Determinism, Electricity, and Intuitionism

Gérard Berry

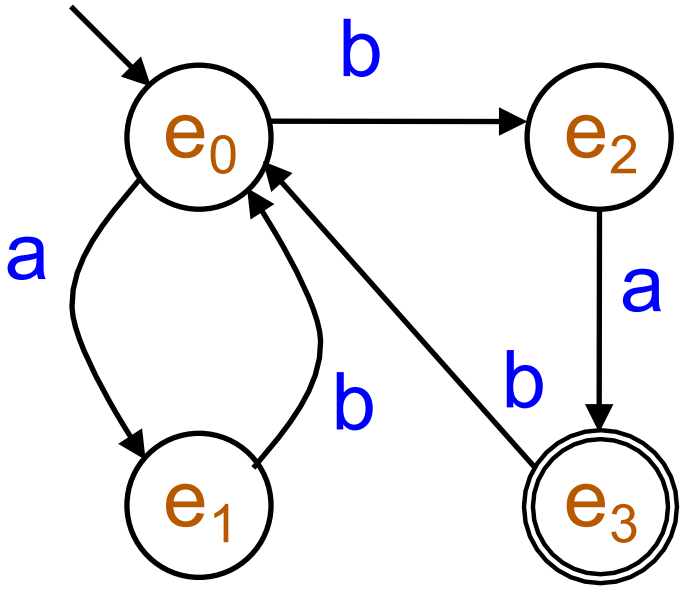
http://www.college-de-france.fr/site/gerard-berry/index.htm
http://www-sop.inria.fr/members/Gerard.Berry/

Collège de France
Algorithms, Machines, and Languages Chair

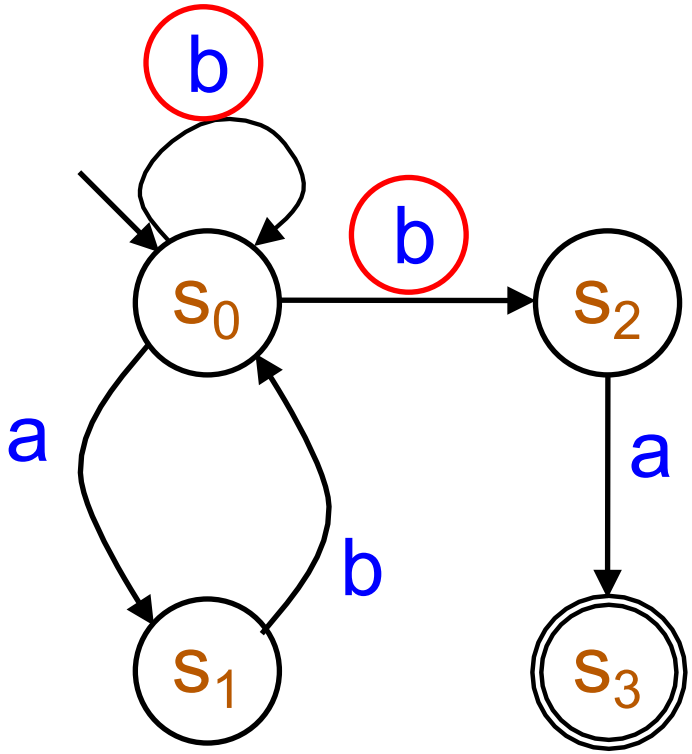
Bordeaux, March 24th, 2016

COLLÈGE
DE FRANCE
1530

Recognizing Automata



Deterministic



Non-deterministic

(w.r.t. s_0 and b)

Derivatives of Languages

$L \rightarrow$ ba, abba, bba, abbabba, bbbbabababbbbba,...

$a^{-1}(L) \rightarrow$ ba, abba, bba, abbabba, bbbbabababbbbba,...

$b^{-1}(L) \rightarrow$ ba, abba, bba, abbabba, bbbbabababbbbba,...

$u^{-1}(L) = \{ v \mid uv \in L \}$ $\quad ua^{-1}(L) = a^{-1}(u^{-1}(L))$

what remains to be written after writing u

Derivatives of Regular Expressions

$a^{-1}(e)$ = regular expression generating $a^{-1}(L(e))$

- $a^{-1}(0) = 0$
- $a^{-1}(1) = 0$
- $a^{-1}(b) = 0$ if $b \neq a$
- $a^{-1}(a) = 1$
- $a^{-1}(e + e') = a^{-1}(e) + a^{-1}(e')$
- $a^{-1}(e \cdot e') = a^{-1}(e) \cdot e' + \varepsilon(e) \cdot a^{-1}(e')$
- $a^{-1}(e^*) = a^{-1}(e) \cdot e^*$
- by words : $\varepsilon^{-1}(e) = e$, $(ua)^{-1}(e) = a^{-1}(u^{-1}(e))$

From Derivatives to Automata

Theorem (Brzozowski) : a regular expression e has at most a finite number of distinct (simplified) derivatives $u^{-1}(e)$

Corollary : one can construct a deterministic finite automaton recognizing L(e) as follows:

- the states are the distinct derivatives of e
- the initial state is e
- there is a-labeled transition from $u^{-1}(e)$ to $ua^{-1}(e)$ if both derivatives are nonempty
- a state e' is final if $\varepsilon(e') = 1$

