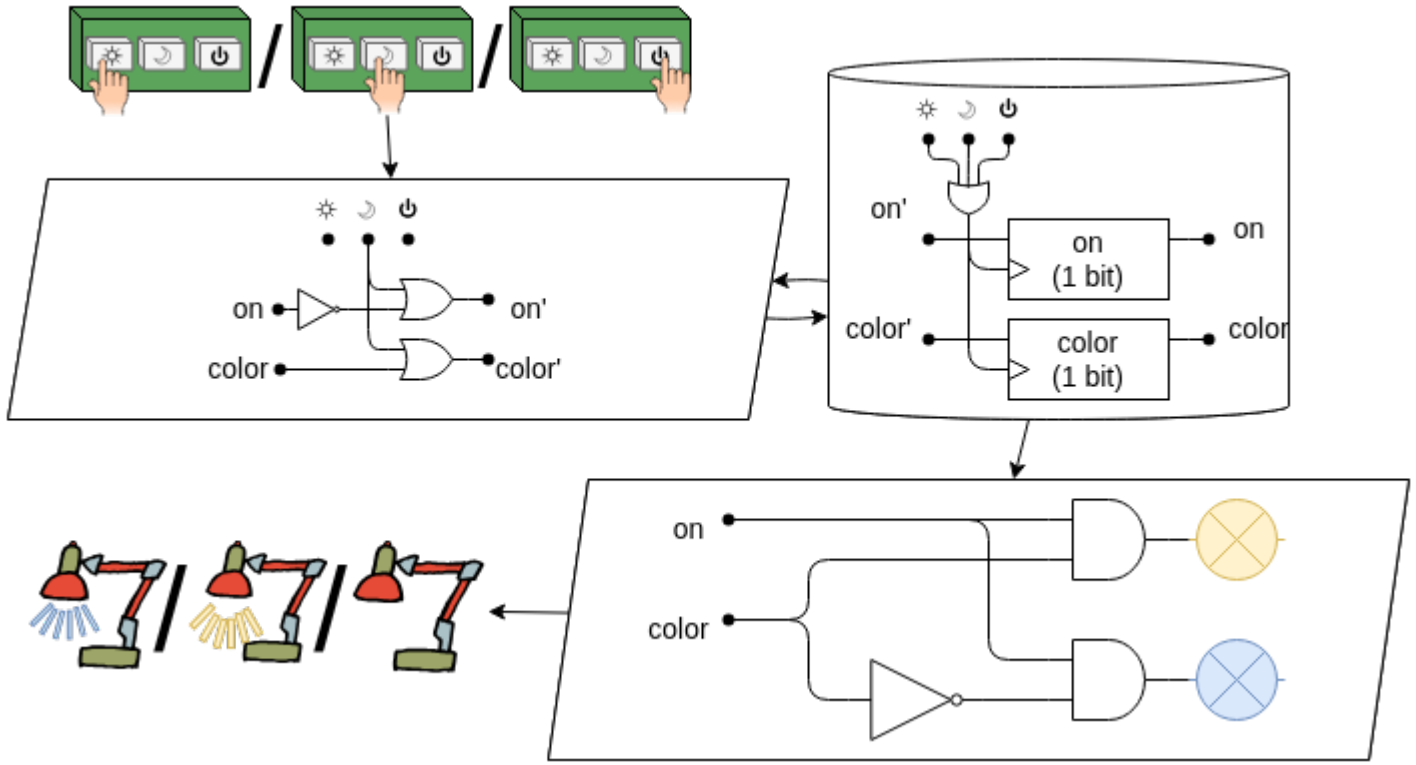
Moore Machines in Haskell



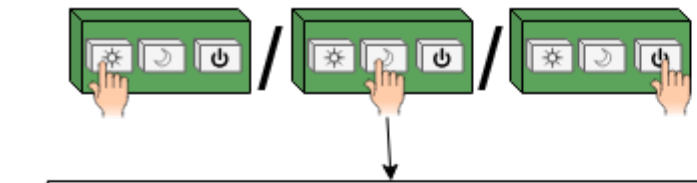
Moore Machines in C



Moore Machines in Hardware



Moore Machines in Mathematics



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

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

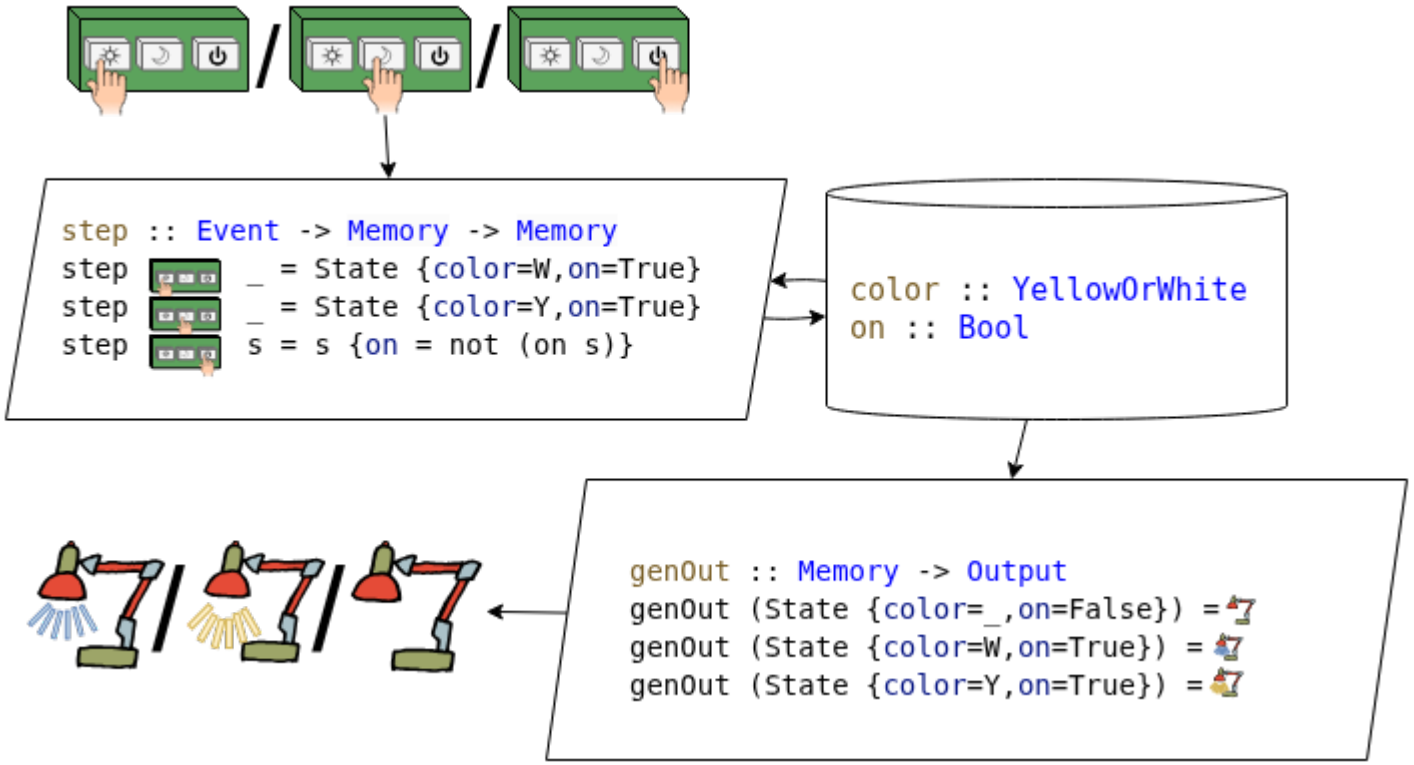
$genOut :: Memory \to Output$
$genOut\ (\_, \bot) =$
$genOut\ (W, \top) =$
$genOut\ (Y, \top) =$

Formal definition [edit]

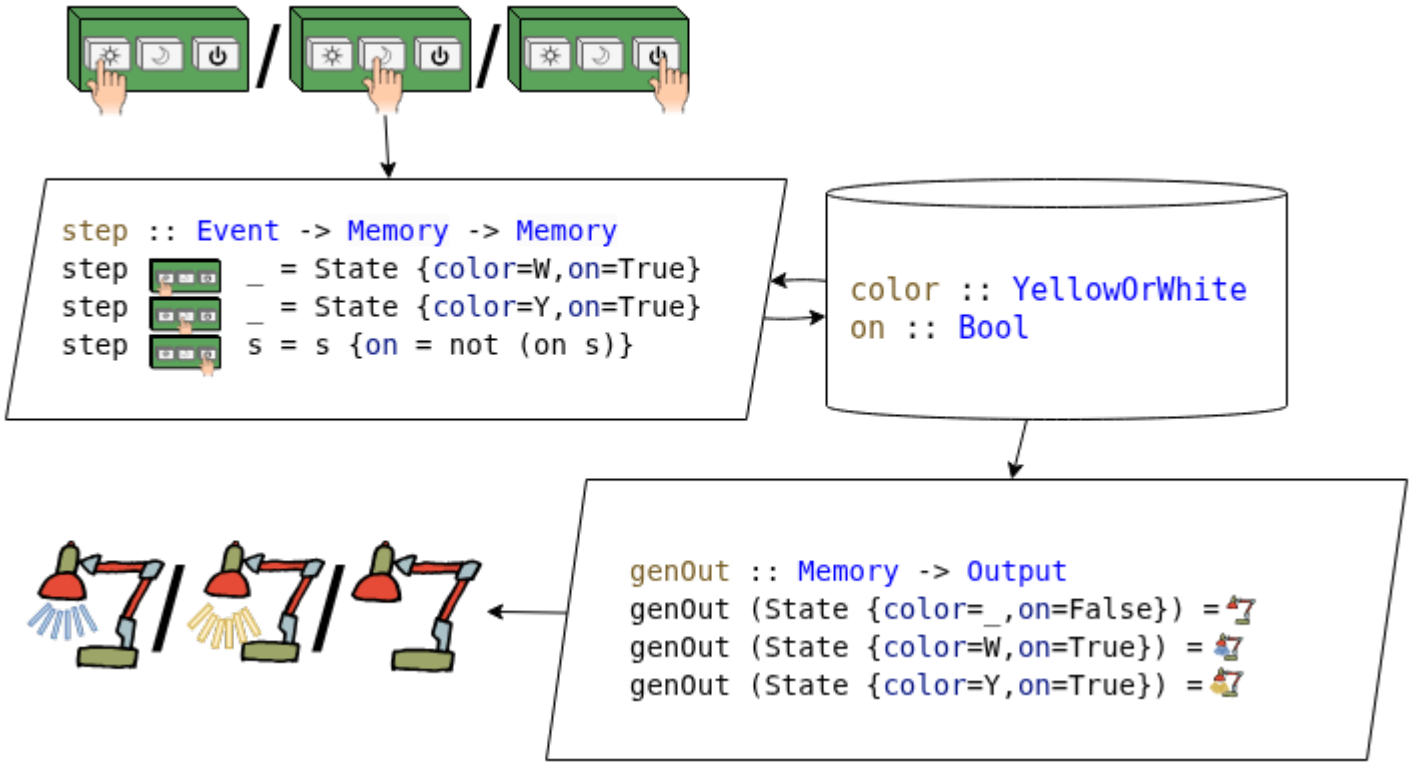
A Moore machine can be defined as a 6-tuple $(S, s_0, \Sigma, O, \delta, G)$ consisting

- 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 Haskell (with types)

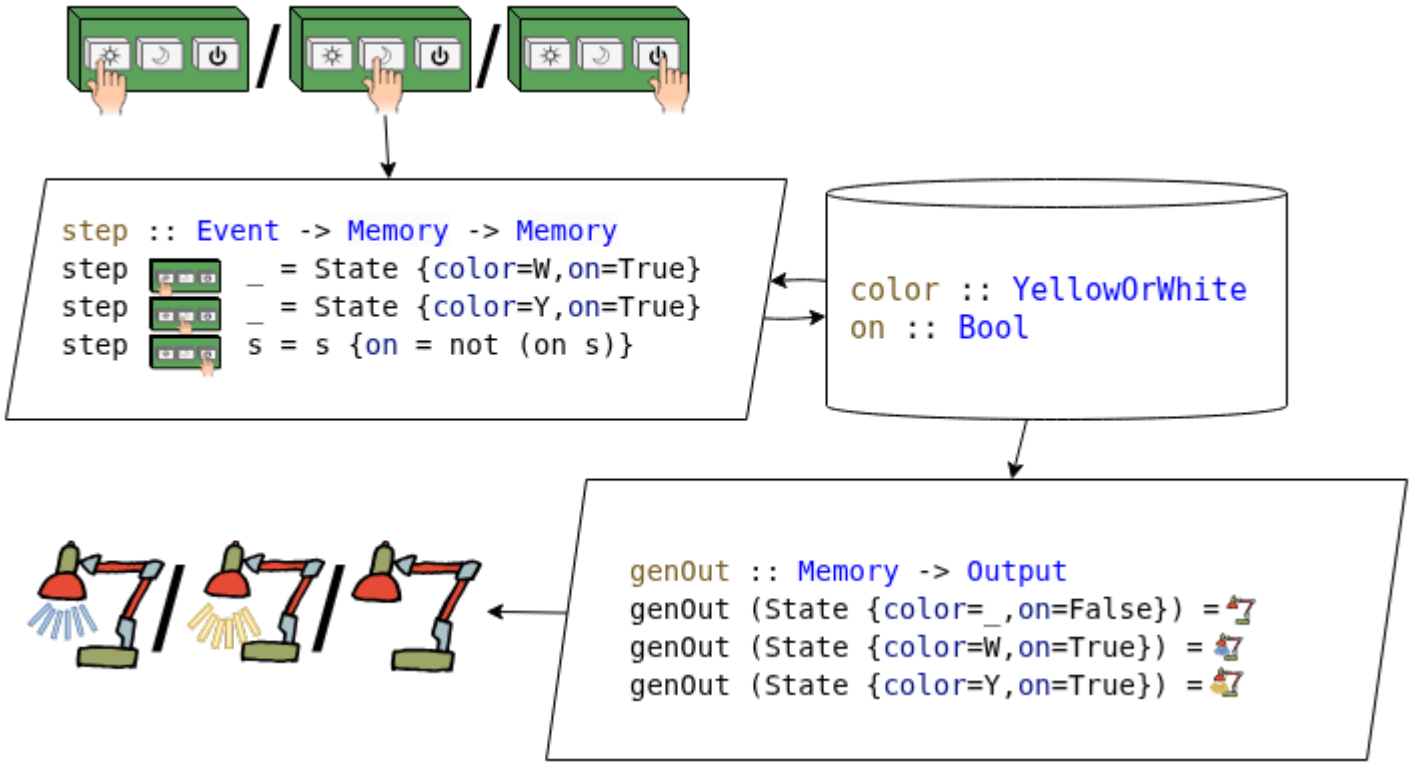


Moore Machines in Haskell (with types)



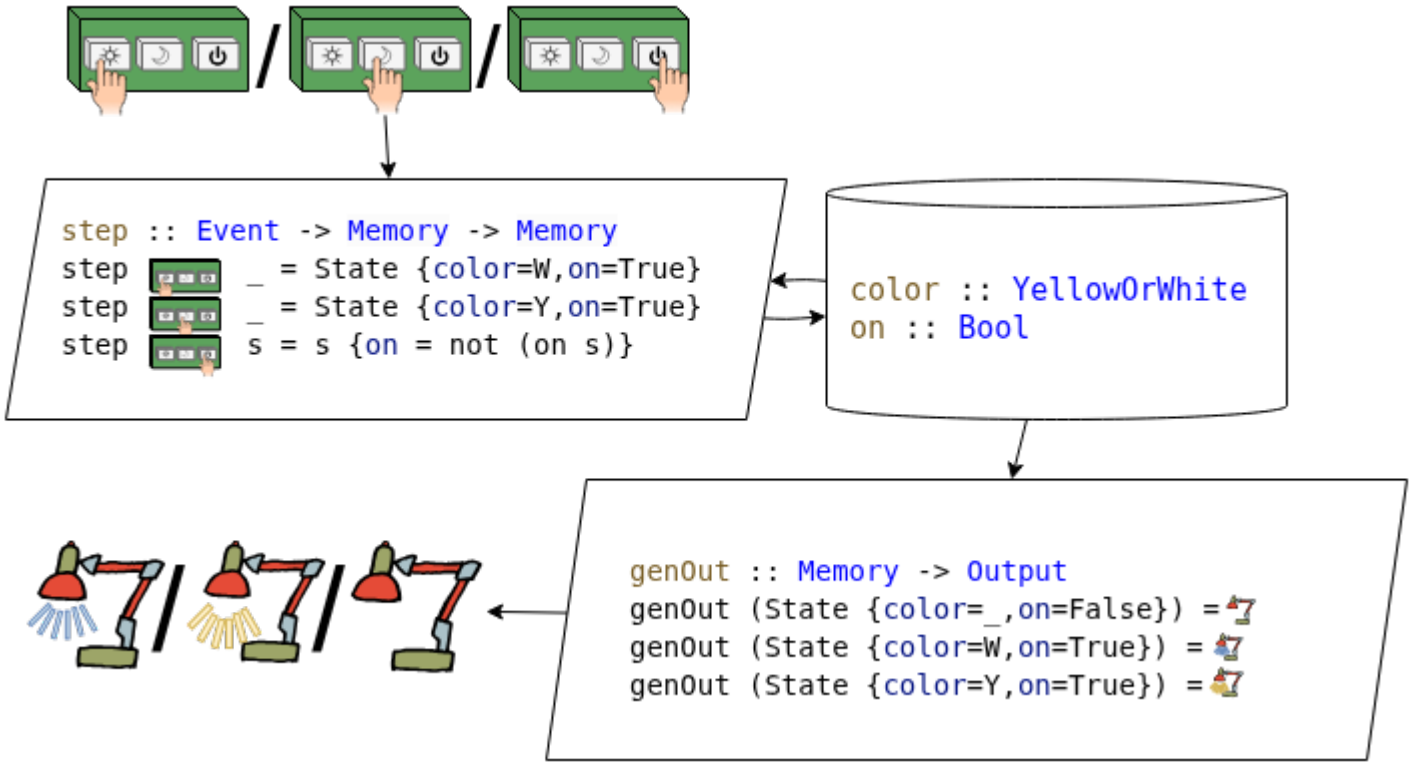

```haskell
data Moore event memory output = Moore
  { step :: event → memory → memory
  , genOut :: memory → output
  , s0 :: memory}
```

Moore Machines in Haskell (with types)



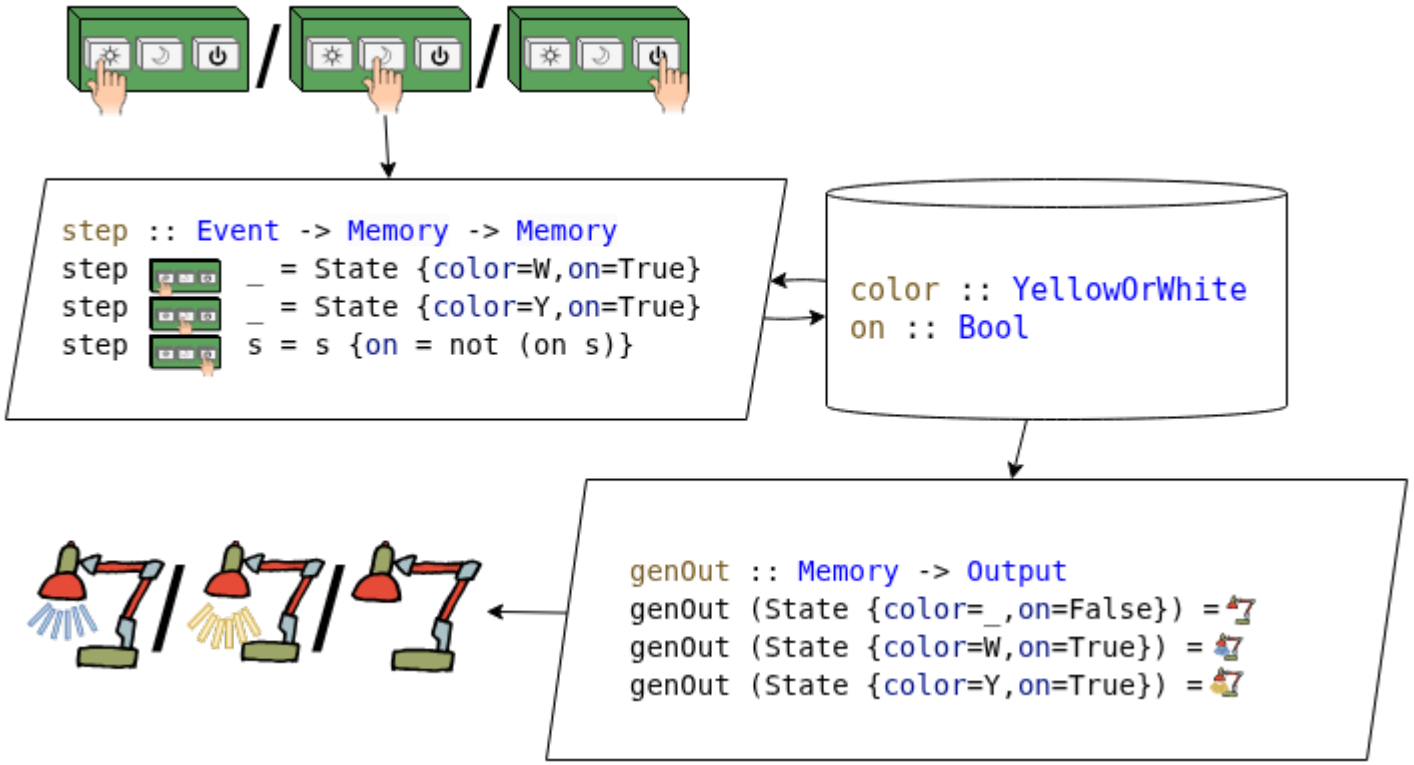

```haskell
data Moore symbol memory output = Moore
  { step :: symbol → memory → memory
  , genOut :: memory → output
  , s0 :: memory}
```

Moore Machines in Haskell (with types)



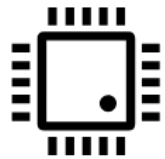

```haskell
data Moore symbol state output = Moore
  { step :: symbol → state → state
  , genOut :: state → output
  , s0 :: state}
```