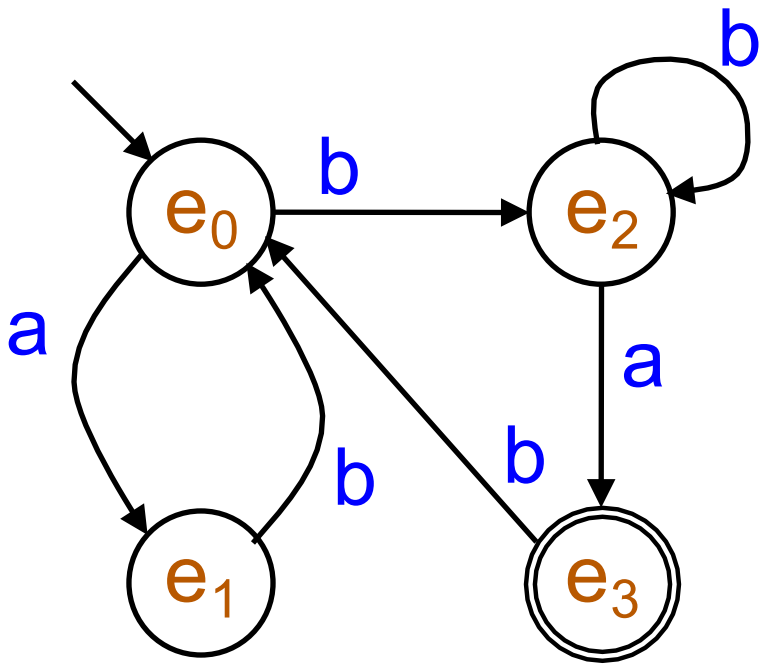
Works for negation, and for any finite-state language by defining its derivatives → first Esterel compiler

Convergent Iterative Process

- $e = e_0 = (ab+b)^*ba$
- $a^{-1}(e_0) = b(ab+b)^*ba = e_1$
- $b^{-1}(e_0) = (ab+b)^*ba + a = e_2$
- $a^{-1}(e_1) = 0$
- $b^{-1}(e_1) = (ab+b)^*ba = e_0$
- $a^{-1}(e_2) = b(ab+b)^*ba + 1 = e_3$
- $b^{-1}(e_2) = (ab+b)^*ba + a = e_2$
- $a^{-1}(e_3) = 0$
- $b^{-1}(e_3) = (ab+b)^*ba = e_0$

From Derivatives to Automata

- $a^{-1}(e_0) = e_1$
- $b^{-1}(e_0) = e_2$
- $a^{-1}(e_1) = 0$
- $b^{-1}(e_1) = e_0$
- $a^{-1}(e_2) = e_3$
- $b^{-1}(e_2) = e_2$
- $a^{-1}(e_3) = 0$
- $b^{-1}(e_3) = e_0$
- $\varepsilon(e_3) = 1$



Deterministic automaton,
but can be exponential !

Exponential Explosion Danger Both Ways !

- The Brzozowski finite automation may be exponentially bigger than e
  try e = •*aa + •*bb + •*cc + …
- For some expressions, this is true for all equivalent deterministic automata (try e above)
- Conversely, for some automata, any equivalent regular expression may be exponentially bigger than the automaton

Linear Expression

- an expression is linear if it contains each letter at most once
- linearize expressions by uniquely indexing letters

$s_0 = (a_0b_1 + b_2)^* b_3a_4$

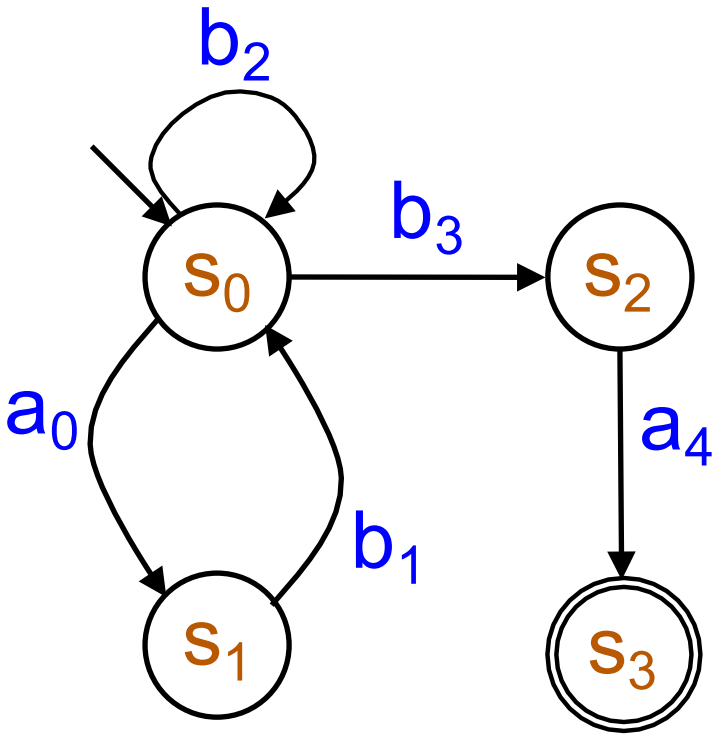
$a_0^{-1}(s_0) = b_1(a_0b_1 + b_2)^* b_3a_4 = s_1$

$b_2^{-1}(s_0) = (a_0b_1 + b_2)^* b_3a_4 = s_0$

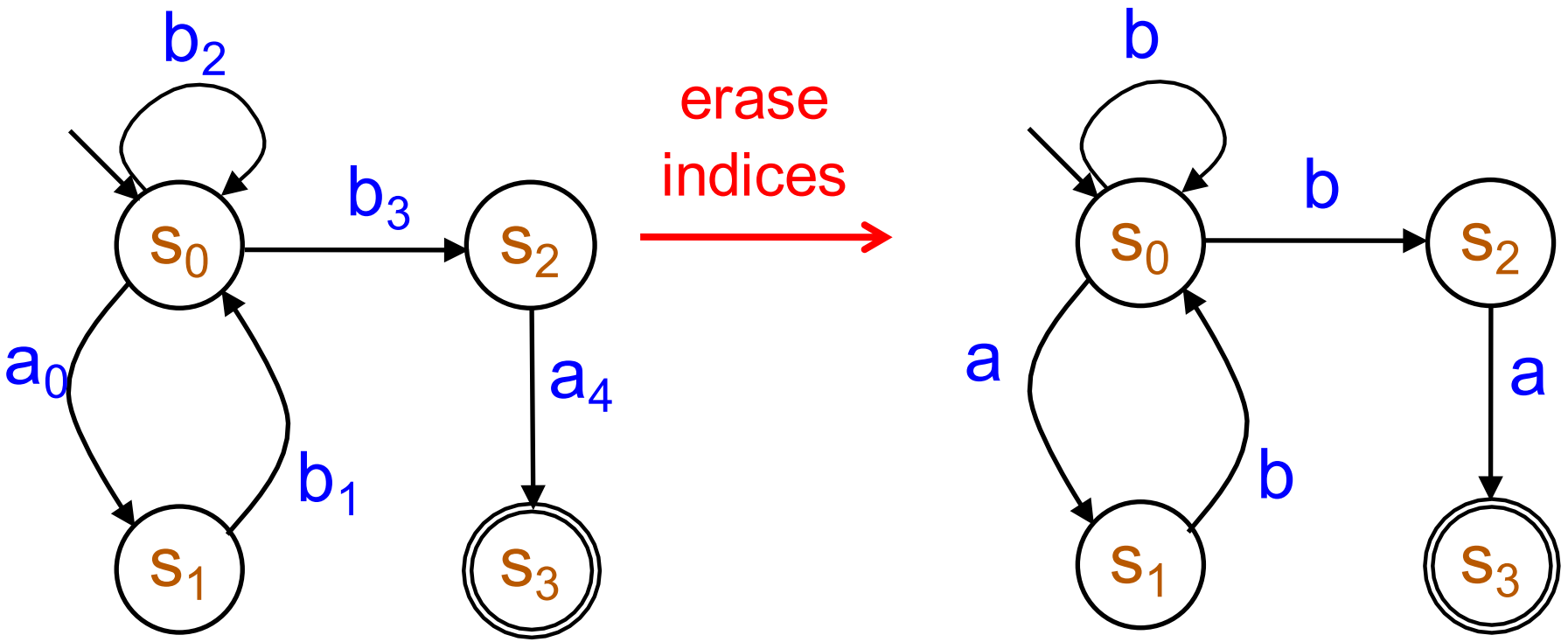
$b_3^{-1}(s_0) = a_4 = s_2$

$b_1^{-1}(s_1) = s_0$

$a_4^{-1}(s_0) = \varepsilon = s_3$



Non-Deterministic Automaton



Non-deterministic automaton recognizing L(e_0)

Linear Derivatives Are Simple

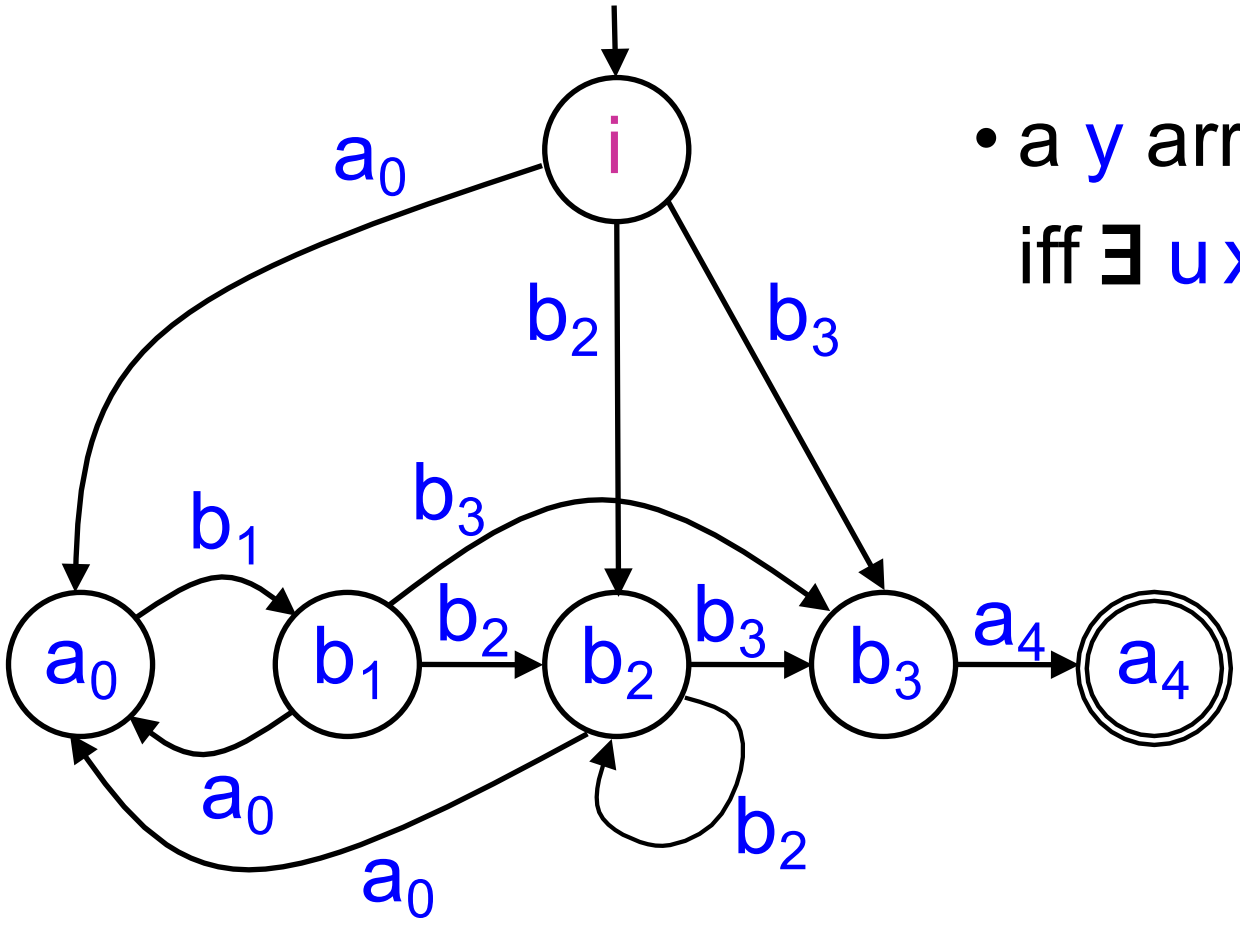
Theorem : if e is linear, then a non-null derivative is fully characterized by its last letter : all derivatives of the form $ux^{-1}(e)$ are null or equal