Moore Machines in Haskell (with types)




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

Moore Machines summary

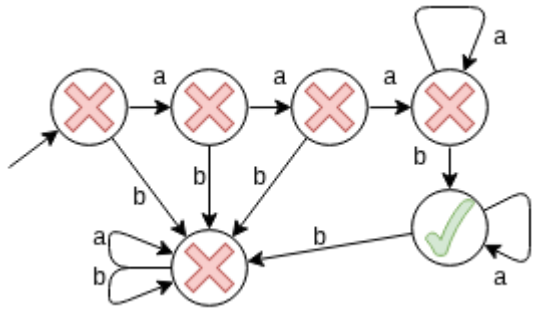
- Easy to use 
- Easy to modify 
- Easy to verify 
- Easy to implement    

Moore Machines for RegExp Matching

a*aaaba*

Moore Machines for RegExp Matching

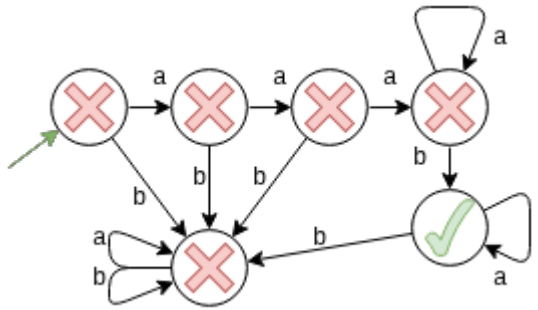
a*aaaba*



aaaaaaabaabba

Moore Machines for RegExp Matching

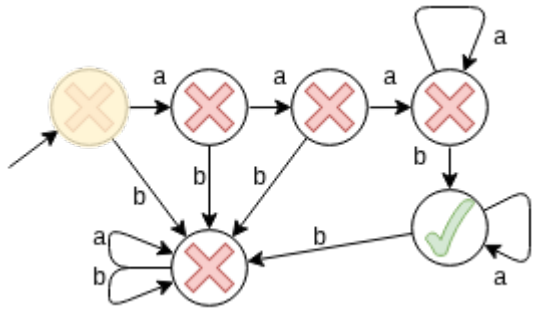
a*aaaba*



aaaaaaabaabba

Moore Machines for RegExp Matching

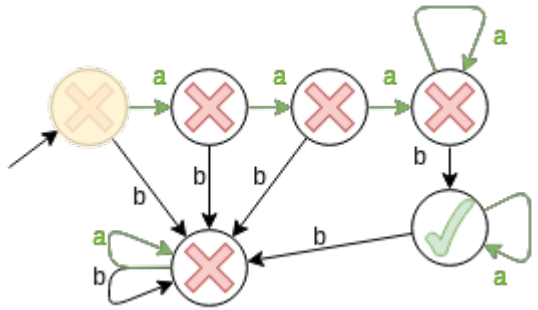
a*aaaba*



|aaaaaaabaabba

Moore Machines for RegExp Matching

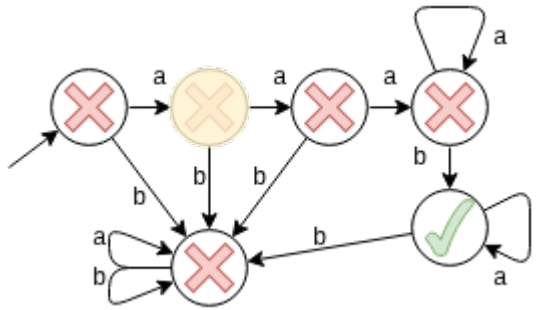
a*aaaba*



[aaaaaaabaabba

Moore Machines for RegExp Matching

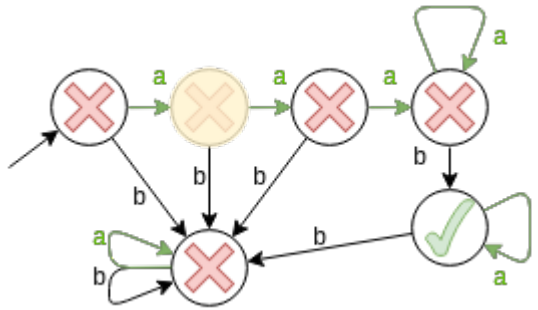
a*aaaba*



a|aaaaaabaabba

Moore Machines for RegExp Matching

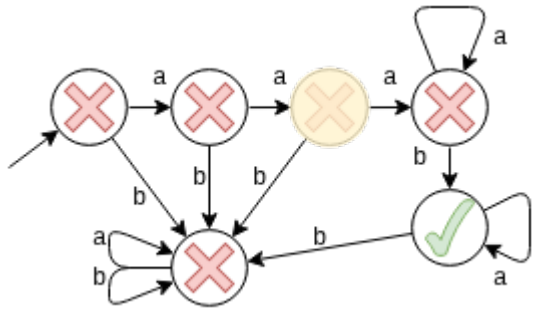
a*aaaba*



a[aaaaaabaabba

Moore Machines for RegExp Matching

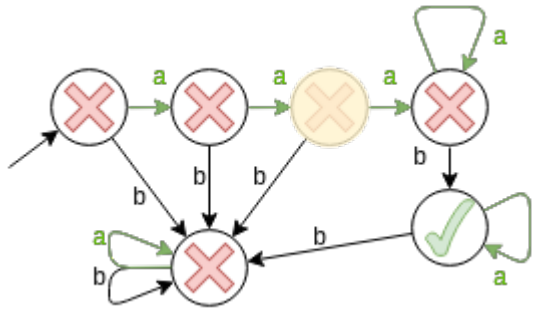
a*aaaba*



aa|aaaaabaabba

Moore Machines for RegExp Matching

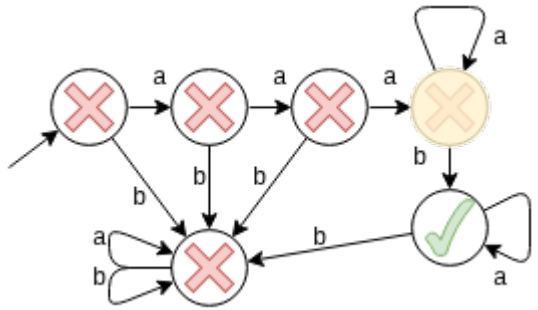
a*aaaba*



aa[aaaaabaabba

Moore Machines for RegExp Matching

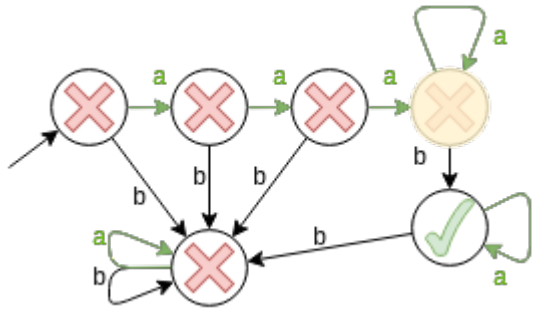
a*aaaba*



aaa|aaaabaabba

Moore Machines for RegExp Matching

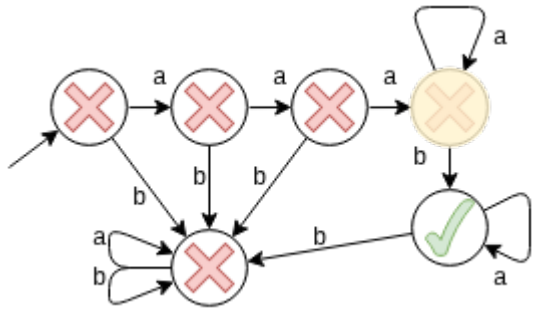
a*aaaba*



aaa[aaaabaabba

Moore Machines for RegExp Matching

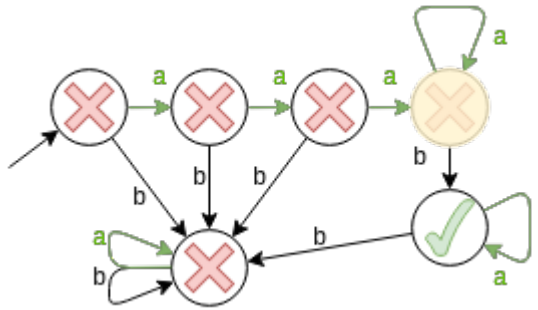
a*aaaba*



aaaa|aaabaabba

Moore Machines for RegExp Matching

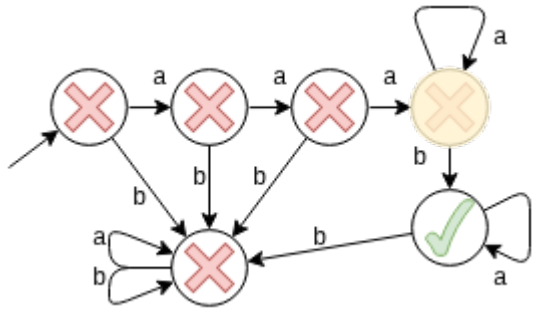
a*aaaba*



aaaa[aaabaabba

Moore Machines for RegExp Matching

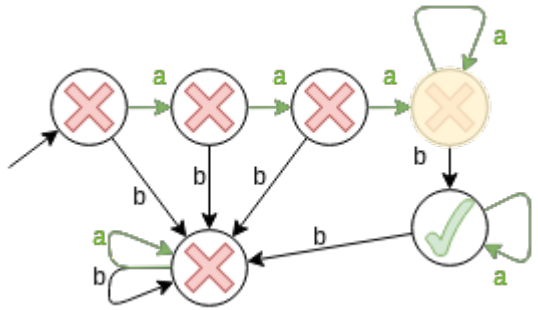
a*aaaba*



aaaaa|aabaabba

Moore Machines for RegExp Matching

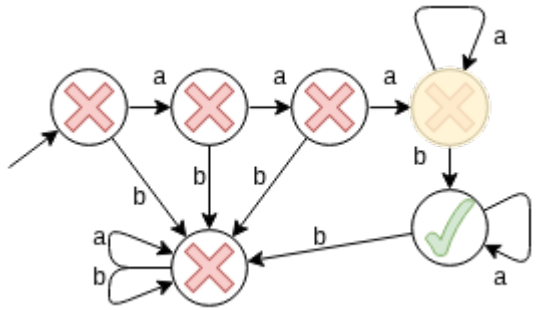
a*aaaba*



aaaaa[aabaabba

Moore Machines for RegExp Matching

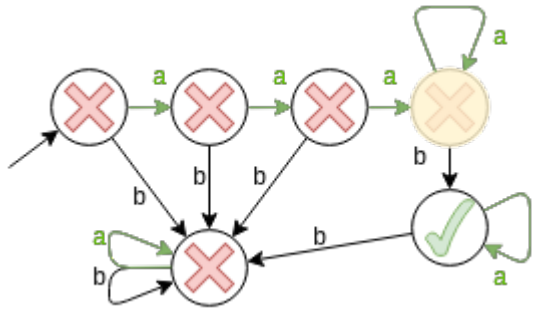
a*aaaba*



aaaaaa|abaabba

Moore Machines for RegExp Matching

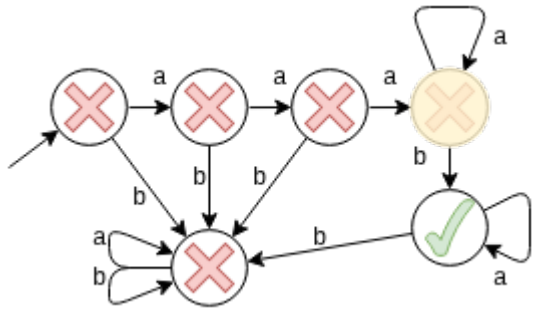
a*aaaba*



aaaaaa[abaabba

Moore Machines for RegExp Matching

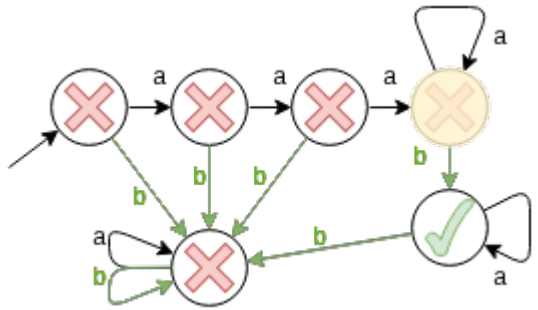
a*aaaba*



aaaaaaa|baabba

Moore Machines for RegExp Matching

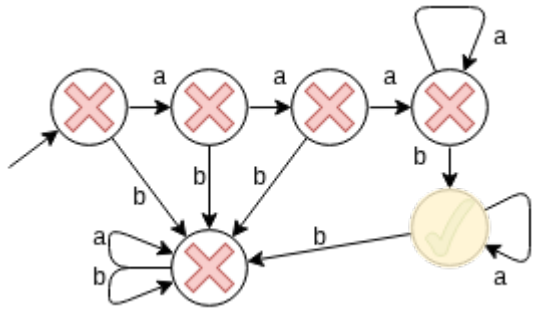
a*aaaba*



aaaaaaa[baabba

Moore Machines for RegExp Matching

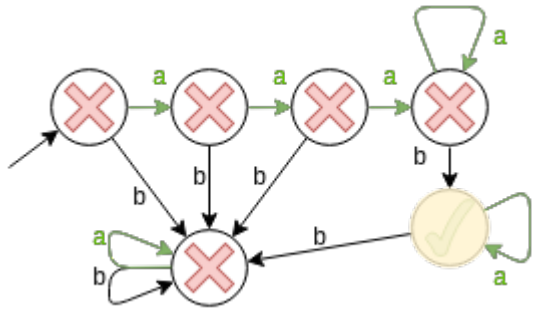
a*aaaba*



aaaaaaab|aabba

Moore Machines for RegExp Matching

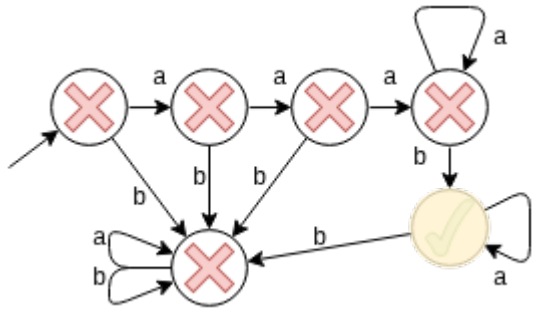
a*aaaba*



aaaaaaab[aabba

Moore Machines for RegExp Matching

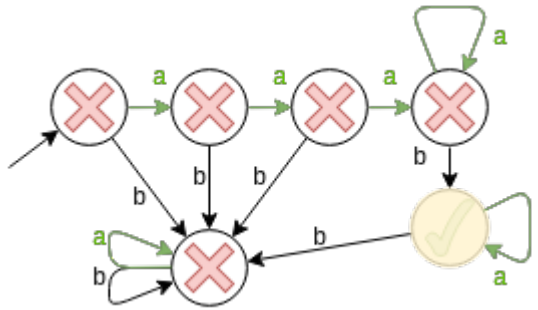
a*aaaba*



aaaaaaaba|abba

Moore Machines for RegExp Matching

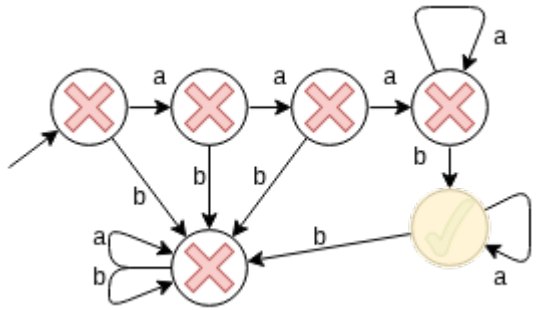
a*aaaba*



aaaaaaaba[abba

Moore Machines for RegExp Matching

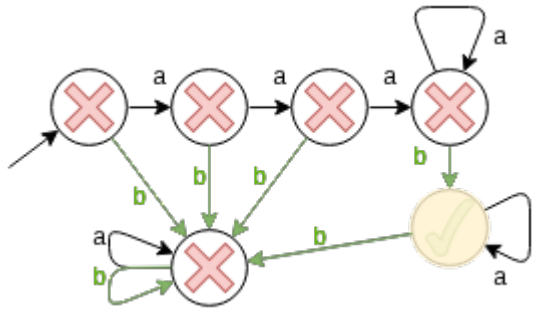
a*aaaba*



aaaaaaabaa|bba

Moore Machines for RegExp Matching

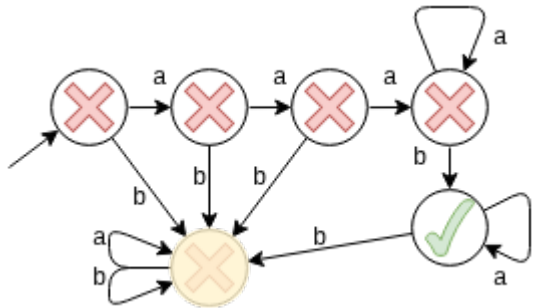
a*aaaba*



aaaaaaabaa[bba

Moore Machines for RegExp Matching

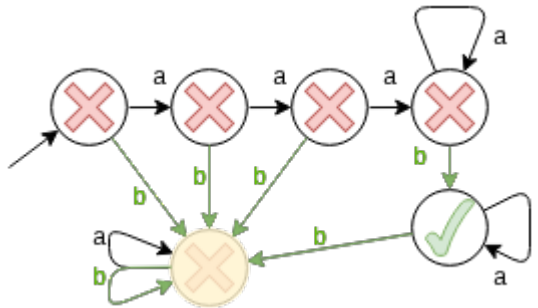
a*aaaba*



aaaaaaabaab|ba

Moore Machines for RegExp Matching

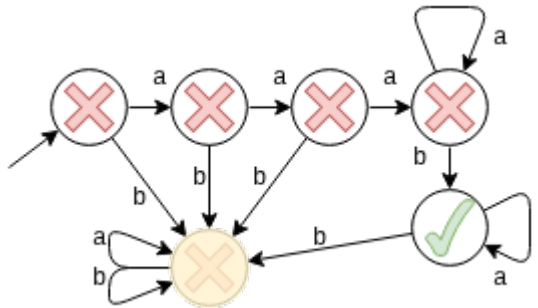
a*aaaba*



aaaaaaabaab[ba

Moore Machines for RegExp Matching

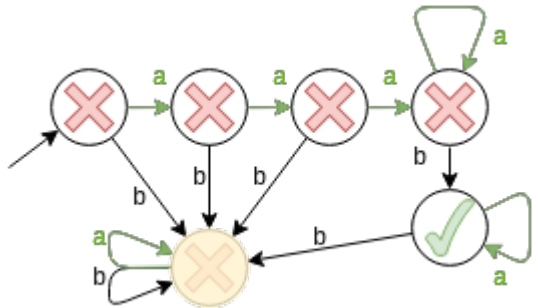
a*aaaba*



aaaaaaabaabb|a

Moore Machines for RegExp Matching

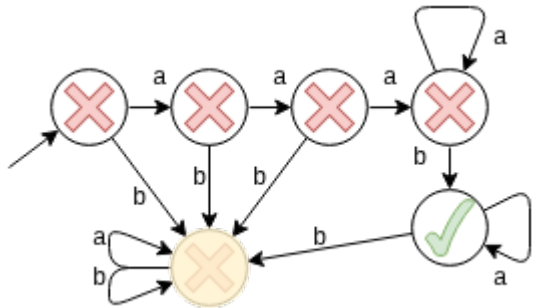
a*aaaba*



aaaaaaabaabb[a

Moore Machines for RegExp Matching

a*aaaba*



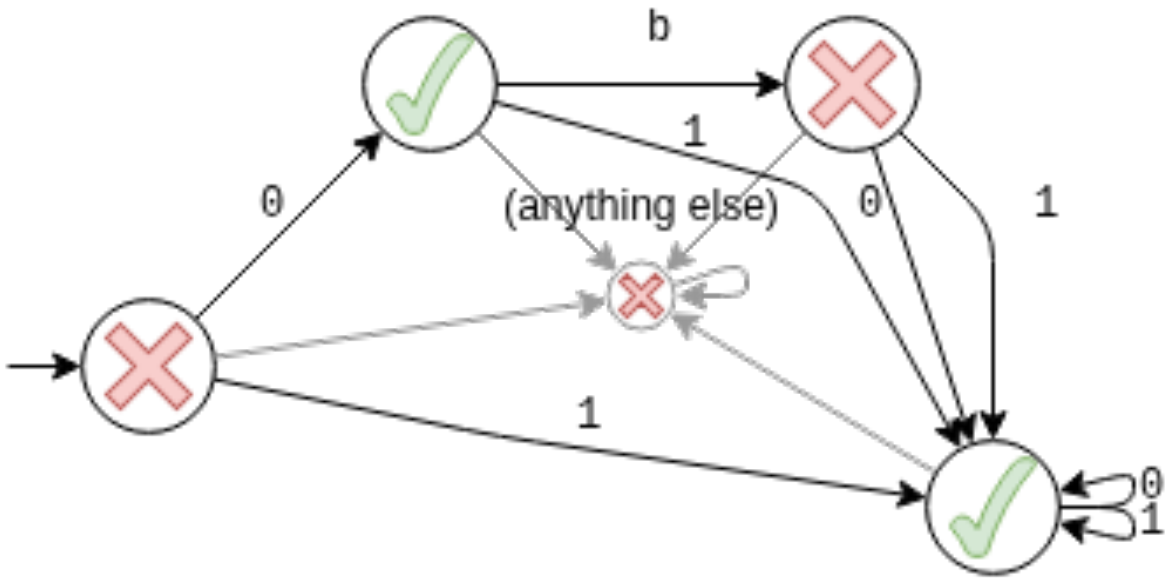
aaaaaaabaabba|

More Moore Machine Matching

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

More Moore Machine Matching

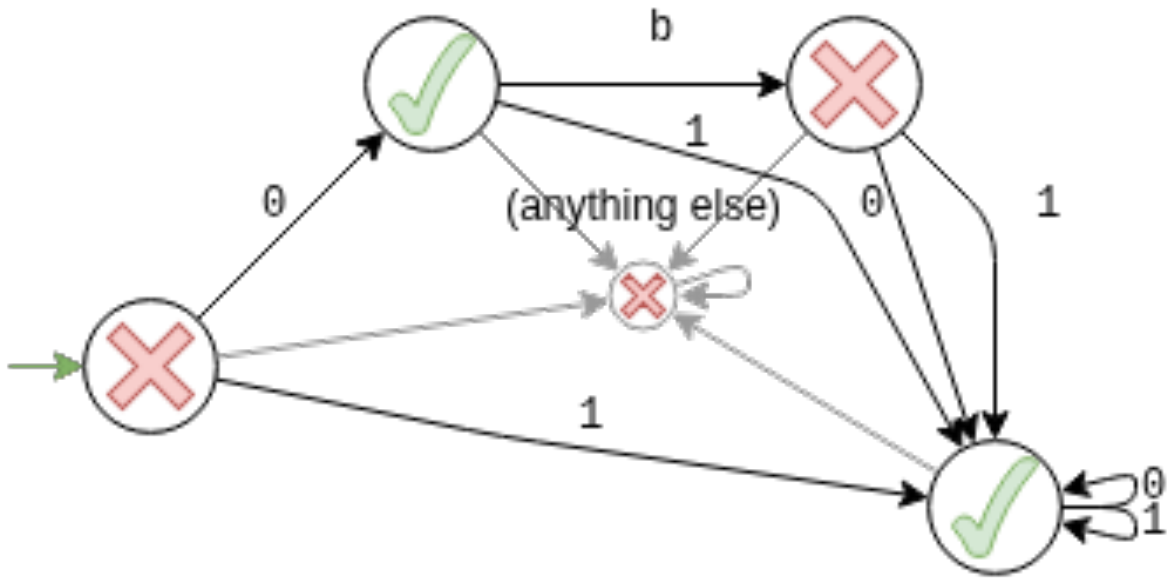
```
(0b)?(0|1)+
```




```
0b110100,10
```

More Moore Machine Matching

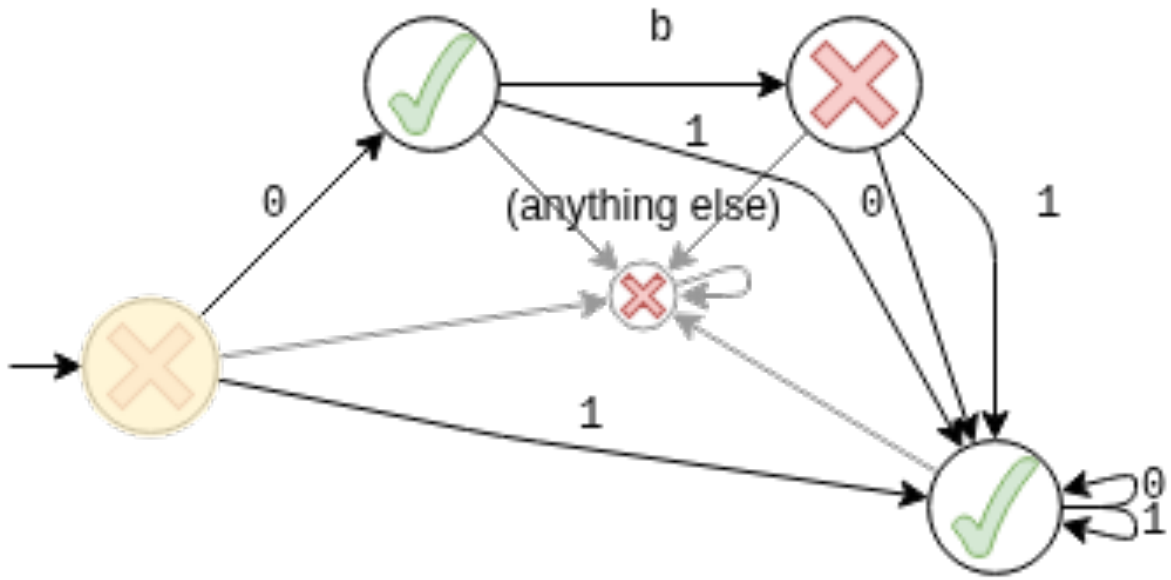
```
(0b)?(0|1)+
```




```
0b110100,10
```

More Moore Machine Matching

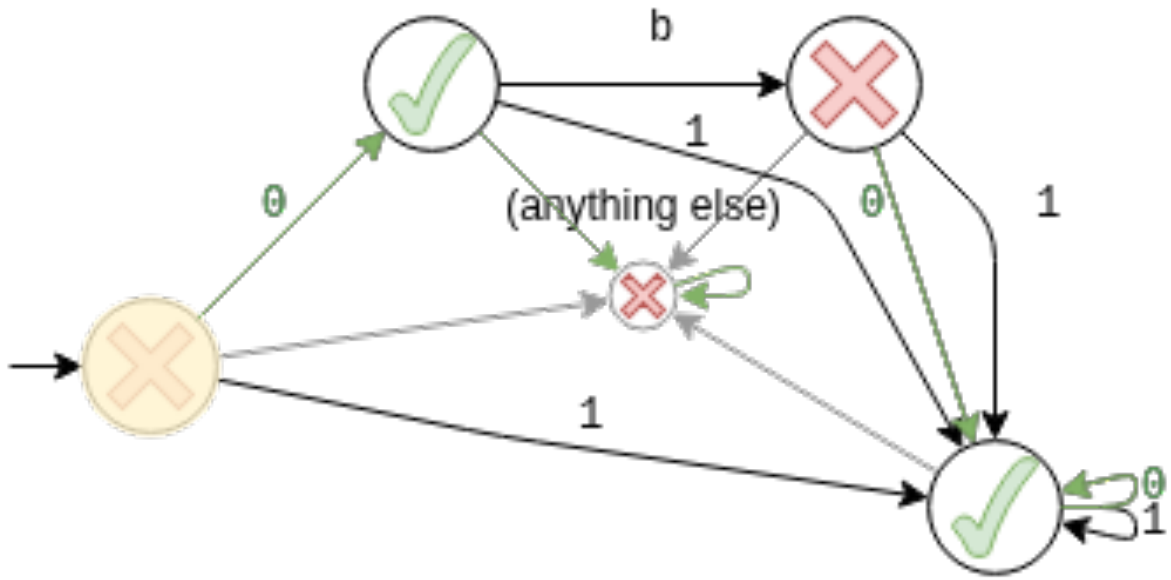
```
(0b)?(0|1)+
```




```
|0b110100,10
```

More Moore Machine Matching

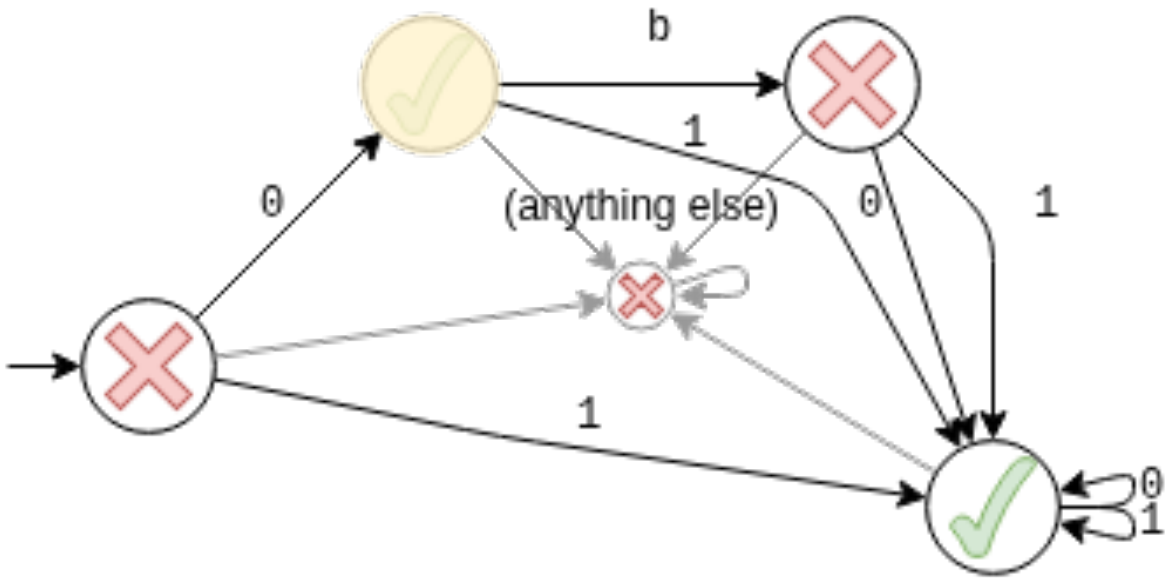
```
(0b)?(0|1)+
```




```
[0b110100,10
```

More Moore Machine Matching

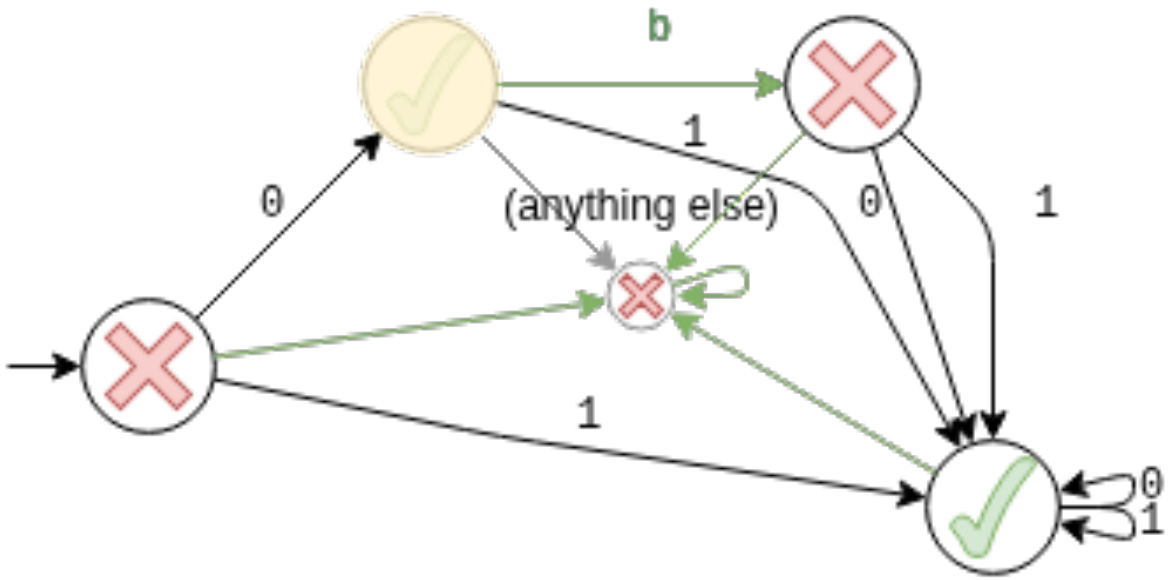
```
(0b)?(0|1)+
```




```
0|b110100,10
```

More Moore Machine Matching

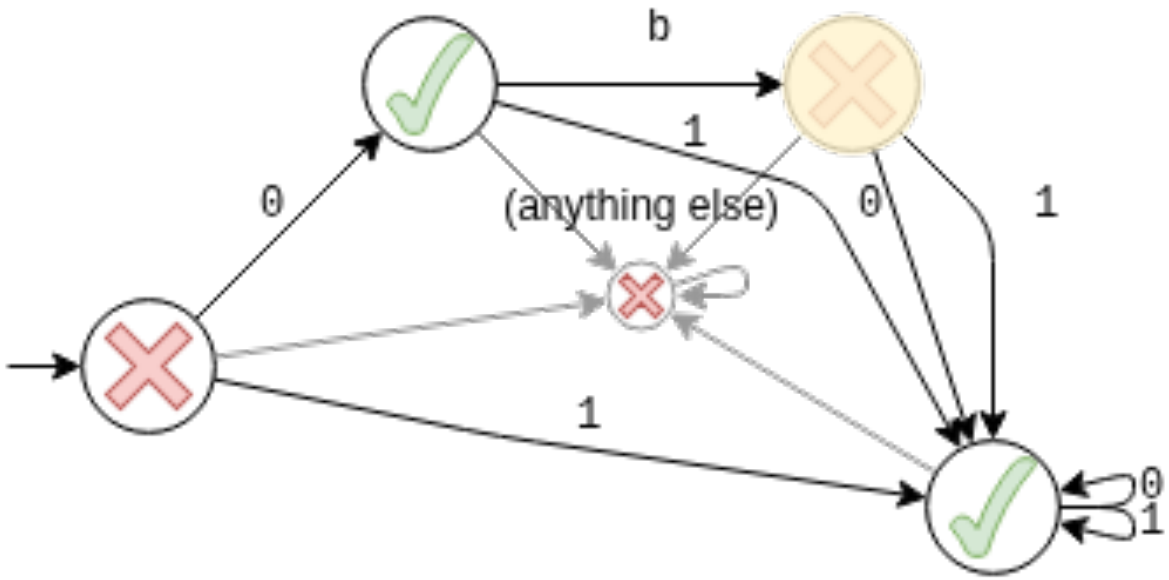
```
(0b)?(0|1)+
```




```
0[b110100,10
```

More Moore Machine Matching

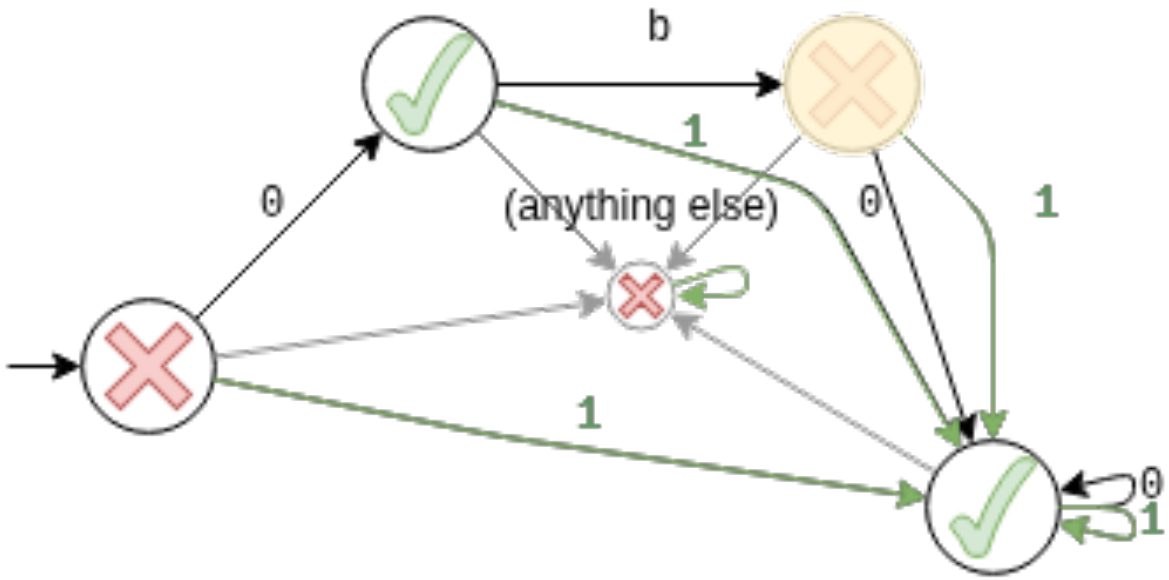
```
(0b)?(0|1)+
```




```
0b|110100,10
```

More Moore Machine Matching

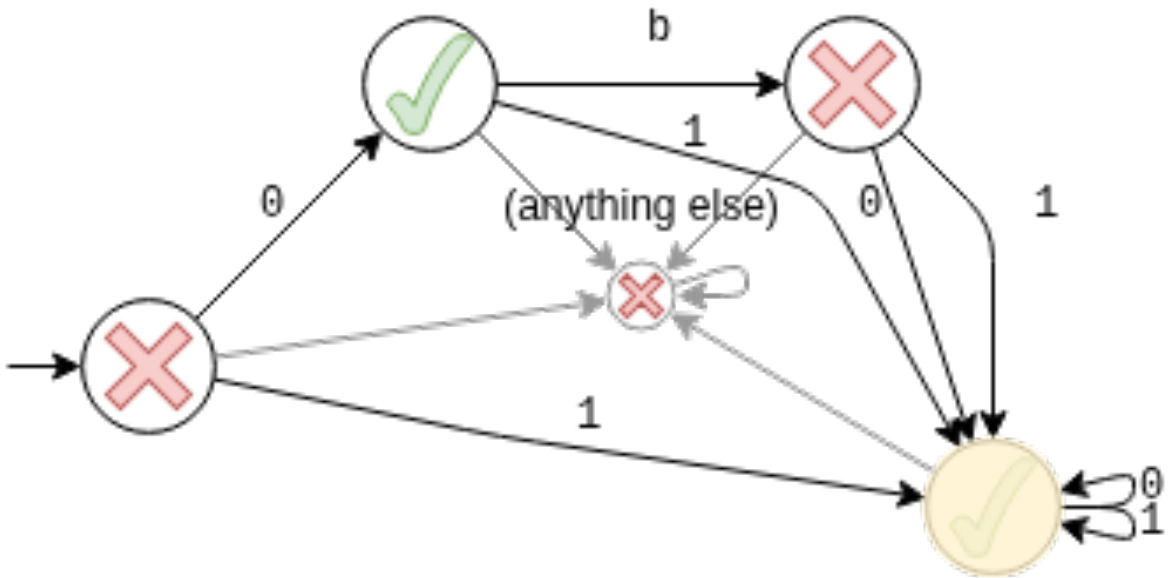
```
(0b)?(0|1)+
```




```
0b[110100,10
```

More Moore Machine Matching

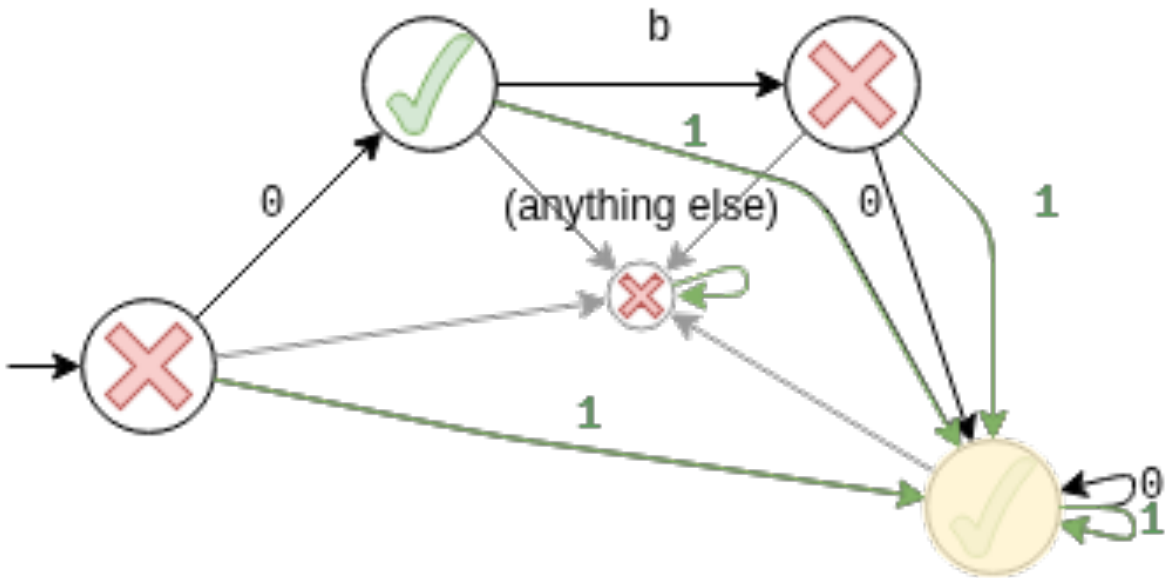
```
(0b)?(0|1)+
```




```
0b1|10100,10
```

More Moore Machine Matching

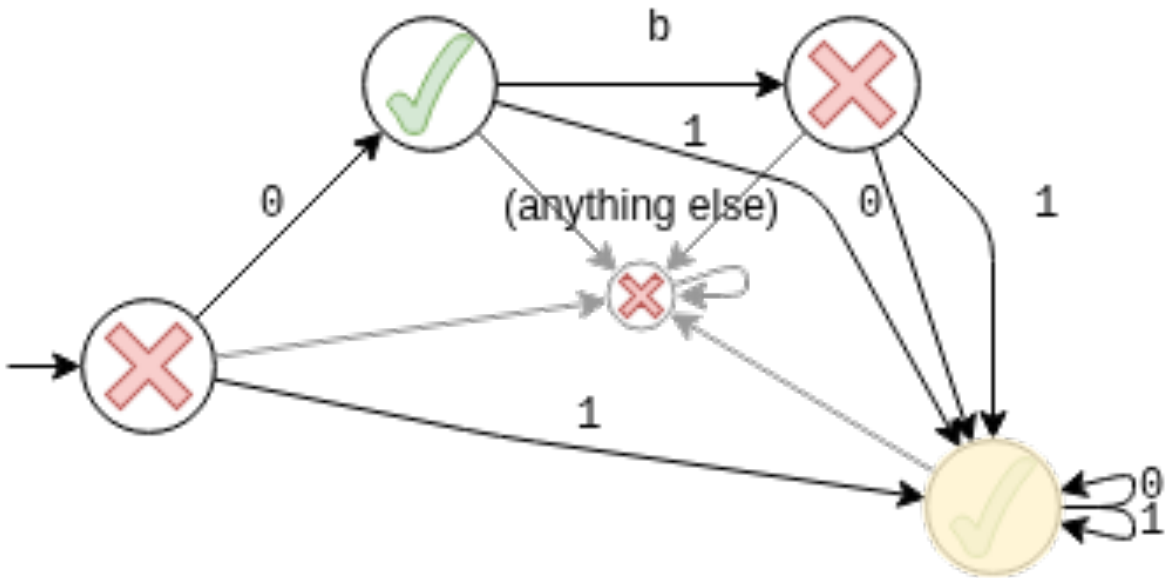
```
(0b)?(0|1)+
```




```
0b1[10100,10
```