Proof : by induction on |e| (size of e). Assume $ux^{-1}(e)$ non-null.

1. obvious if $e = x$ (then $|u| = 0$)
2. if $e = e_1 + e_2$, then x appears either in e_1 or in e_2 but not in both
   - if x appears in e_1, then $ux^{-1}(e) = ux^{-1}(e_1)$ with $|e_1| < |e|$
   - if x appears in e_2, then $ux^{-1}(e) = ux^{-1}(e_2)$ with $|e_2| < |e|$
3. if $e = e_1 \cdot e_2$, similar proof by computing $ux^{-1}(e_1 \cdot e_2)$ as a sum of two terms such that only one can contain x
4. if $e = e_1^*$, similar proof to case 3

Linear Expression → 1-local Automaton

$s_0 = (a_0 b_1 + b_2)^* b_3 a_4$

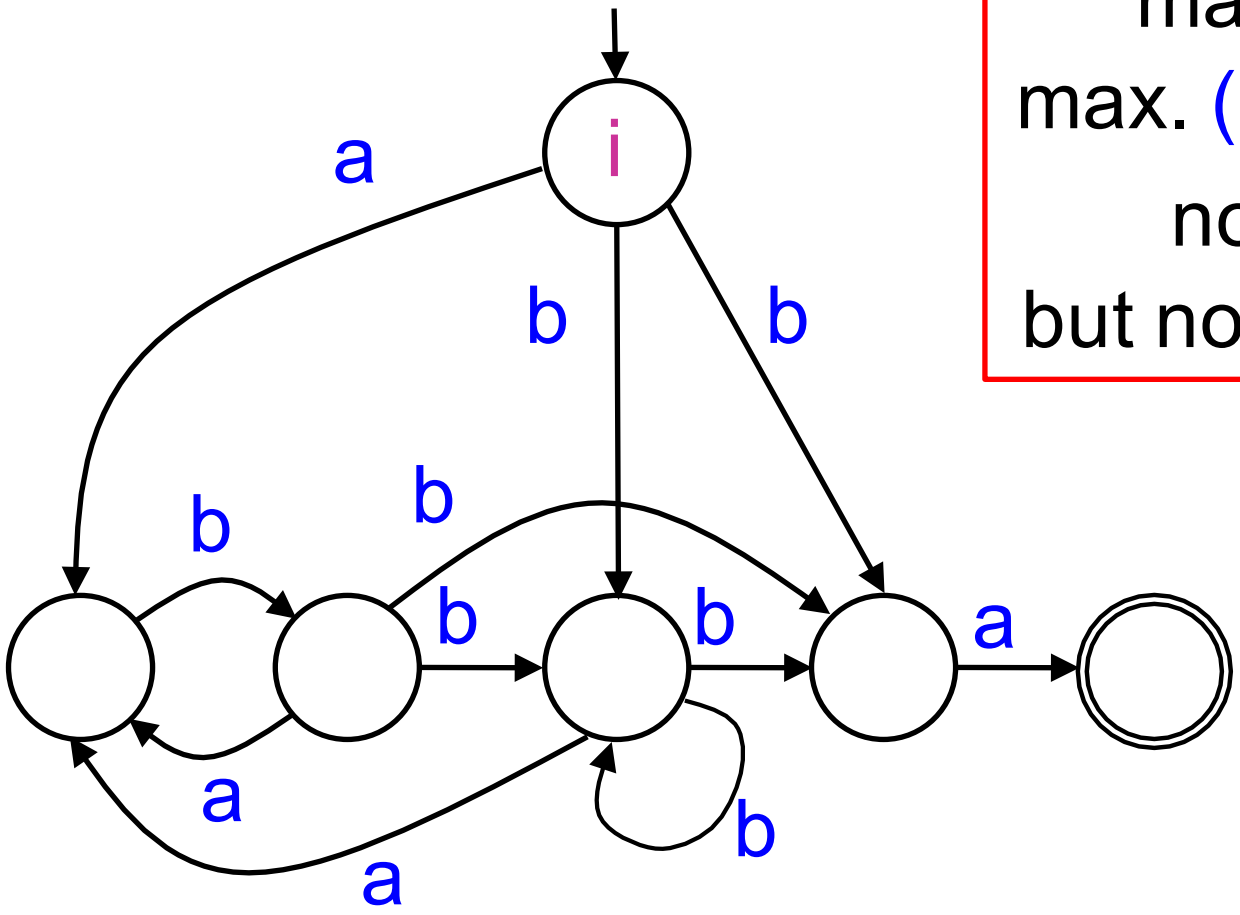


- an arrow from i to x
  if ∃ u = xu ∈ L(s_0)
- a y arrow from x to y
  iff ∃ u x y v ∈ L(s_0)

Very fast
first and follow
computation

Erasing Indices

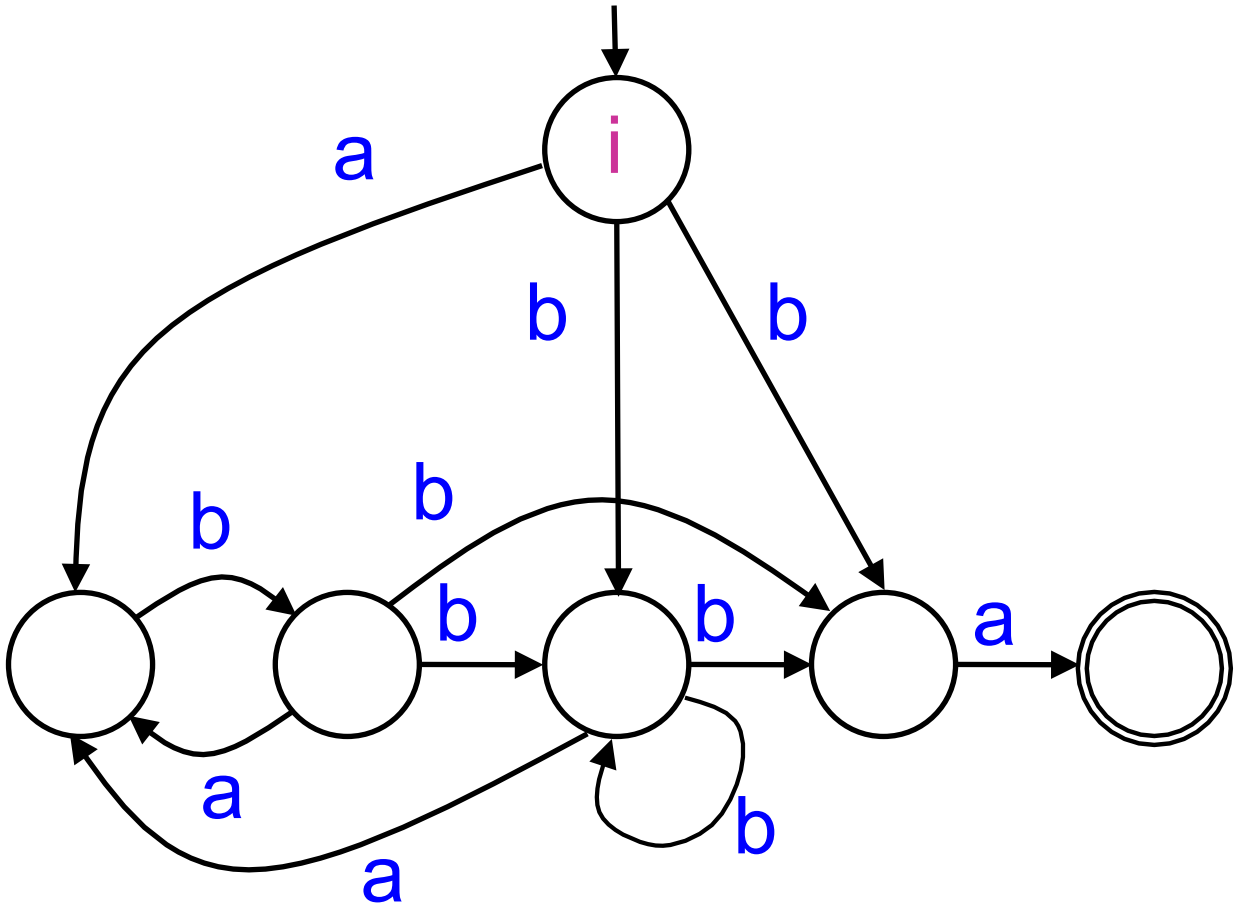
$e = (ab + b)^* ba$



for n letter occurrences
max. n+1 states
max. $(n+1)^2$ transitions
no explosion !
but non-deterministic..