More Moore Machine Matching

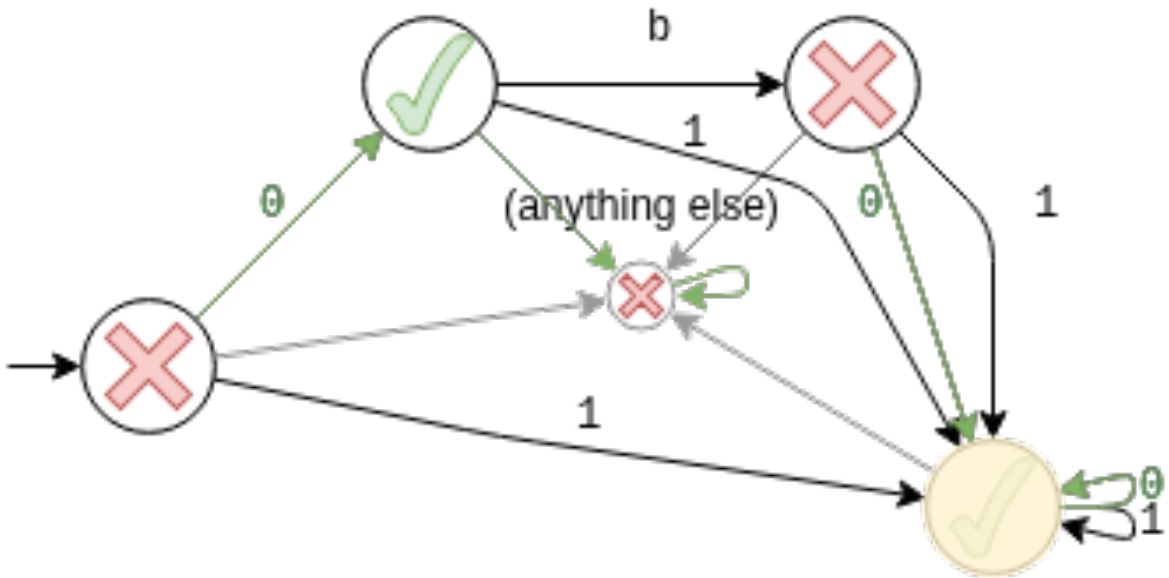
```
(0b)?(0|1)+
```




```
0b11|0100,10
```

More Moore Machine Matching

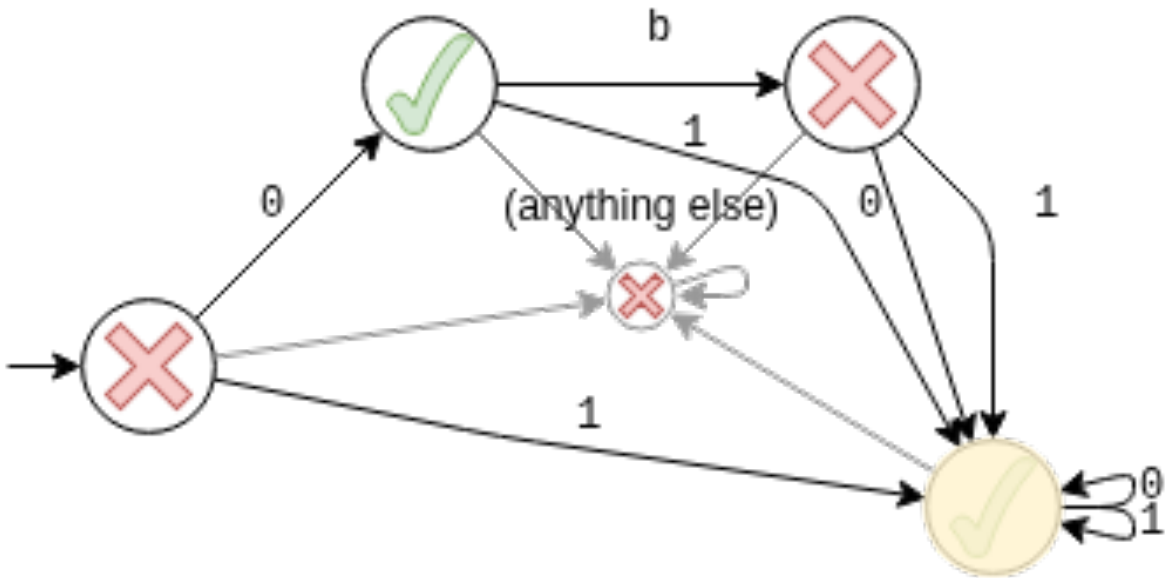
```
(0b)?(0|1)+
```




```
0b11[0100,10
```

More Moore Machine Matching

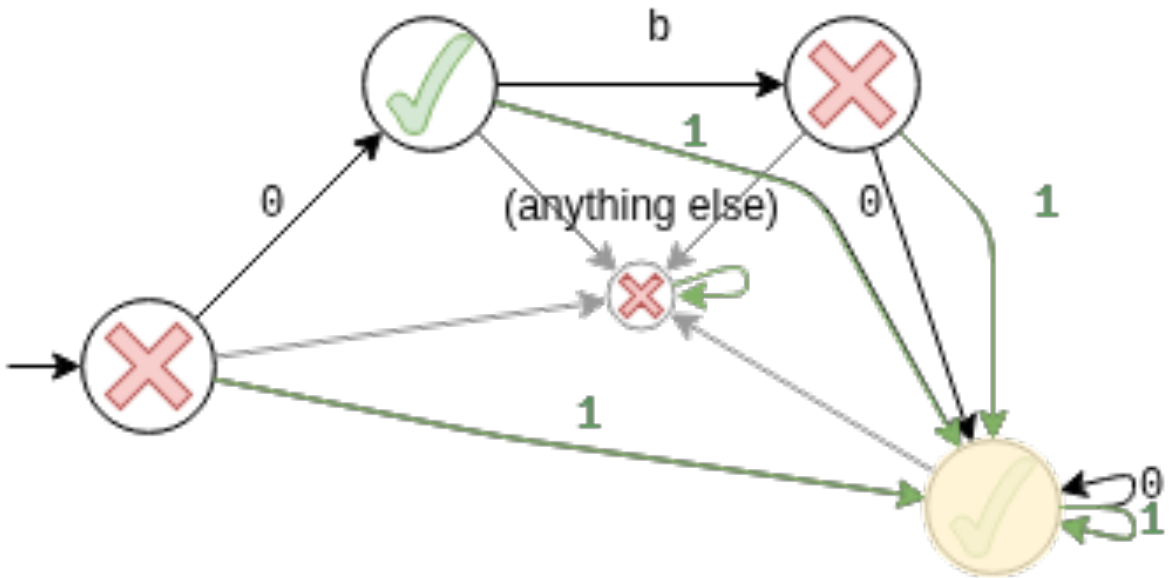
```
(0b)?(0|1)+
```




```
0b110|100,10
```

More Moore Machine Matching

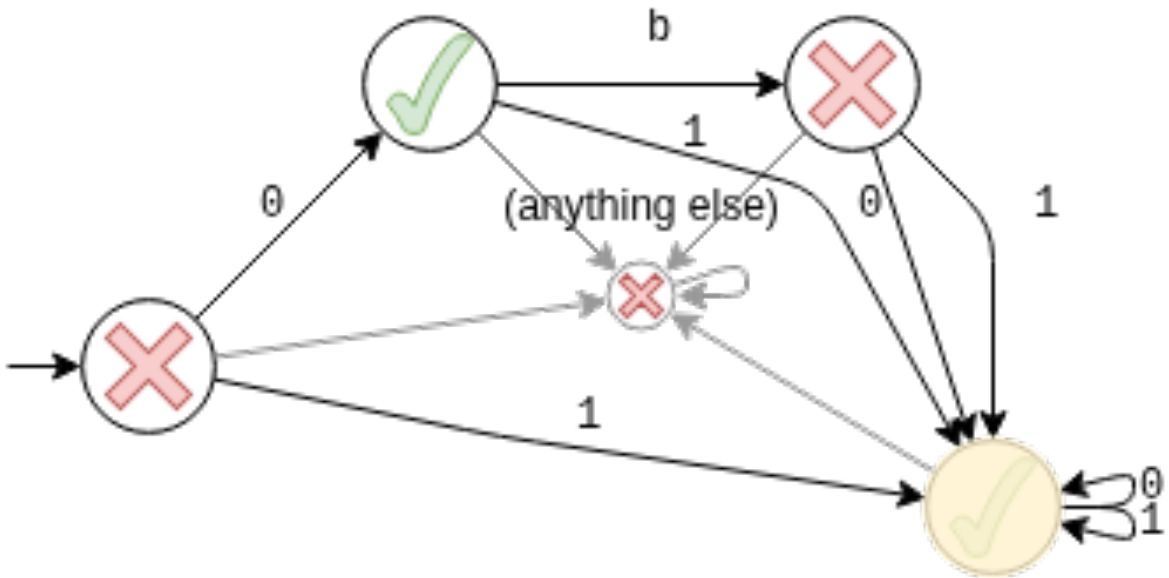
```
(0b)?(0|1)+
```




```
0b110[100,10
```

More Moore Machine Matching

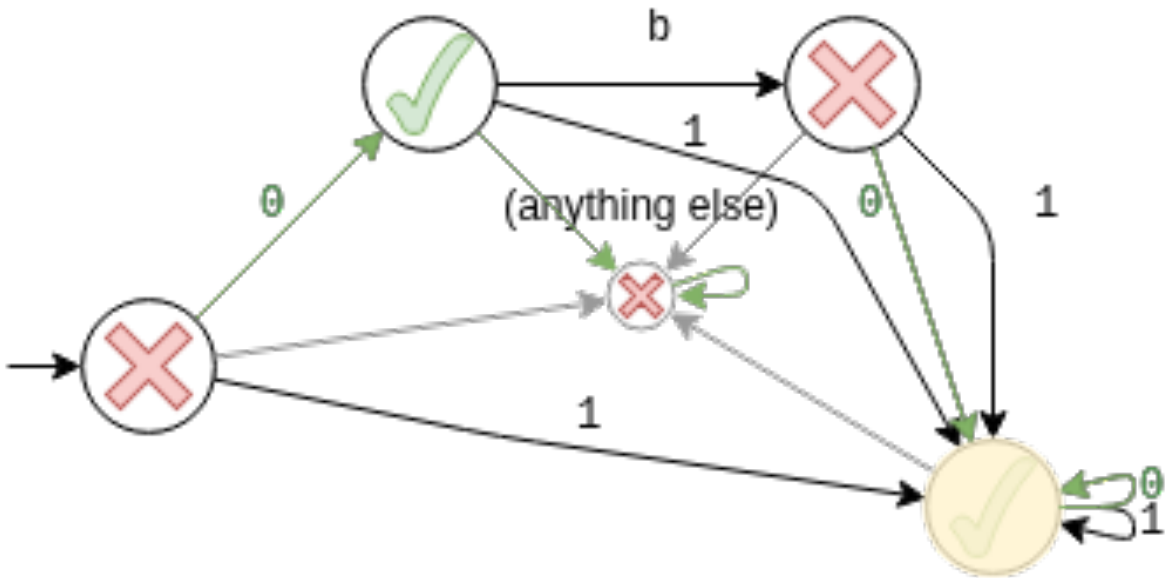
```
(0b)?(0|1)+
```




```
0b1101|00,10
```

More Moore Machine Matching

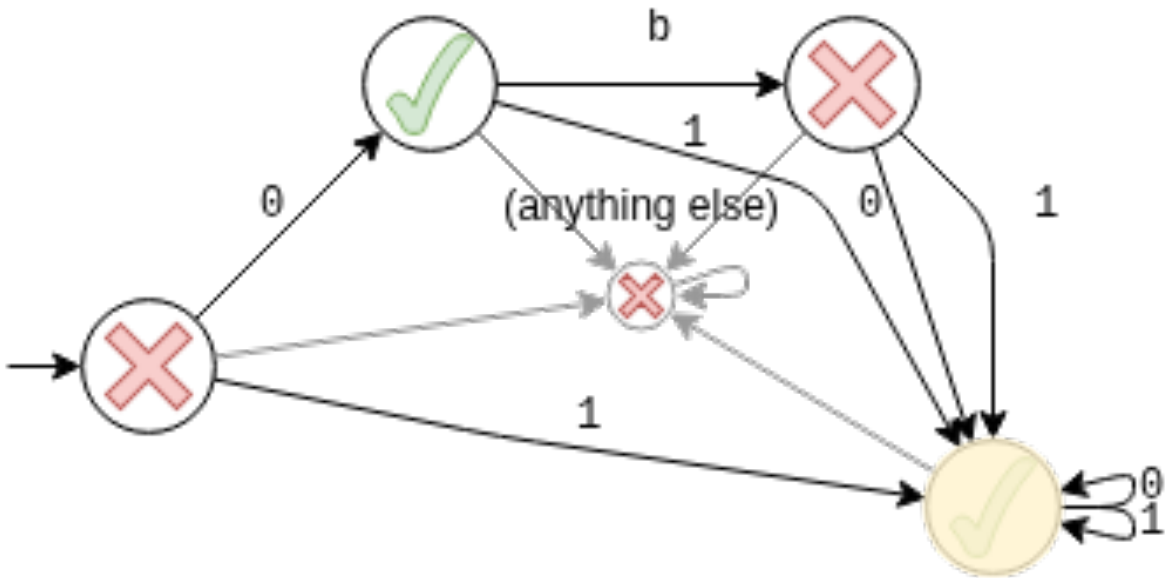
```
(0b)?(0|1)+
```




```
0b1101[00,10
```

More Moore Machine Matching

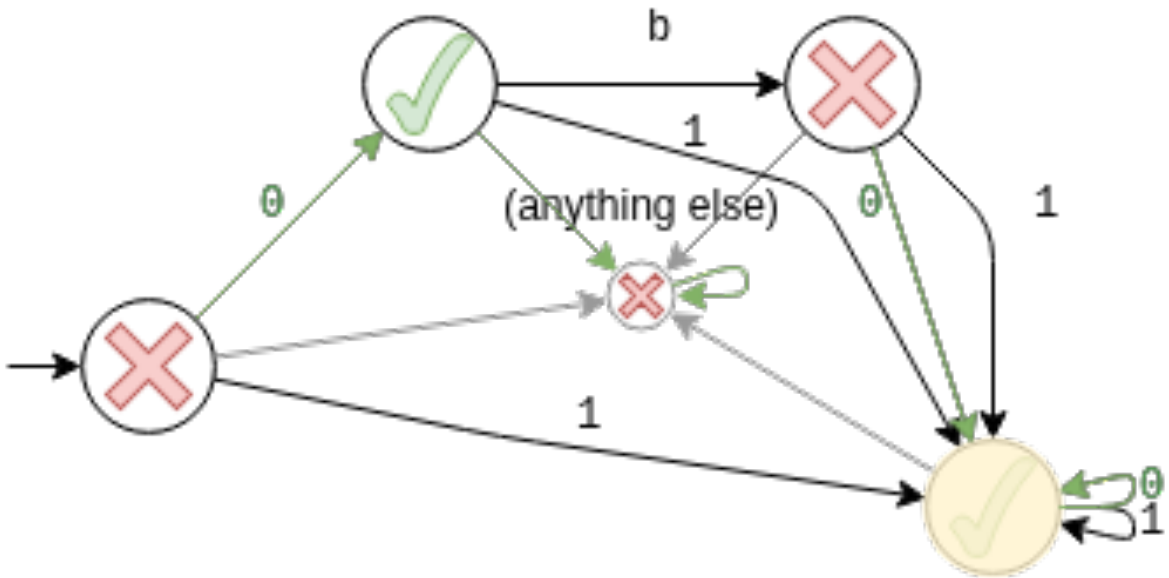
```
(0b)?(0|1)+
```




```
0b11010|0,10
```

More Moore Machine Matching

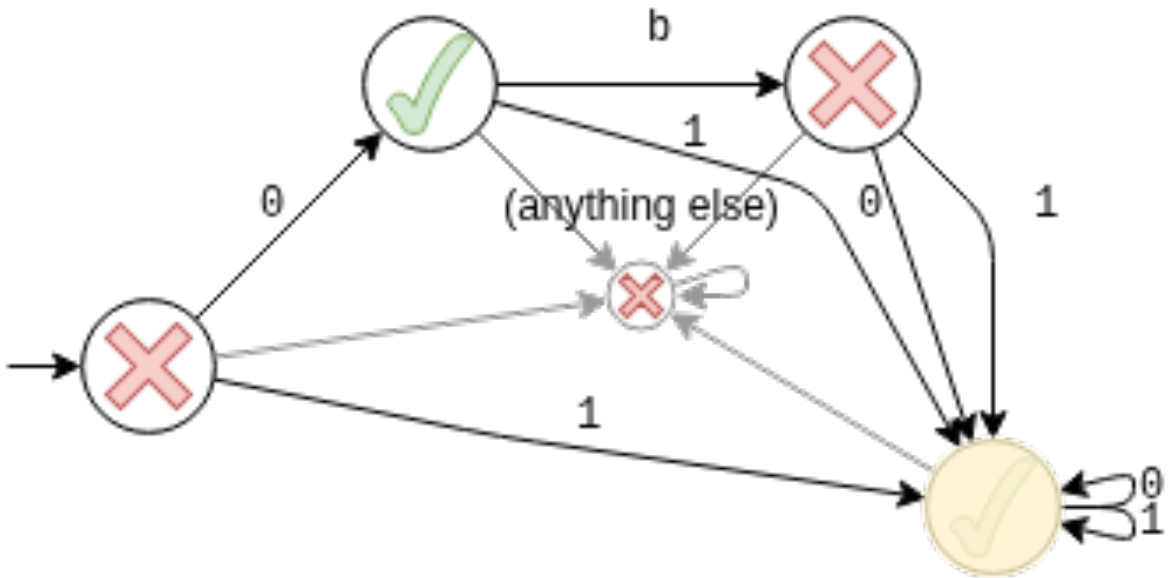
```
(0b)?(0|1)+
```




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

More Moore Machine Matching

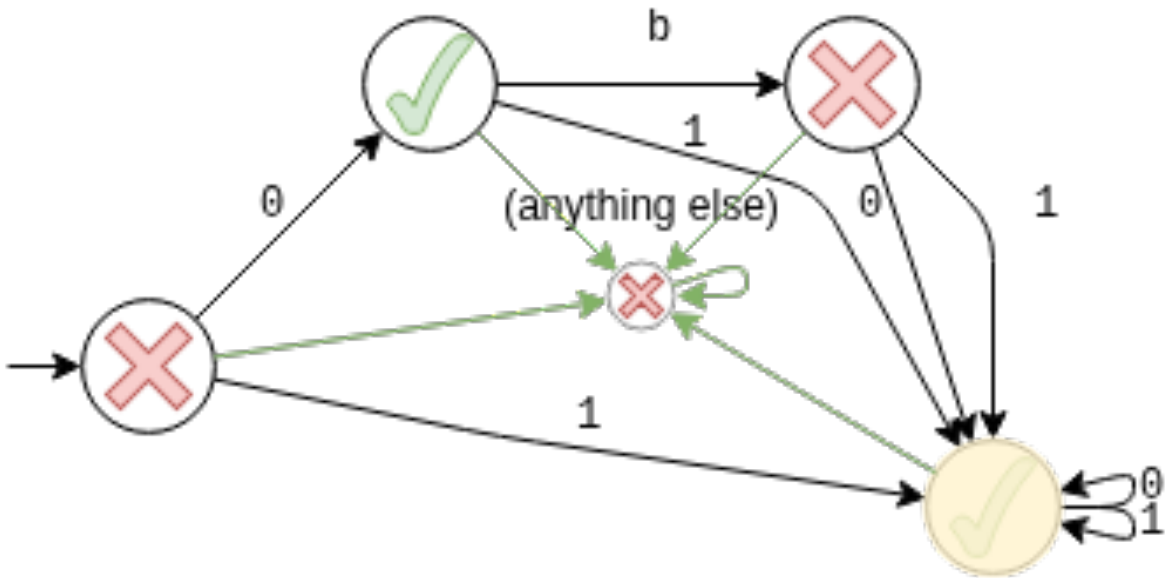
```
(0b)?(0|1)+
```




```
0b110100|,10
```

More Moore Machine Matching

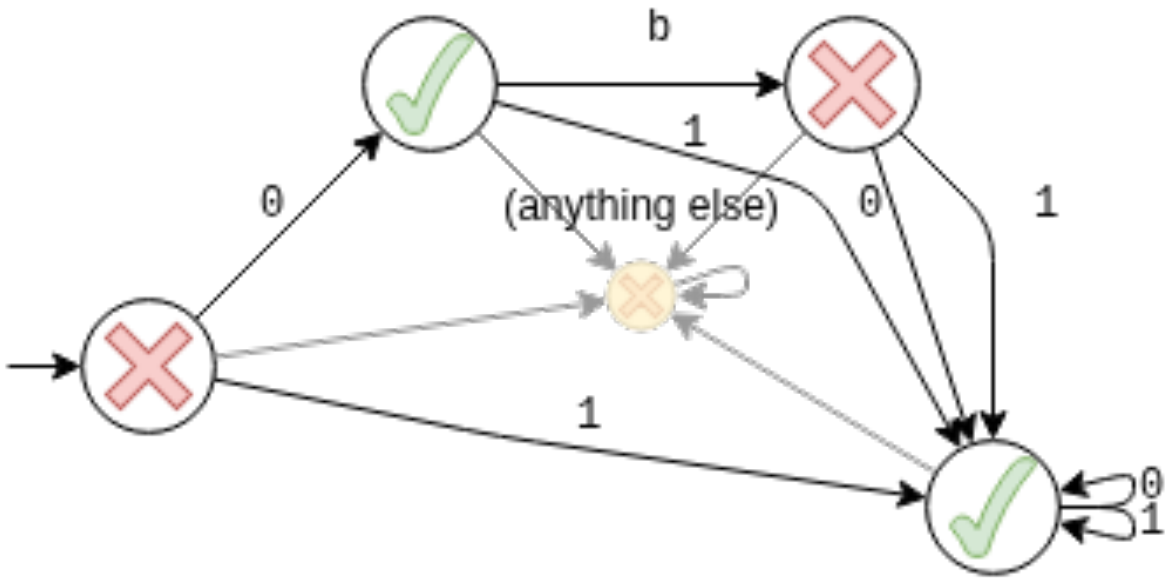
```
(0b)?(0|1)+
```




```
0b110100[,10
```