Optimizing Away the Initial State

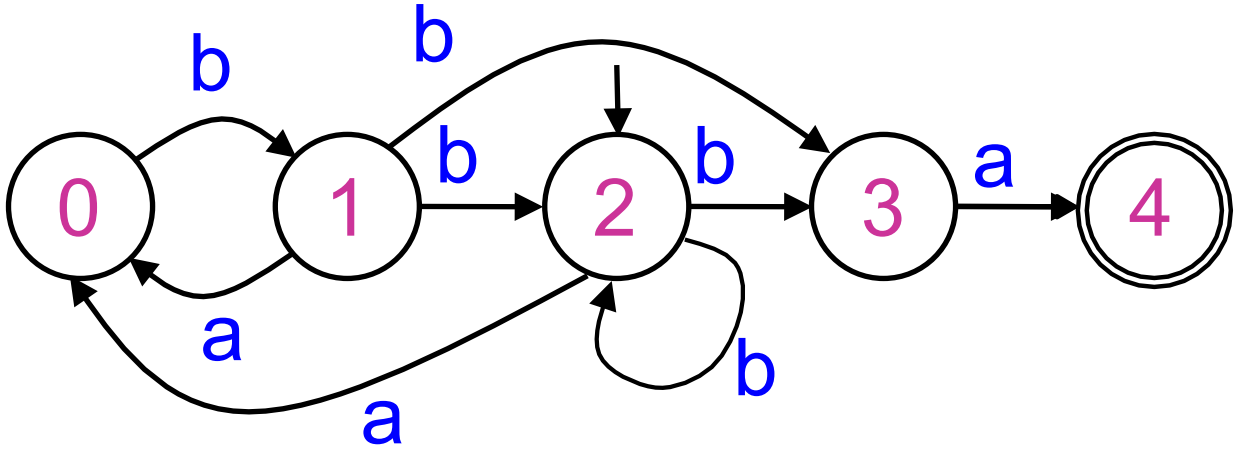
$e = (ab + b)^* ba$



Determinization

e = (ab+b)* ba

00100 : initial

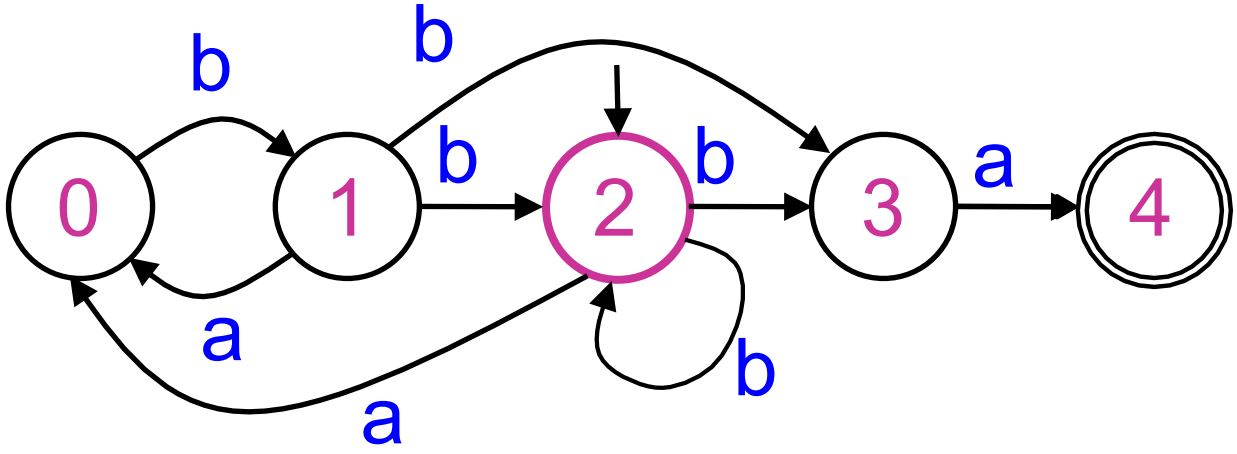


Determinization

$e = (ab + b)^* ba$

00100 : initial

00100 : a →

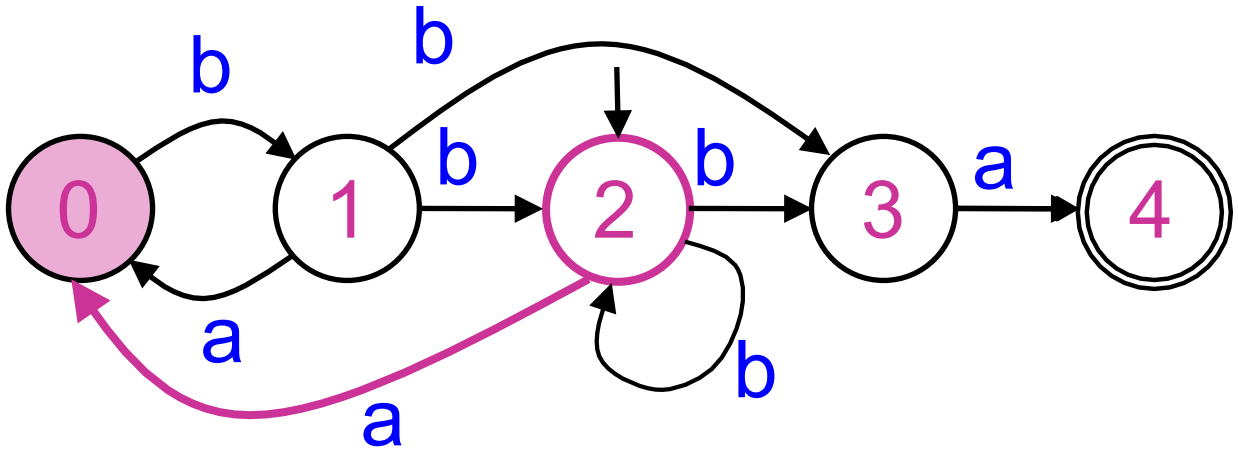


Determinization

$e = (ab + b)^* ba$

00100 : initial

00100 : a → 10000

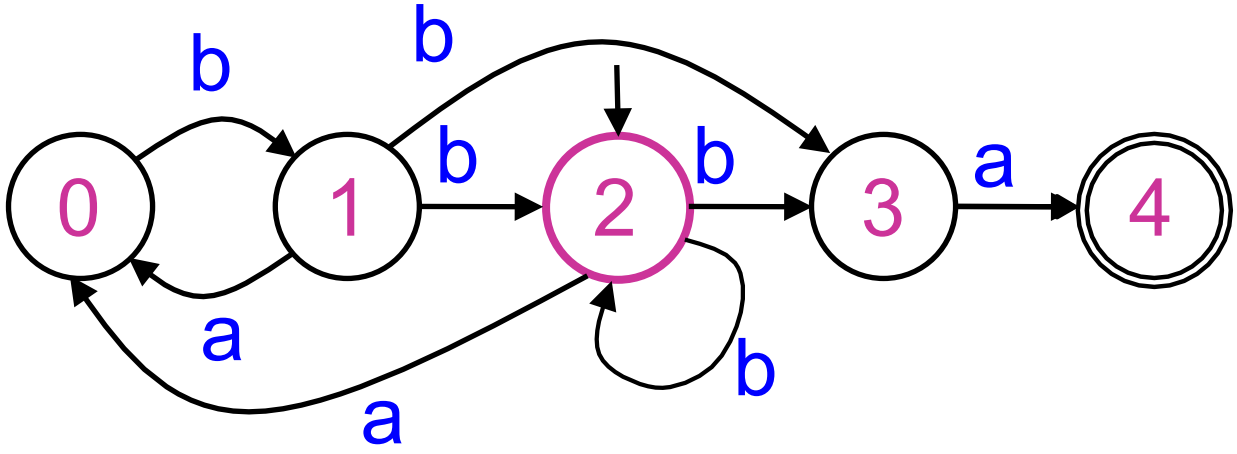


Determinization

$e = (ab + b)^* ba$

00100 : initial

00100 : a → 10000
b →

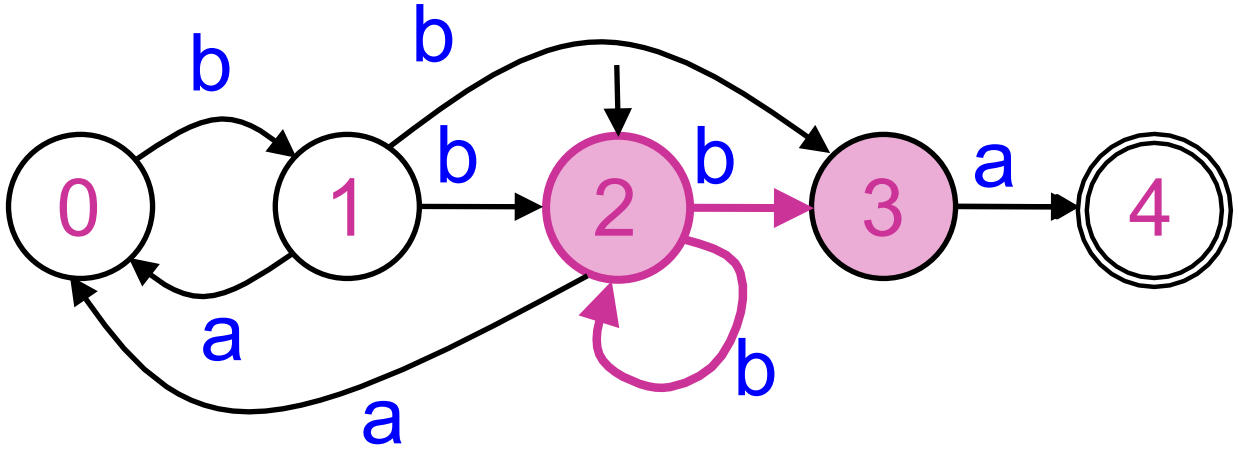


Determinization

$e = (ab + b)^* ba$

00100 : initial

00100 : a → 10000
b → 00110



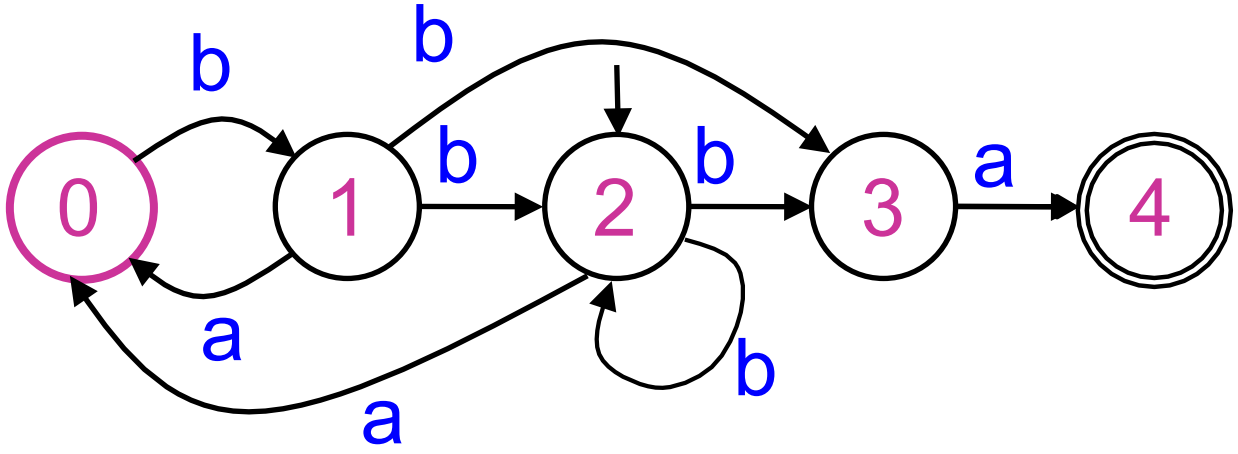
Determinization

$e = (ab + b)^* ba$

00100 : initial

00100 : a → 10000
b → 00110

10000 : b →



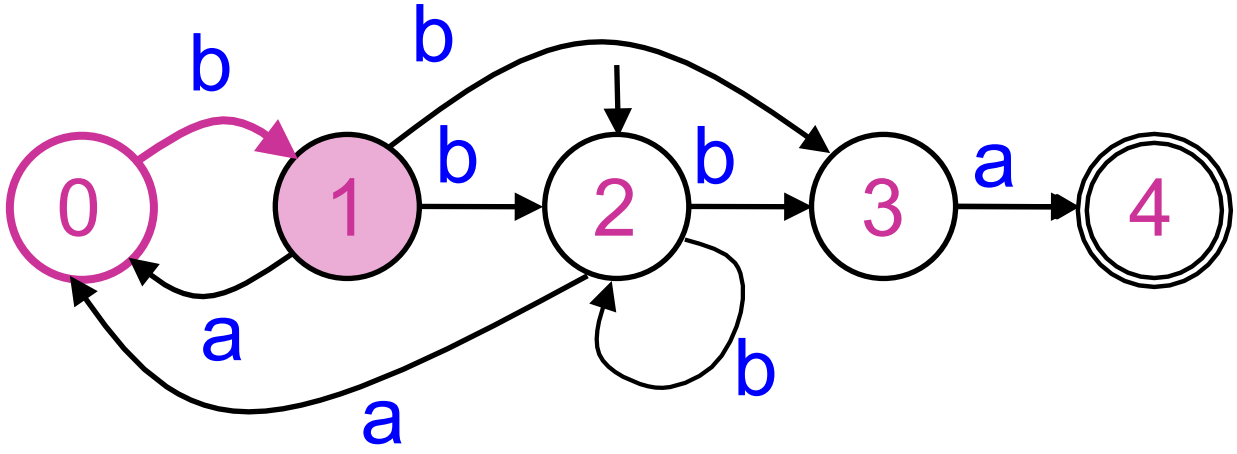