More Moore Machine Matching

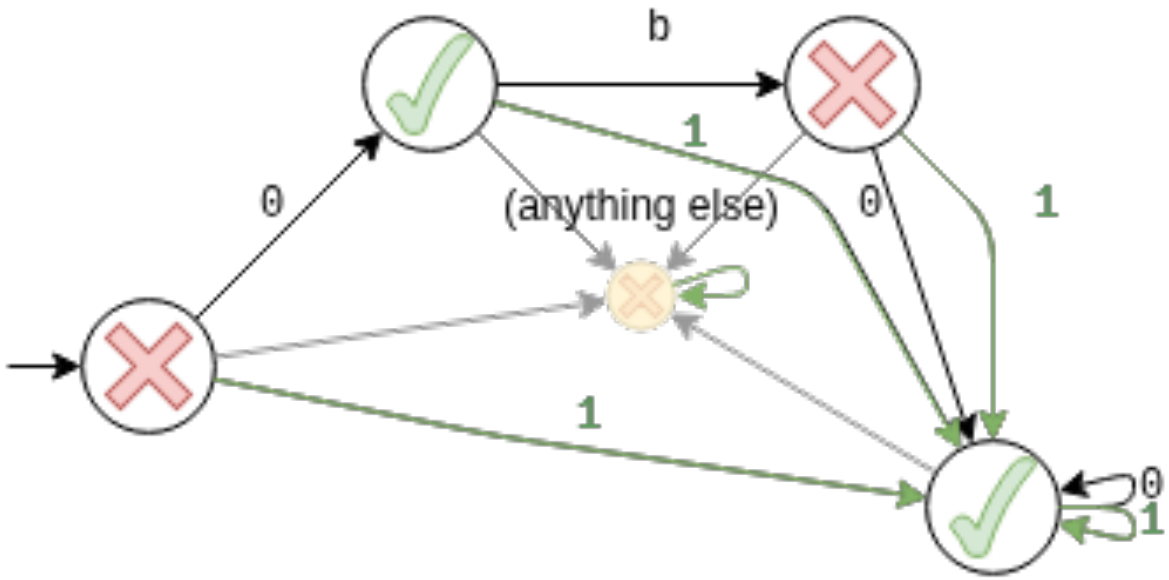
```
(0b)?(0|1)+
```




```
0b110100,|10
```

More Moore Machine Matching

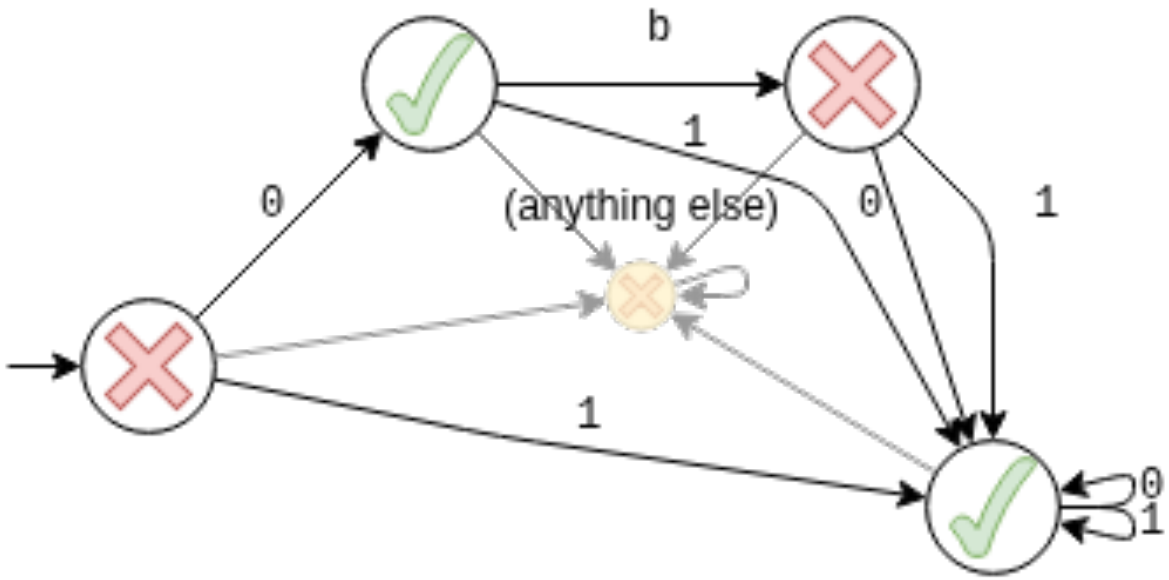
```
(0b)?(0|1)+
```




```
0b110100,[10
```

More Moore Machine Matching

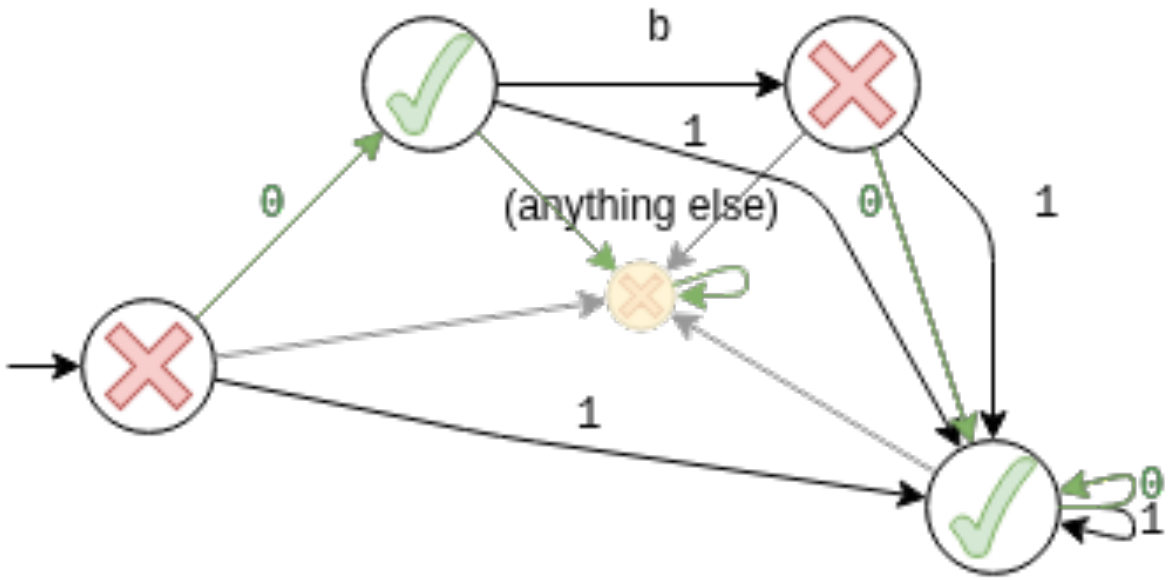
```
(0b)?(0|1)+
```




```
0b110100,1|0
```

More Moore Machine Matching

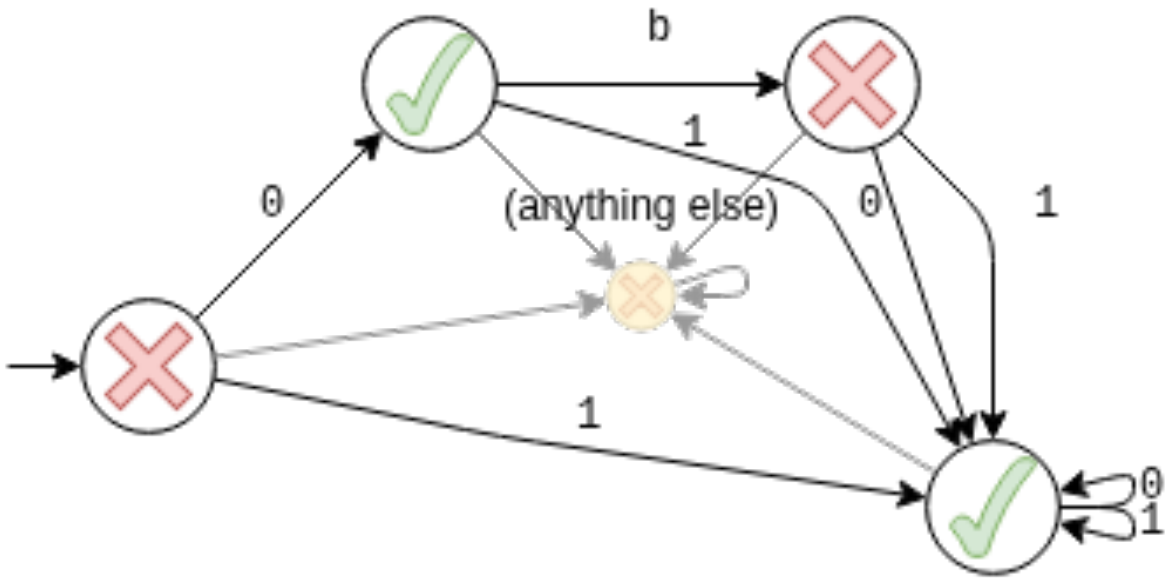
```
(0b)?(0|1)+
```




```
0b110100,1[0
```

More Moore Machine Matching

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




```
0b110100,10|
```

Making Matching Moore Machines

co(bra|d)

Making Matching Moore Machines

co(bra|d)

Making Matching Moore Machines

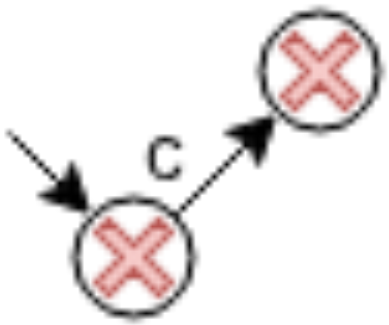
co(bra|d)

Making Matching Moore Machines

co(bra|d)

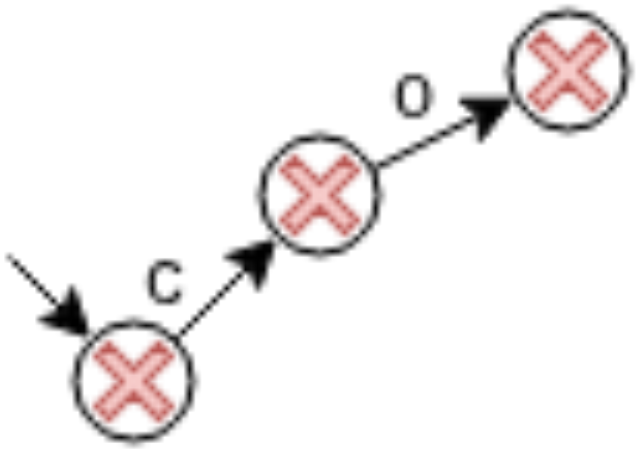
Making Matching Moore Machines

`co(bra|d)`



Making Matching Moore Machines

`co(bra|d)`

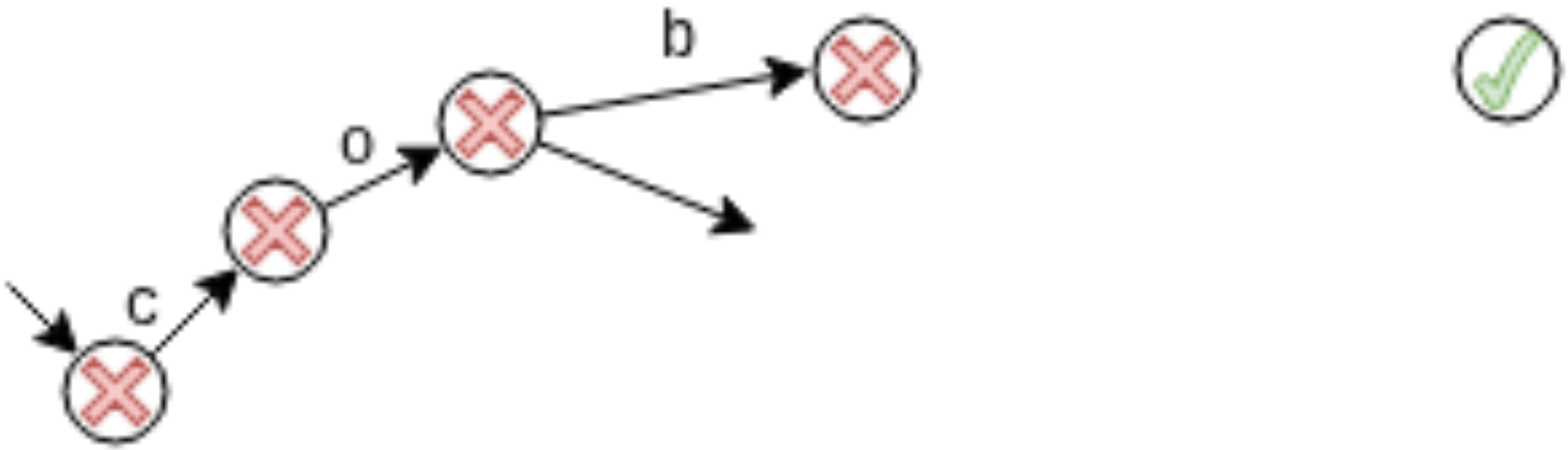


Making Matching Moore Machines

`co(bra|d)`

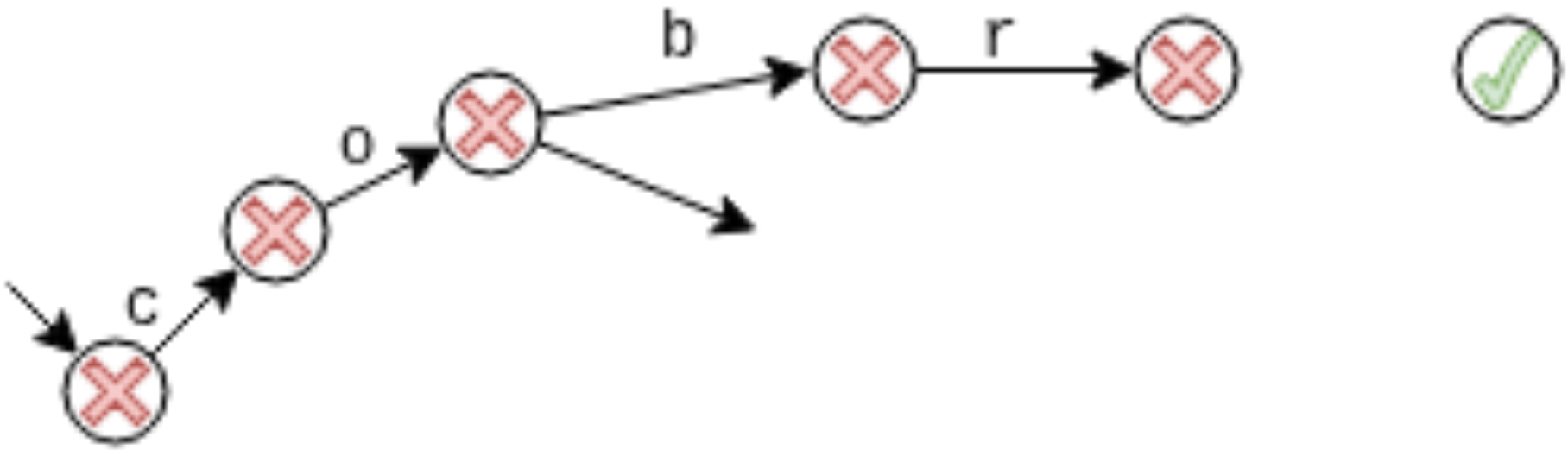
Making Matching Moore Machines

`co(bra|d)`



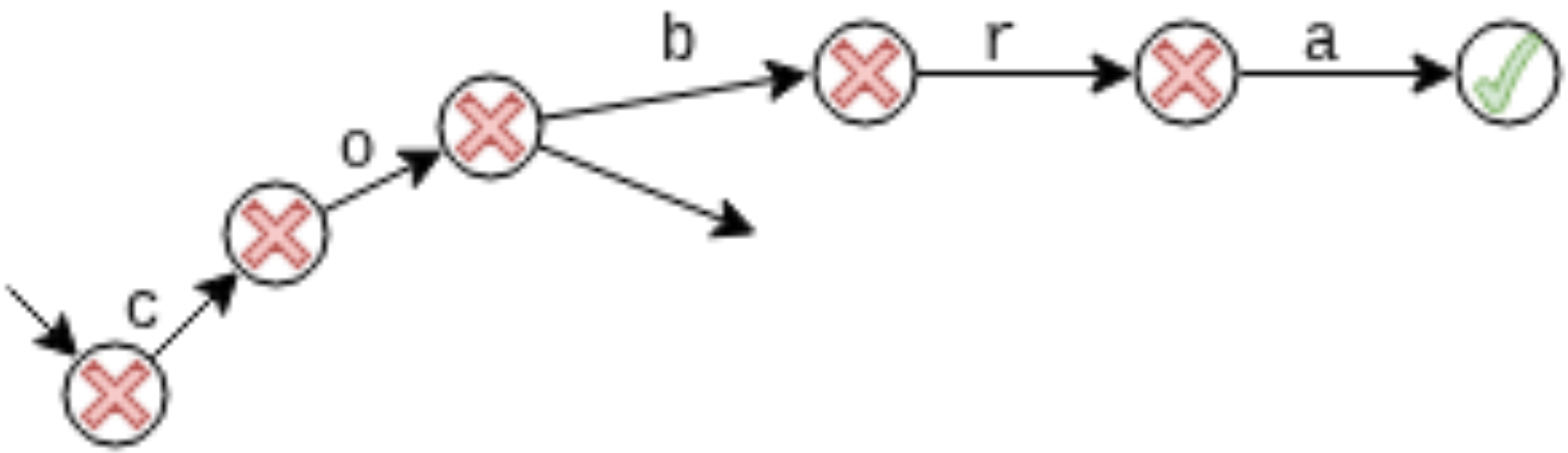
Making Matching Moore Machines

`co(bra|d)`



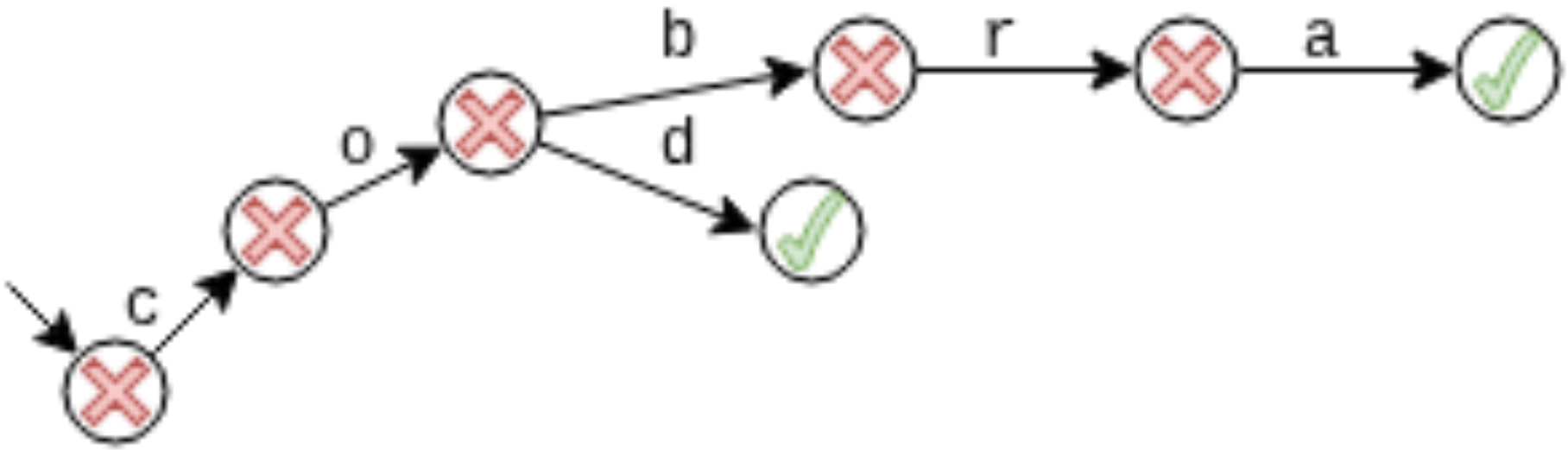
Making Matching Moore Machines

`co(bra|d)`



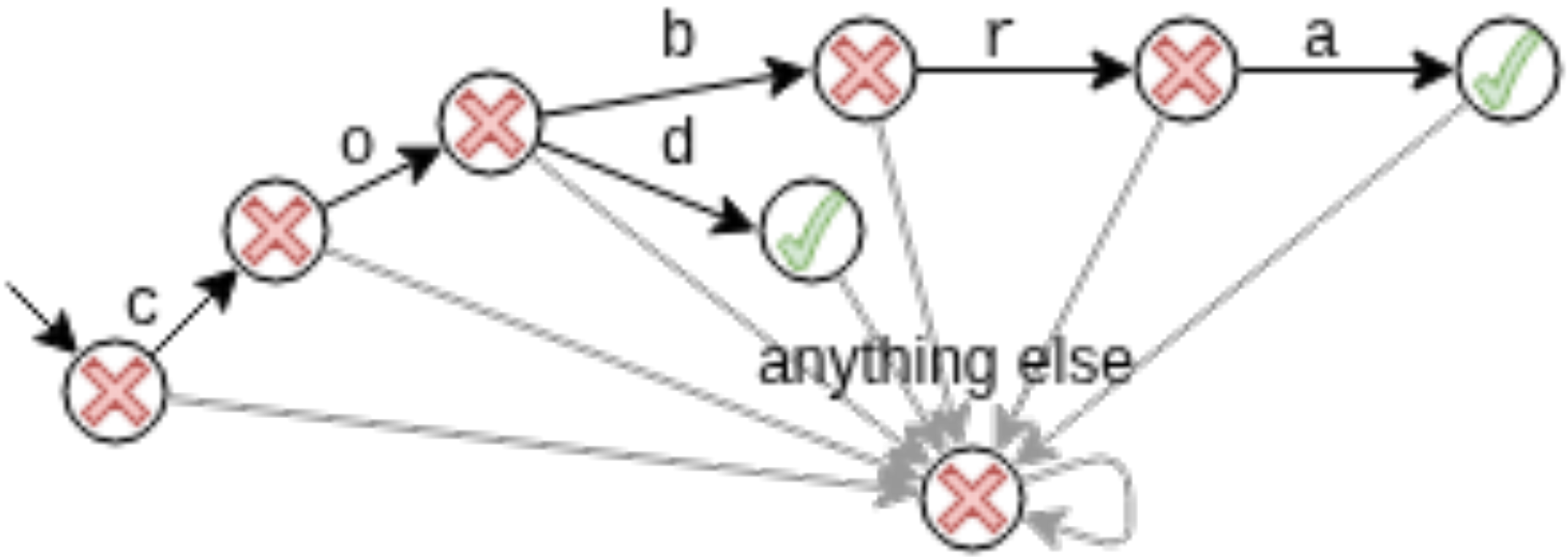
Making Matching Moore Machines

`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

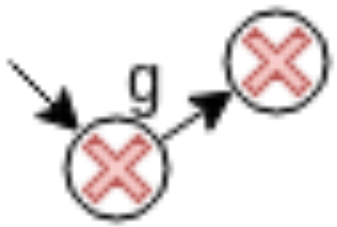
gr(a|e)y|green

More Making Matching Moore Machines

gr(a|e)y|green

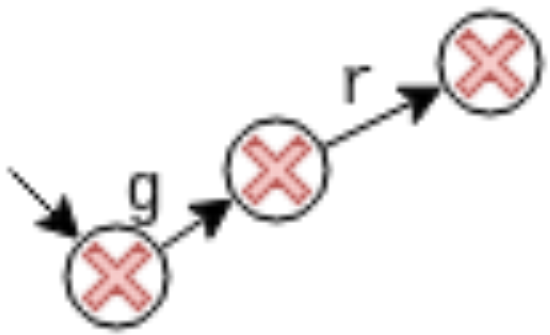
More Making Matching Moore Machines

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



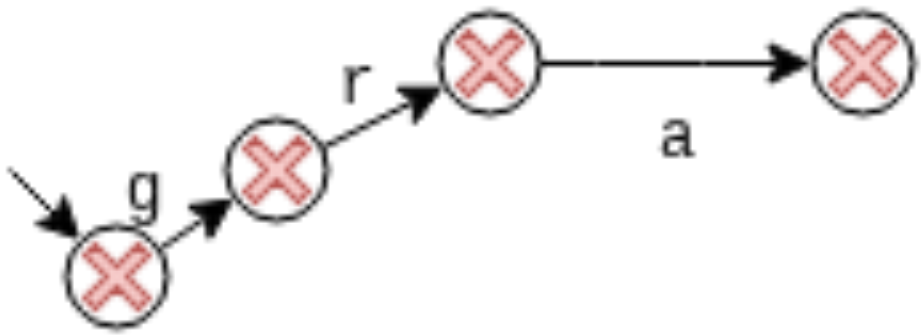
More Making Matching Moore Machines

gr(a|e)y|green



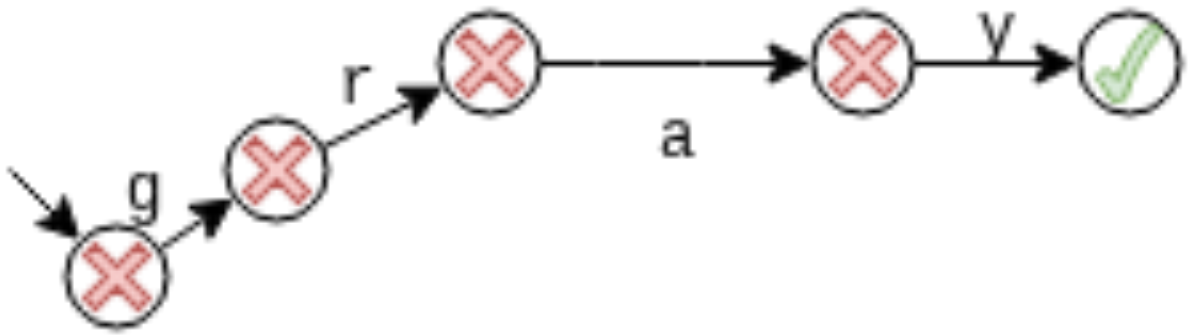
More Making Matching Moore Machines

gr(a|e)y|green



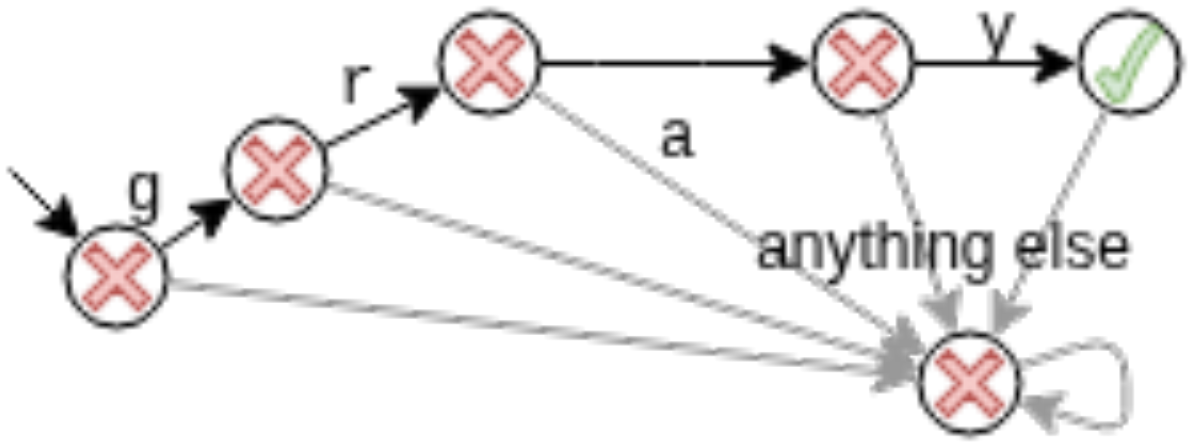
More Making Matching Moore Machines

`gr(a|e)y|green`



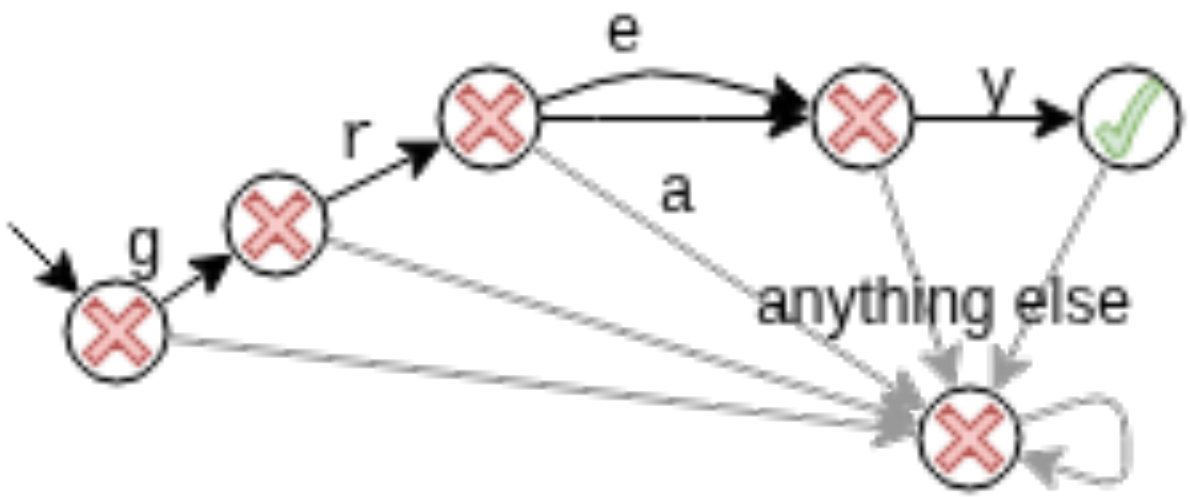
More Making Matching Moore Machines

`gr(a|e)y|green`



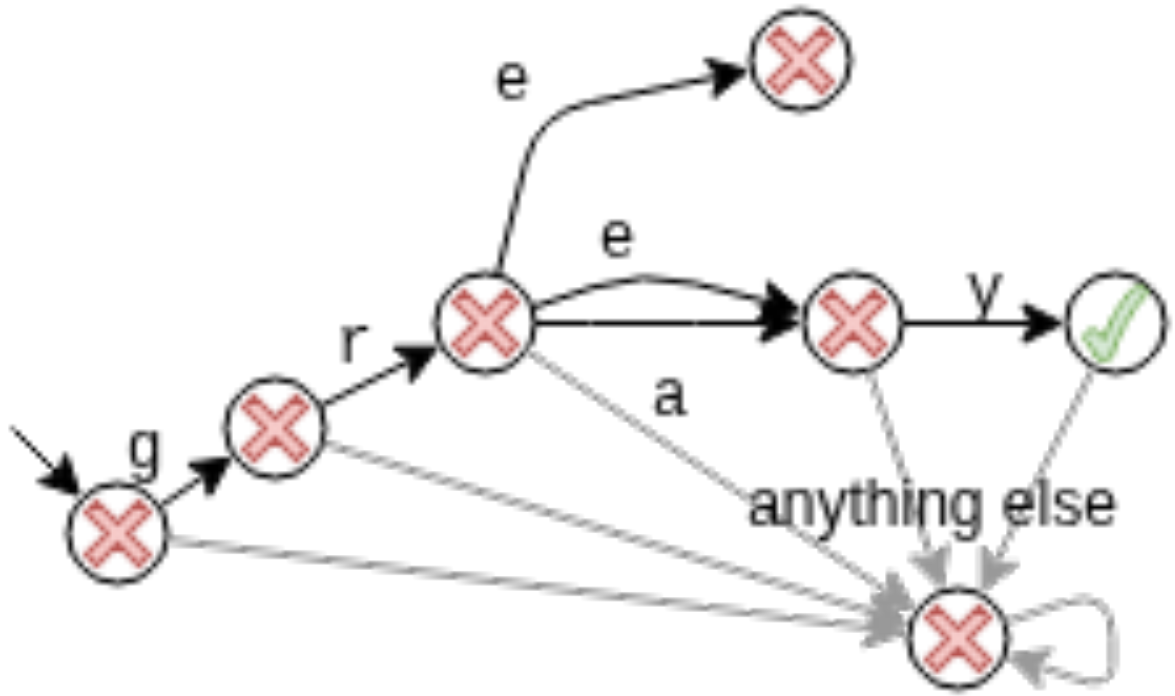
More Making Matching Moore Machines

`gr(a|e)y|green`



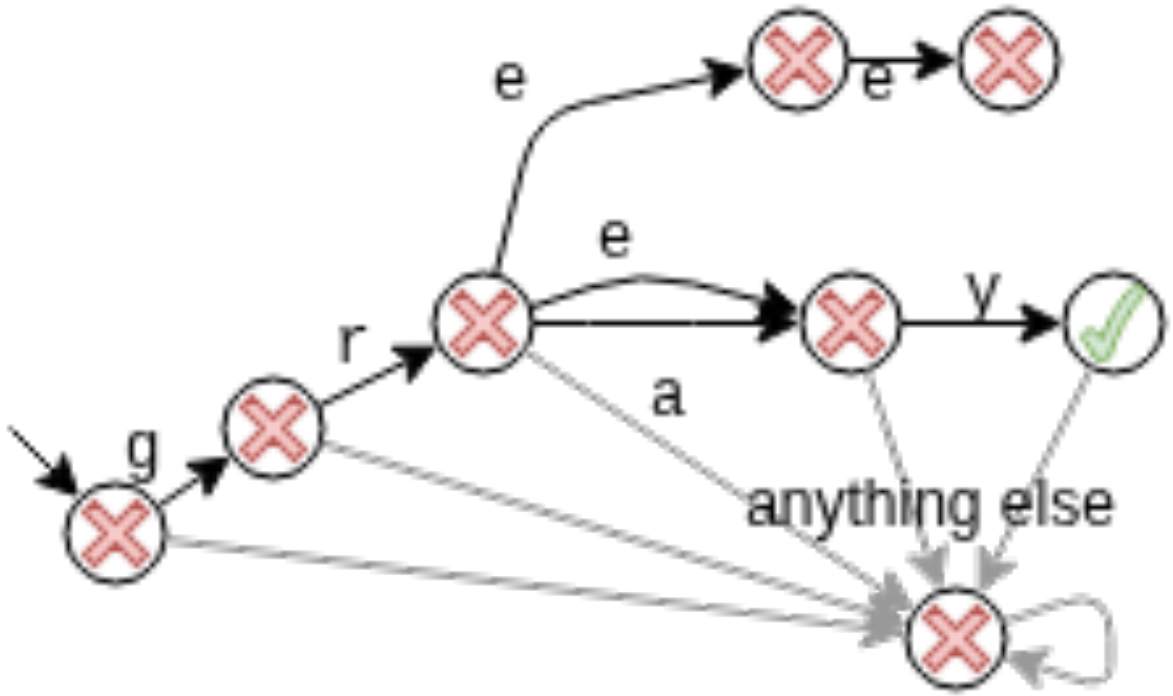
More Making Matching Moore Machines

`gr(a|e)y|green`



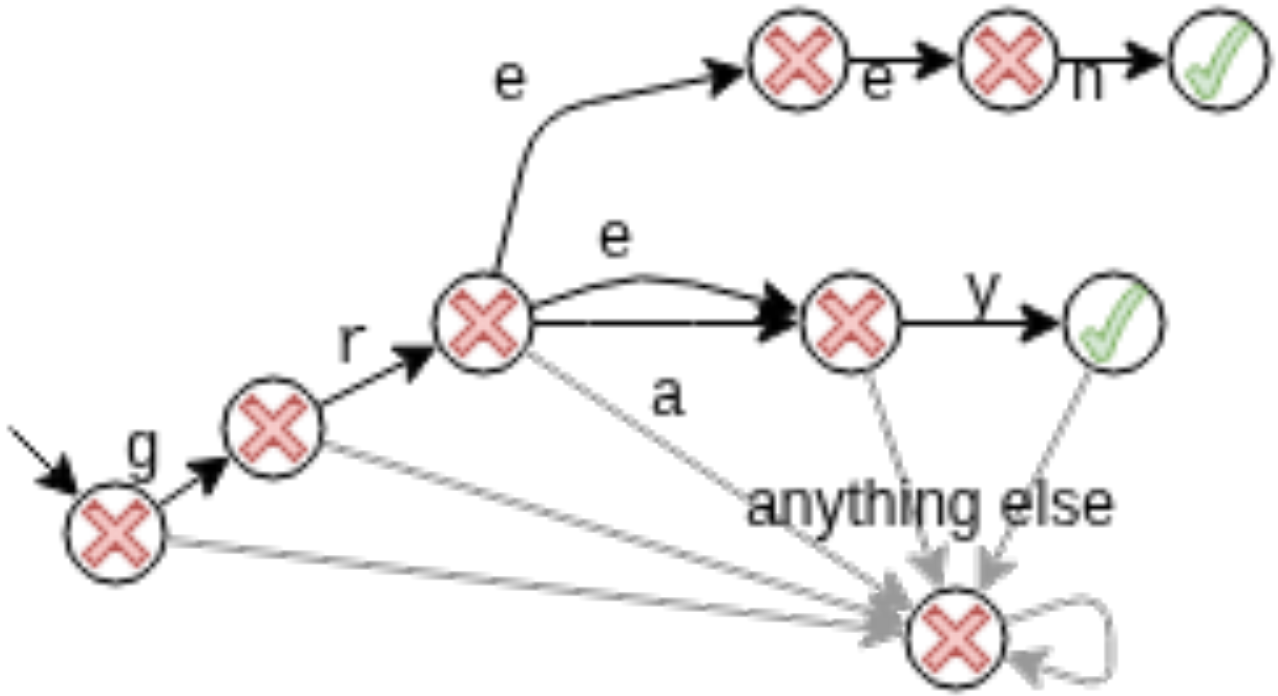
More Making Matching Moore Machines

gr(a|e)y|green



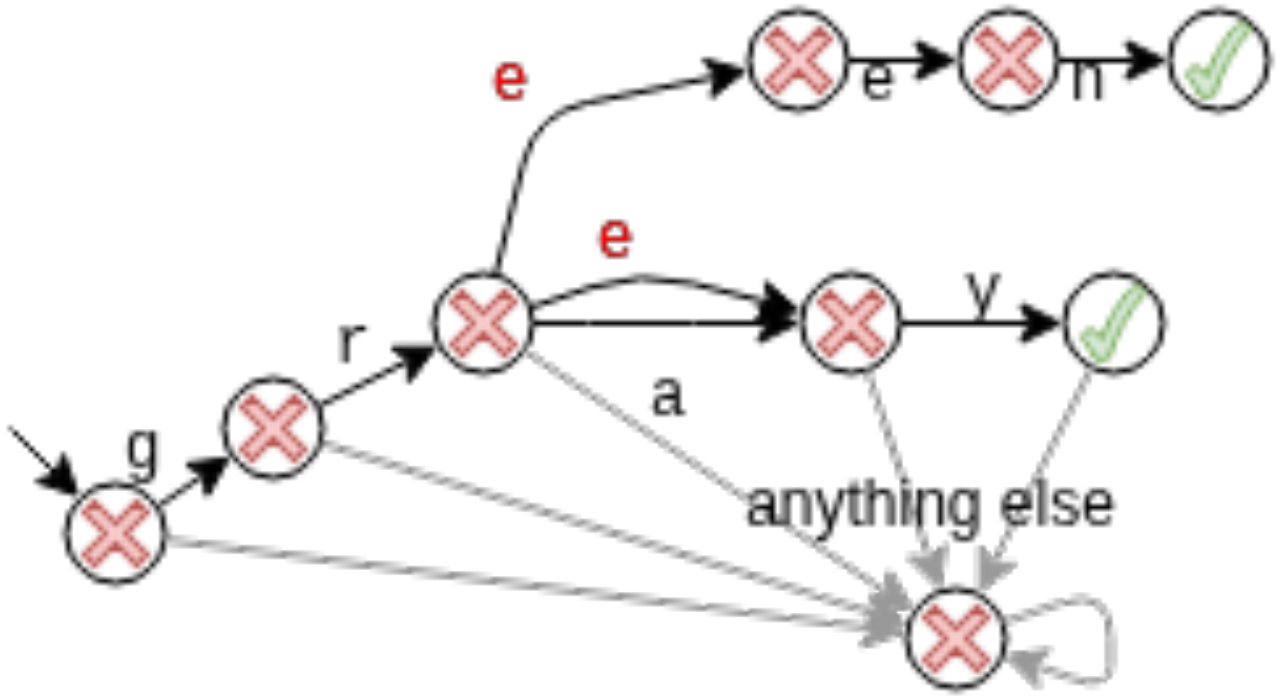
More Making Matching Moore Machines

`gr(a|e)y|green`



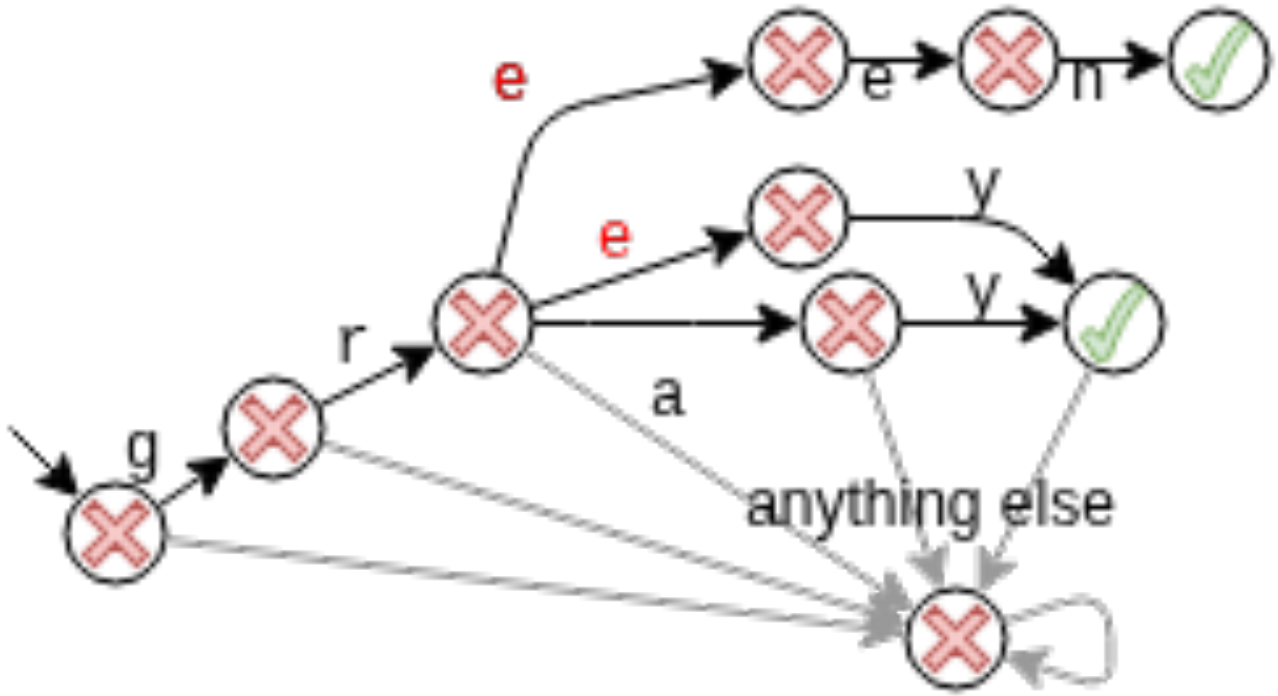
More Making Matching Moore Machines

gr(a|e)y|green



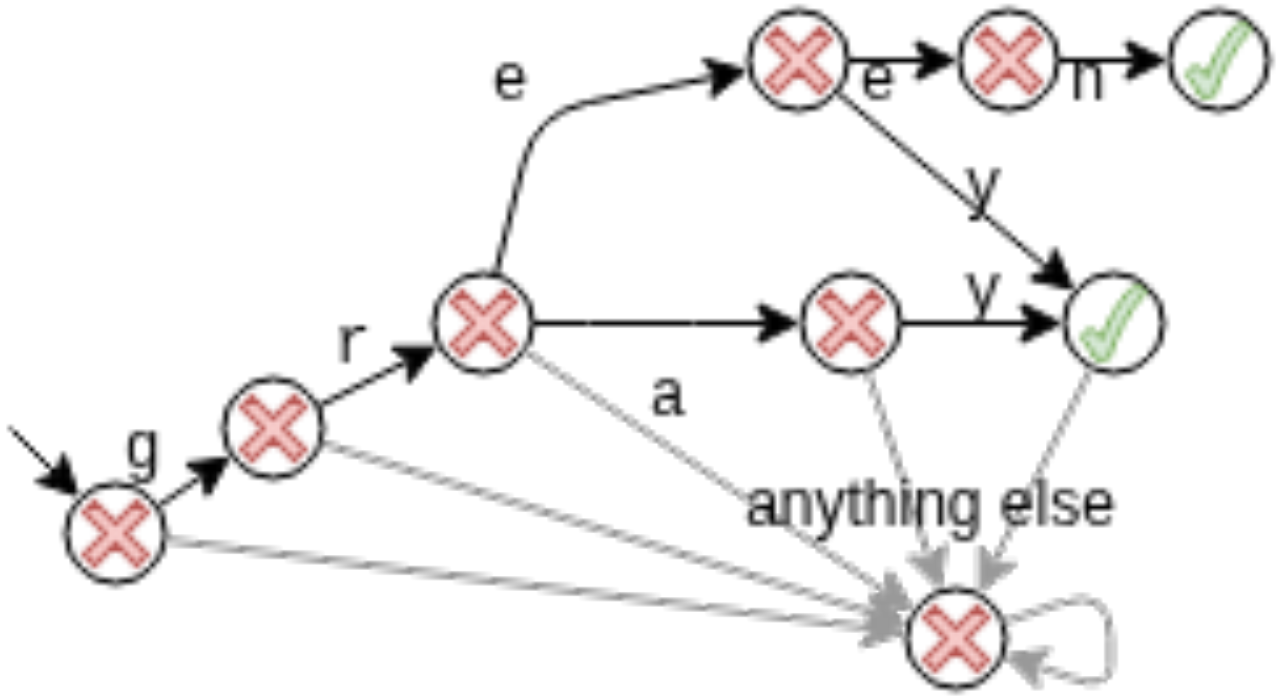
More Making Matching Moore Machines

gr(a|e)y|green



More Making Matching Moore Machines

`gr(a|e)y|green`

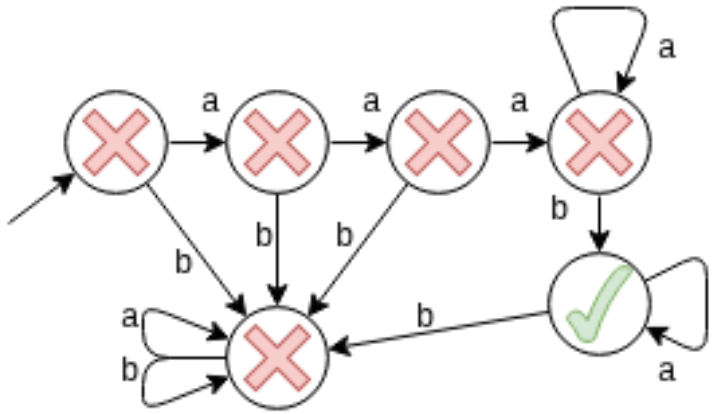


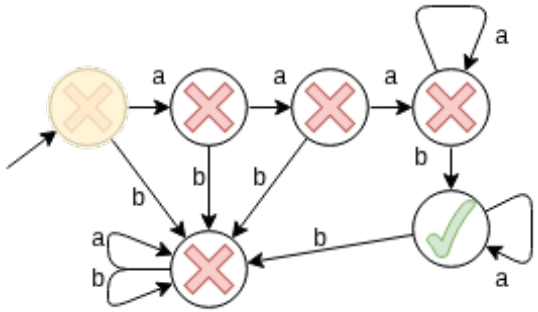
Quiz

wooclap.com XDYJSD

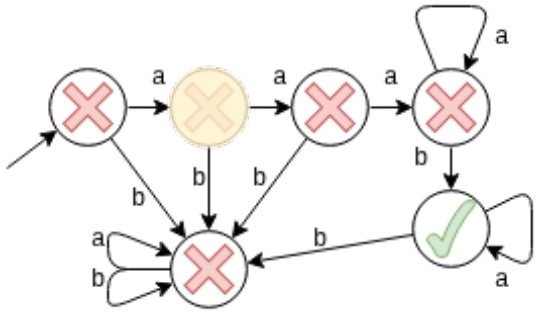
Recap

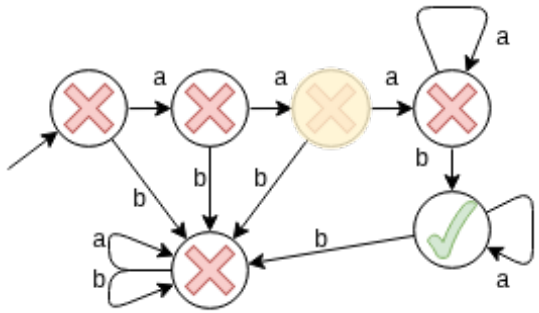
- a*aaaba* ⇝





~[a]↠





~[a]↠

~[a]↠

Running Moore Machines, 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

```haskell
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

```haskell
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
-- Looks familiar...
```

Running Moore Machines, generally

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

Compiling RegExp to DFA

Compiling RegExp to DFA

C

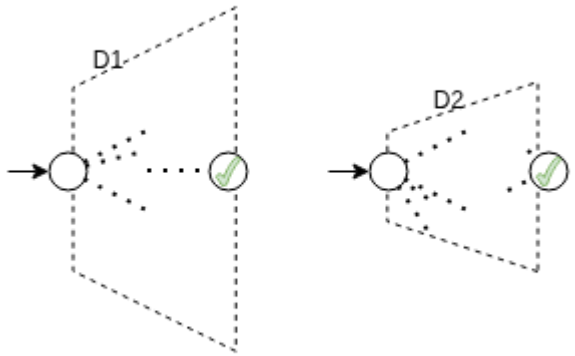
Compiling RegExp to DFA

\d

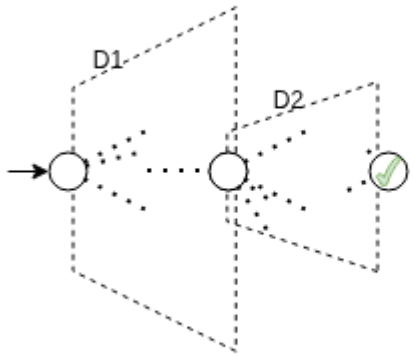
Compiling RegExp to DFA

`[x-z]`

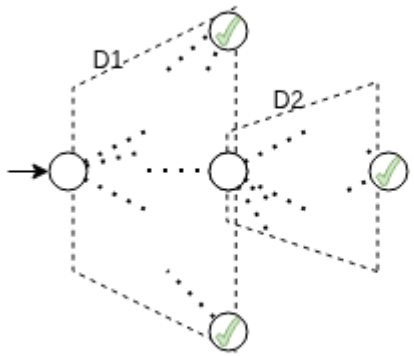
Compiling RegExp to DFA



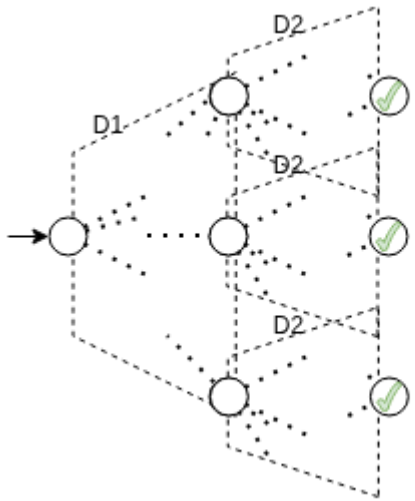
Compiling RegExp to DFA



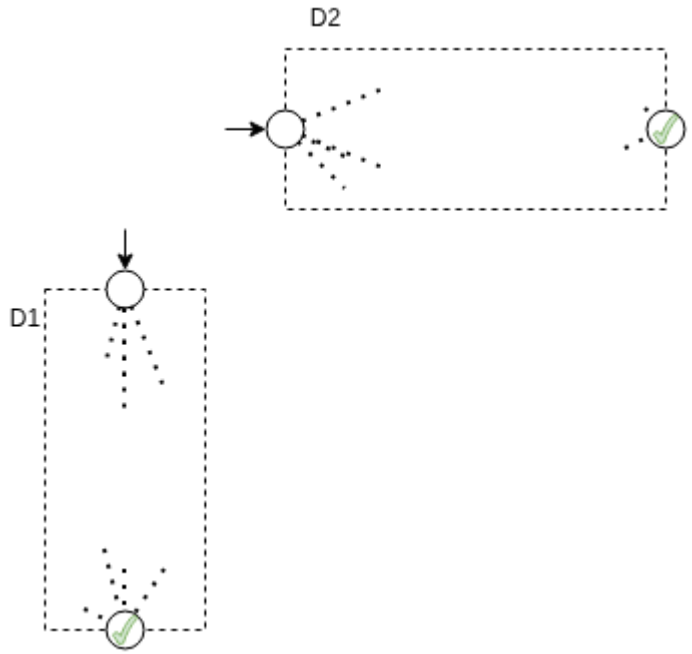
Compiling RegExp to DFA



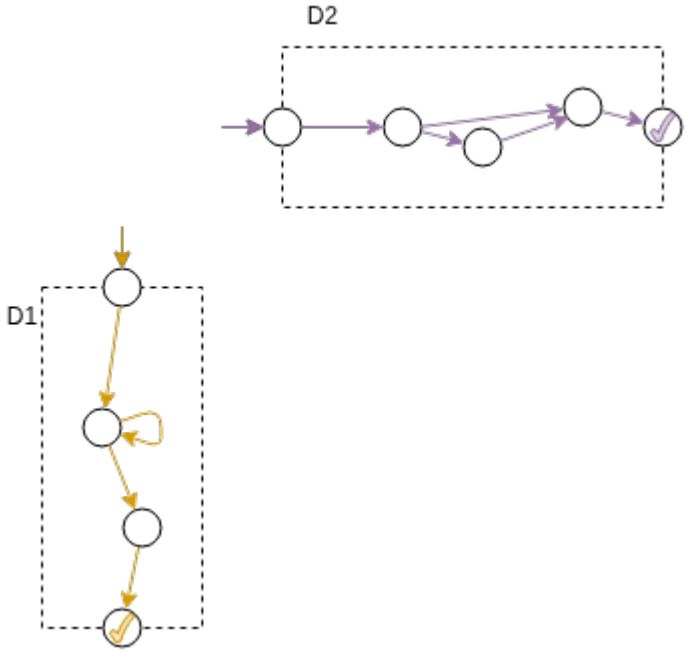
Compiling RegExp to DFA



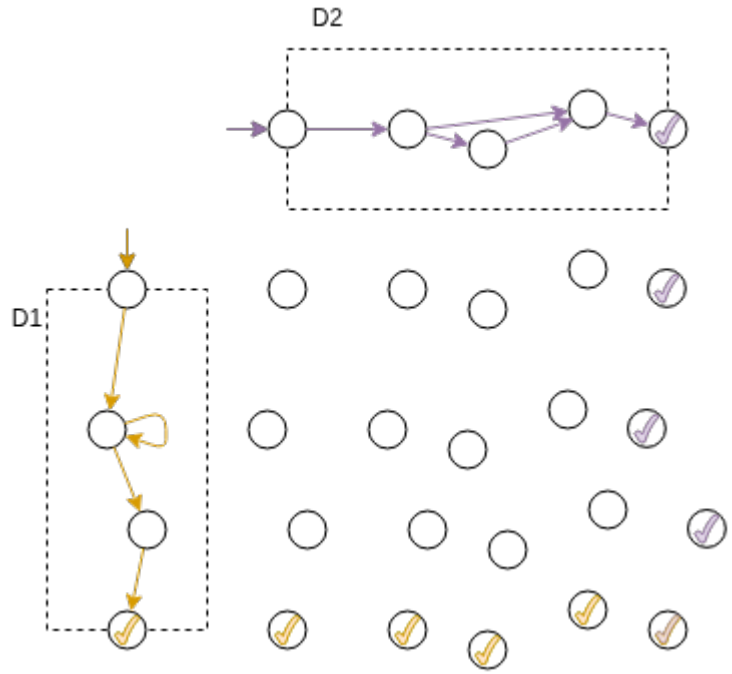
Compiling RegExp to DFA



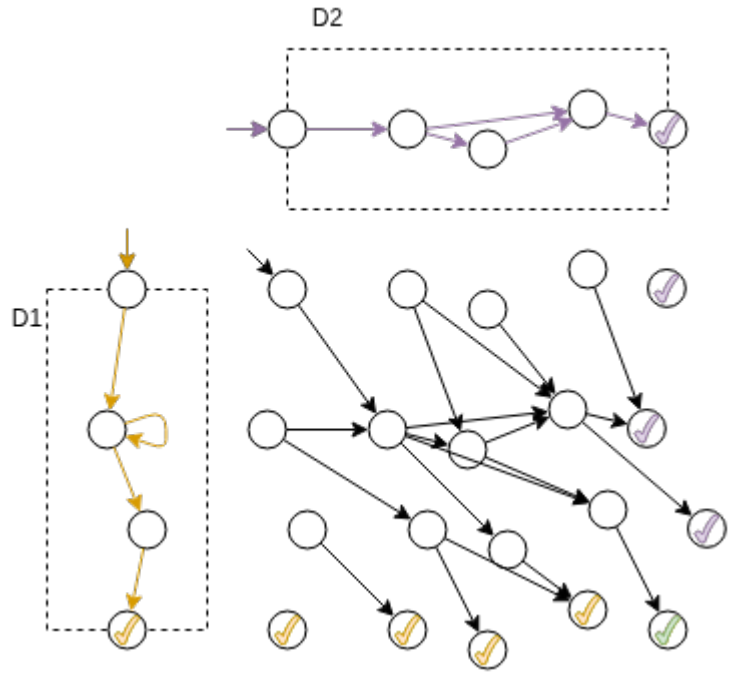
Compiling RegExp to DFA



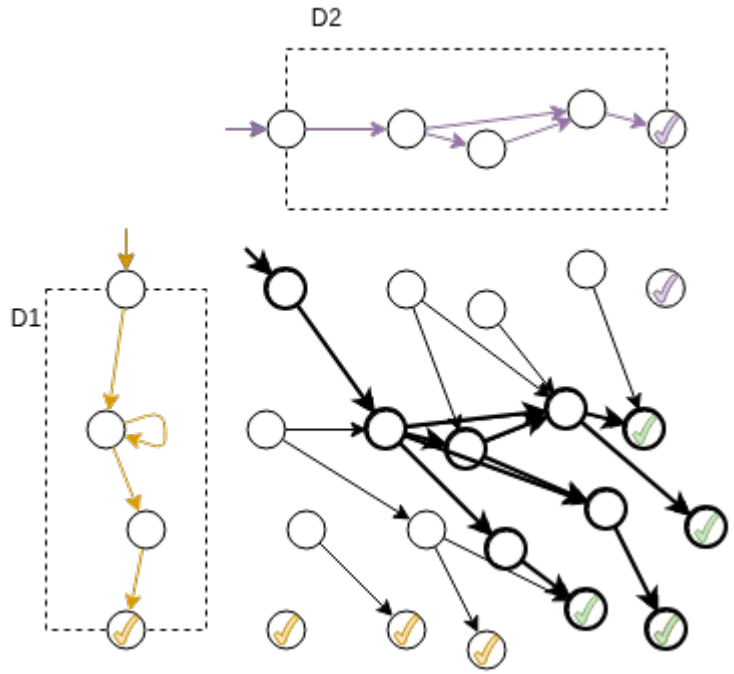
Compiling RegExp to DFA



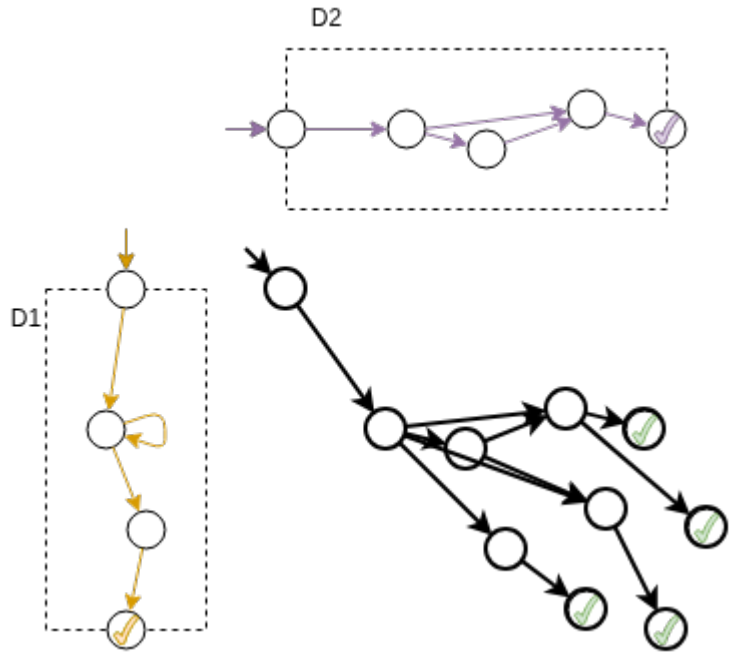
Compiling RegExp to DFA



Compiling RegExp to DFA



Compiling RegExp to DFA



Compiling RegExp to DFA

Compiling RegExp to DFA

❓

Compiling RegExp to DFA

r*

Compiling RegExp to DFA

r?

Compiling RegExp to DFA progress

✅ `c`

✅ `\d`

✅ `[x-z]`

💣 $r_1 r_2$

💣 $r_1 | r_2$

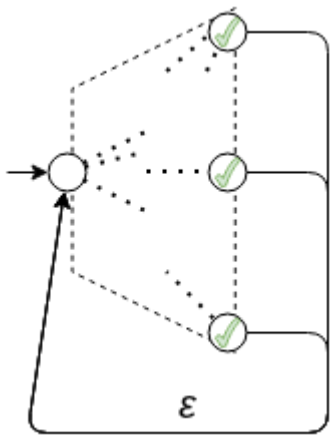
❓ `r+`

❓ `r*`

❓ r?

Compiling RegExp to NFAε

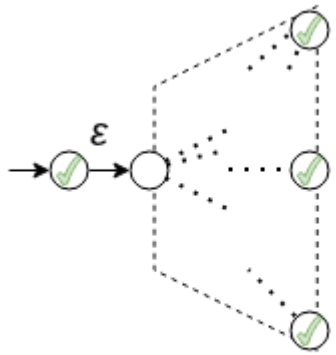
Compiling RegExp to NFAε



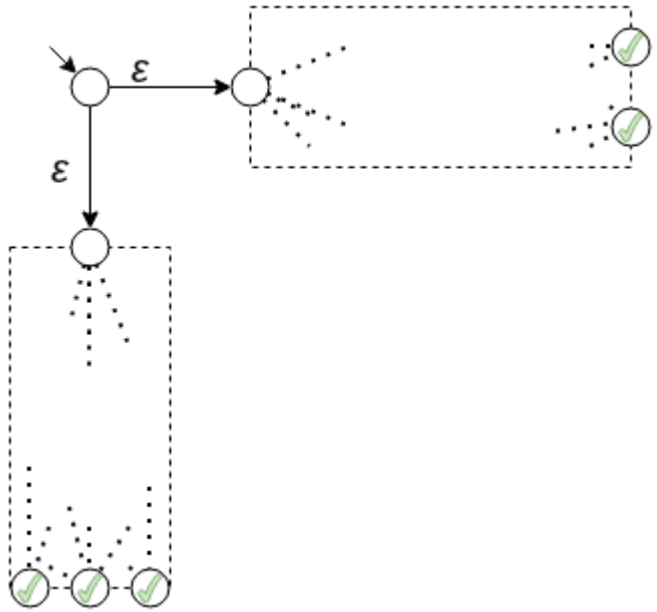
Compiling RegExp to NFAε

r*

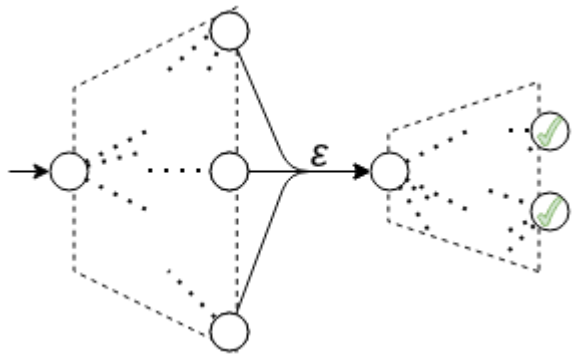
Compiling RegExp to NFAε



Compiling RegExp to NFAε



Compiling RegExp to NFAε



Compiling RegExp to NFAε progress

- ✅ c
- ✅ \d
- ✅ [x-z]
- ✅ $r_1 r_2$
- ✅ $r_1 | r_2$
- ✅ r+
- ✅ r*

✅ r?

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))
```

Putting together the parts

✅

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

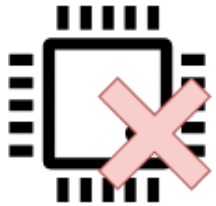
✅

```
r2n :: RegExp → NFAε Char Label -- (by ex
```

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

Not done yet!





Functions seen so far

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

runDFA  :: DFA  sy      st  → [sy] →     st
```

A special case of runDFA

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

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

Convert NFAε to DFA?

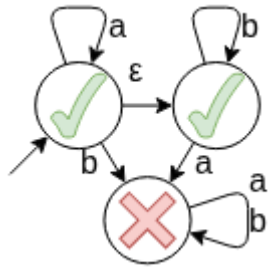
```
runNFAε :: NFAε sy       st  → [sy] → Set st
runNFAε = runDFA . n2d

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

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

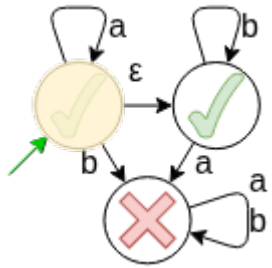
Another NFAε simulation

Another NFAε simulation



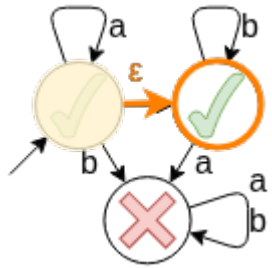
aaba

Another NFAε simulation



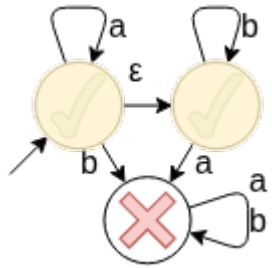
]aaba

Another NFAε simulation



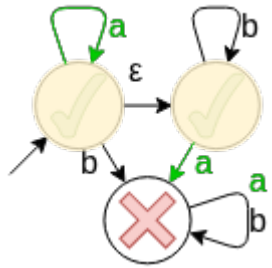
!aaba

Another NFAε simulation



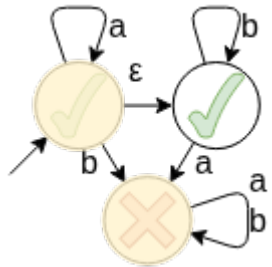
|aaba

Another NFAε simulation



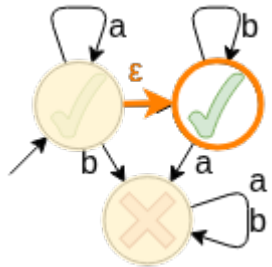
[aaba

Another NFAε simulation



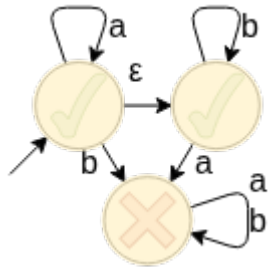
a]aba

Another NFAε simulation



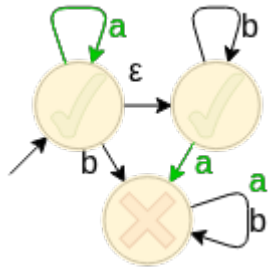
a!aba

Another NFAε simulation



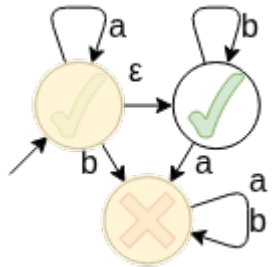
a|aba

Another NFAε simulation



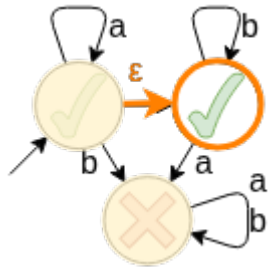
a[aba

Another NFAε simulation



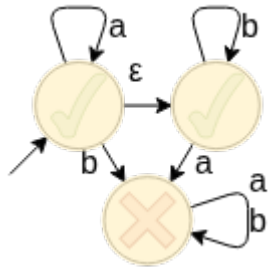
aa]ba

Another NFAε simulation



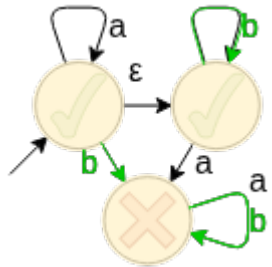
aa!ba

Another NFAε simulation



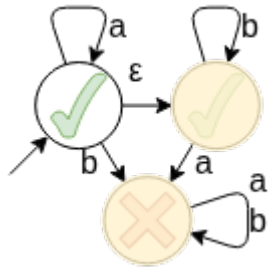
aa|ba

Another NFAε simulation



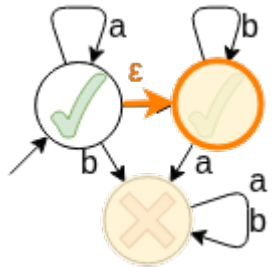
aa[ba

Another NFAε simulation



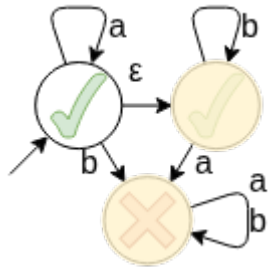
aab]a

Another NFAε simulation



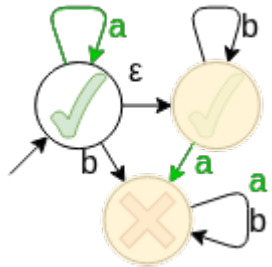
aab!a

Another NFAε simulation



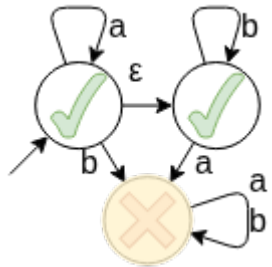
aab|a

Another NFAε simulation



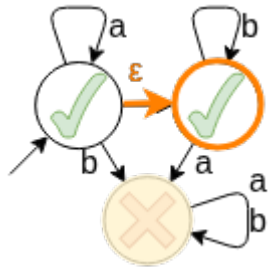
aab[a

Another NFAε simulation



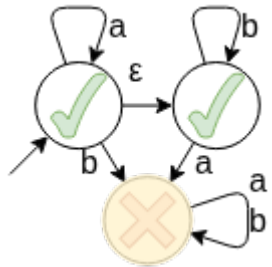
aaba]

Another NFAε simulation



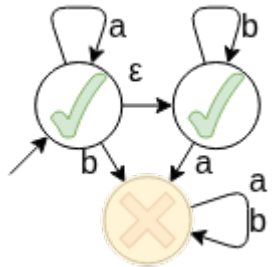
aaba!

Another NFAε simulation

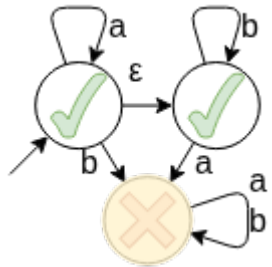


aaba|

Another NFAε simulation

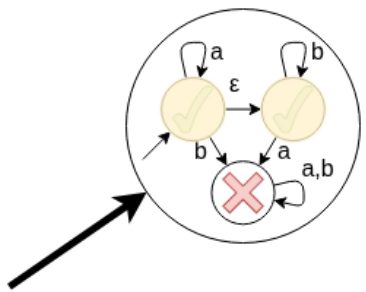


Another NFAε simulation

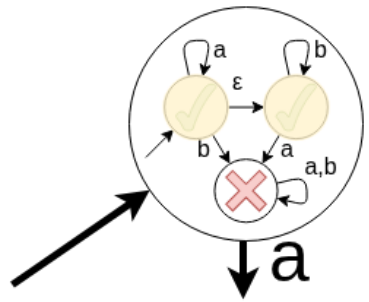


The subset construction

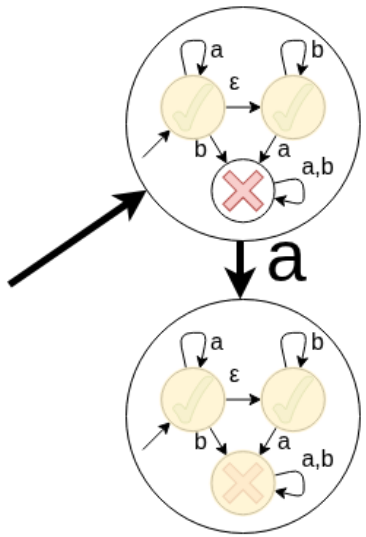
The subset construction



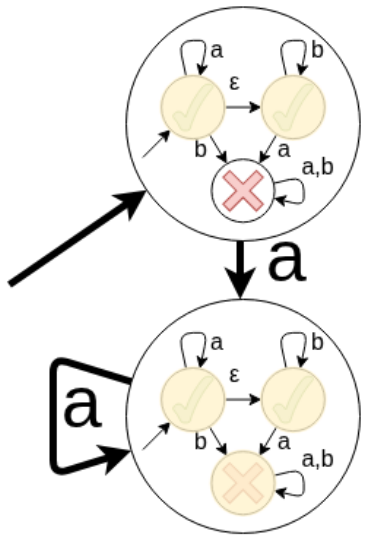
The subset construction



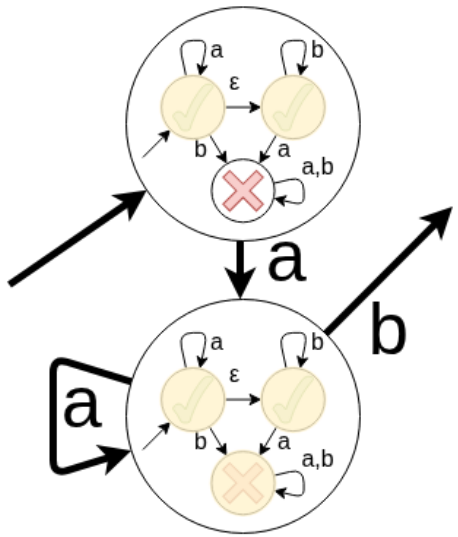
The subset construction



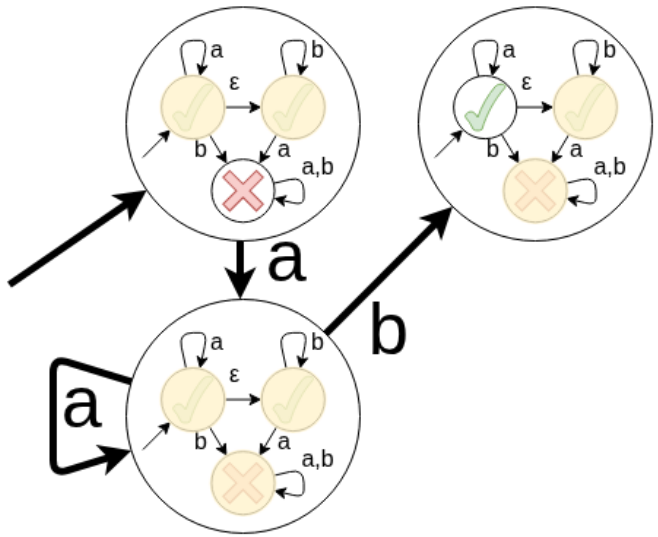
The subset construction



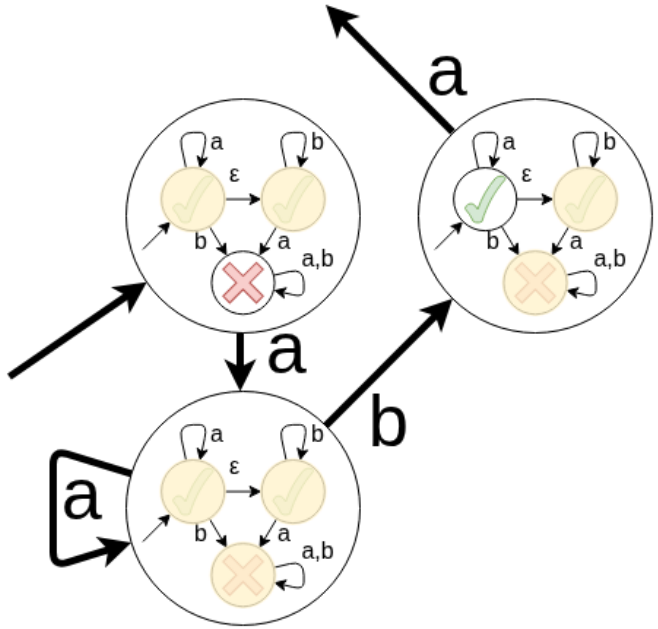
The subset construction



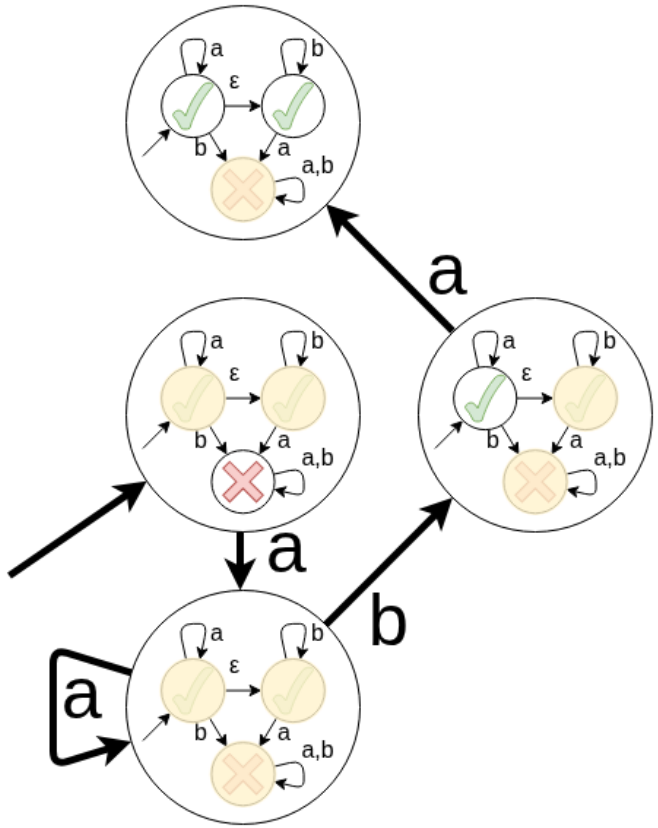
The subset construction



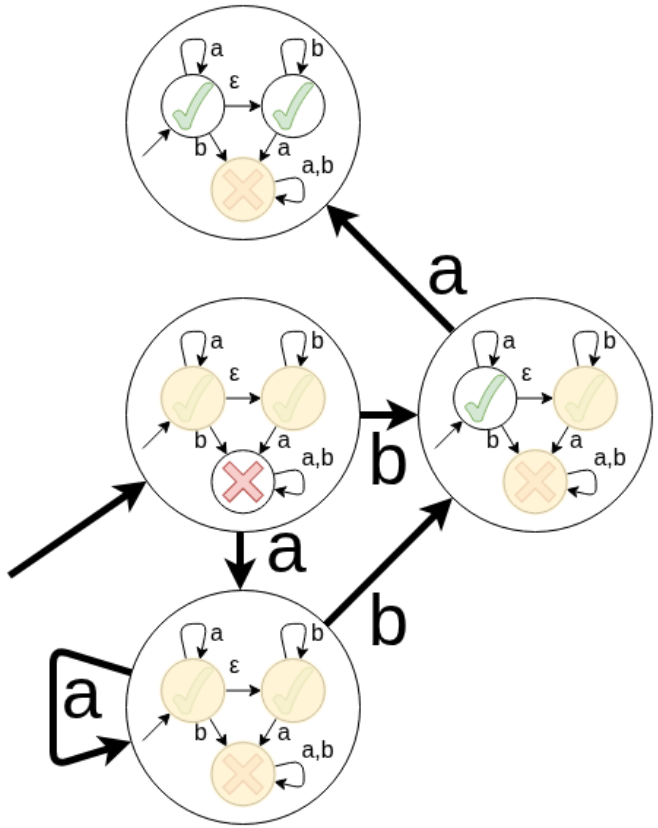
The subset construction



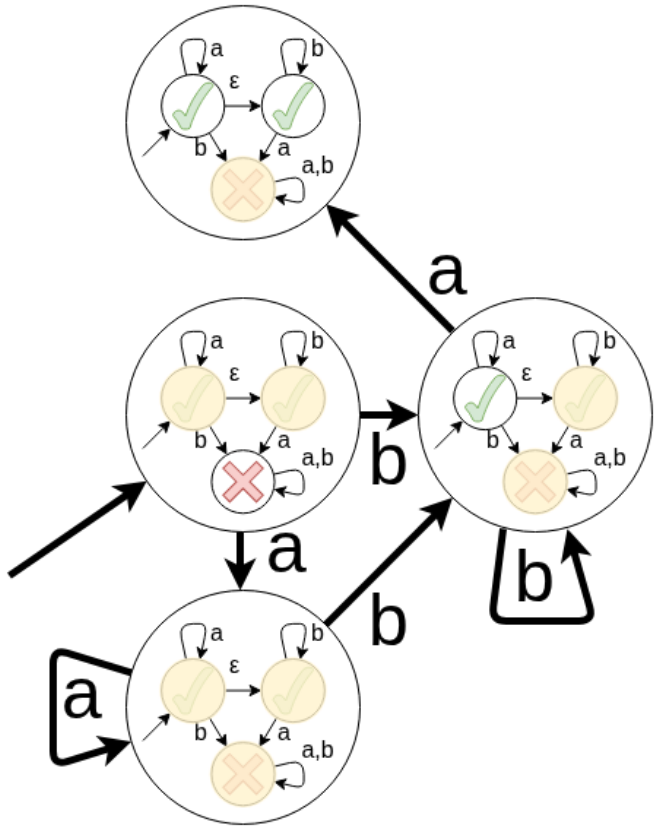
The subset construction



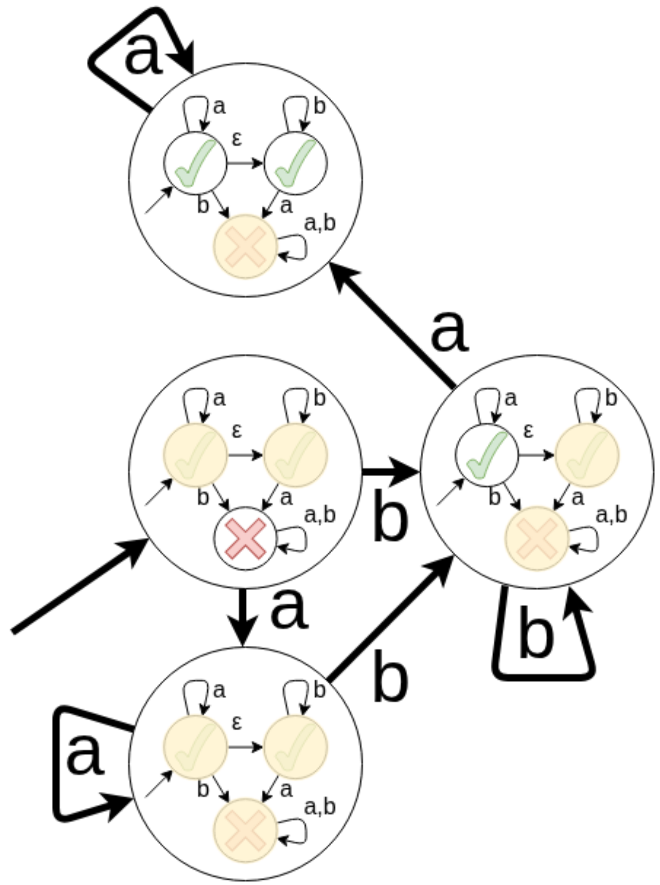
The subset construction



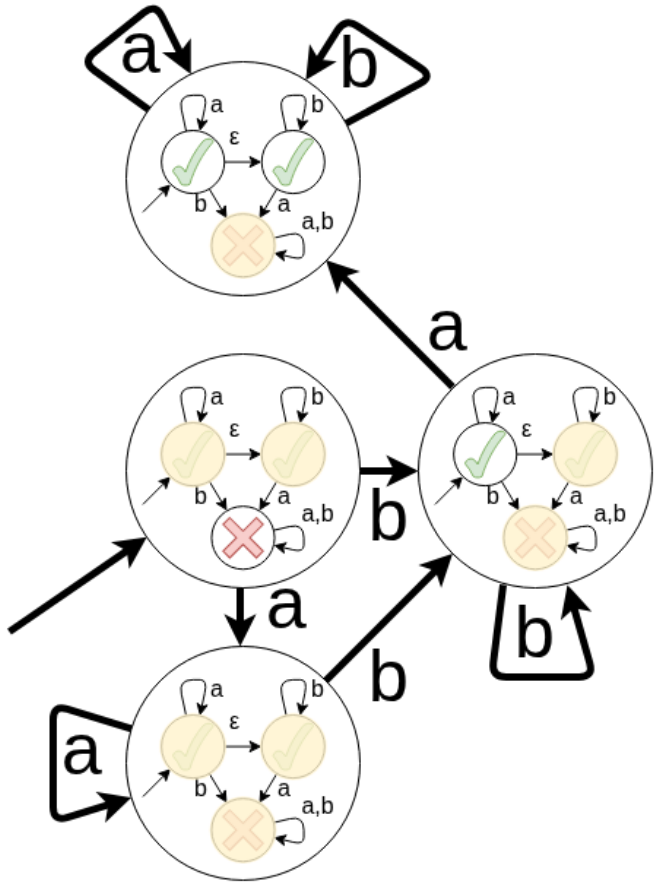
The subset construction



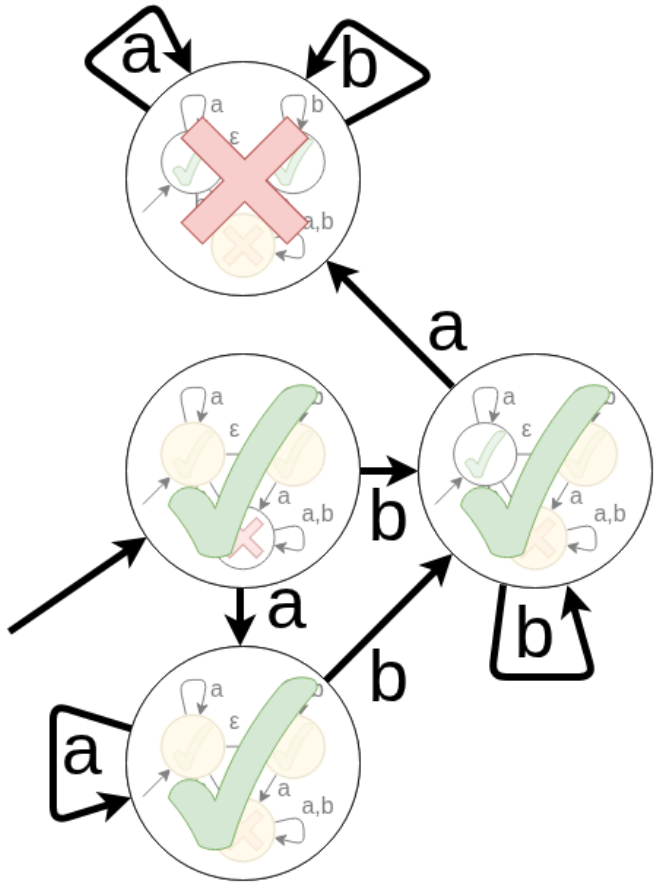
The subset construction



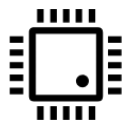
The subset construction



The subset construction



Summary

- RegExp 🚀
- RegExp → NFAε
- NFAε → DFA
- DFA →   
- *Challenge* : implement in Haskell