Determinization

e = (ab+b)* ba

00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000



Determinization

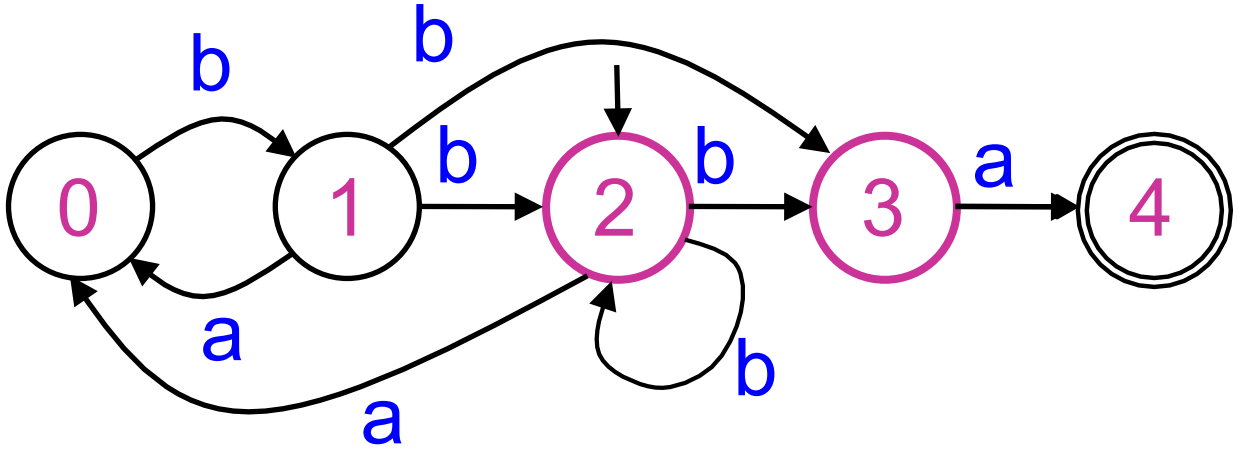
e = (ab+b)* ba

00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a →



Determinization

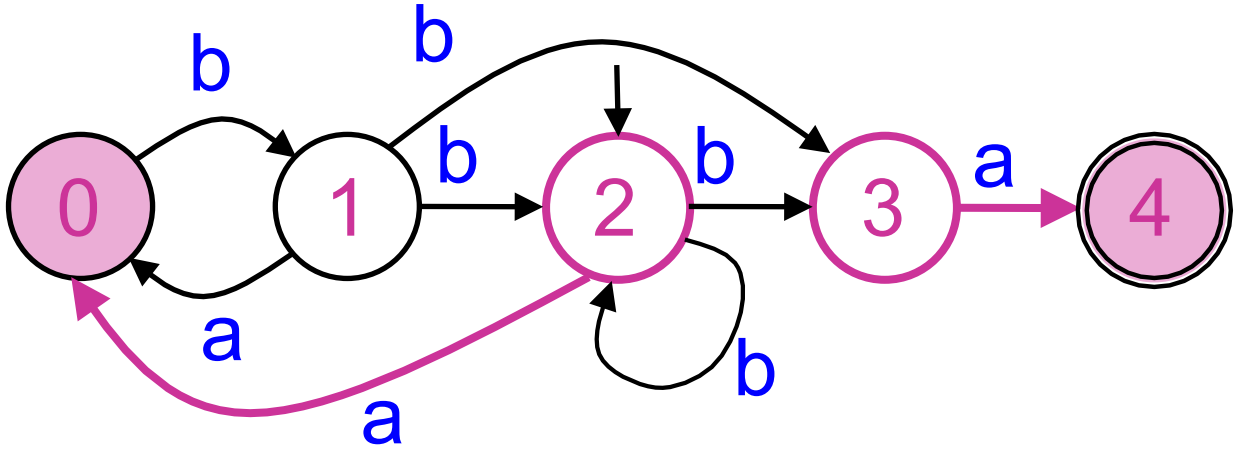
e = (ab+b)* ba

00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → 10001



Determinization

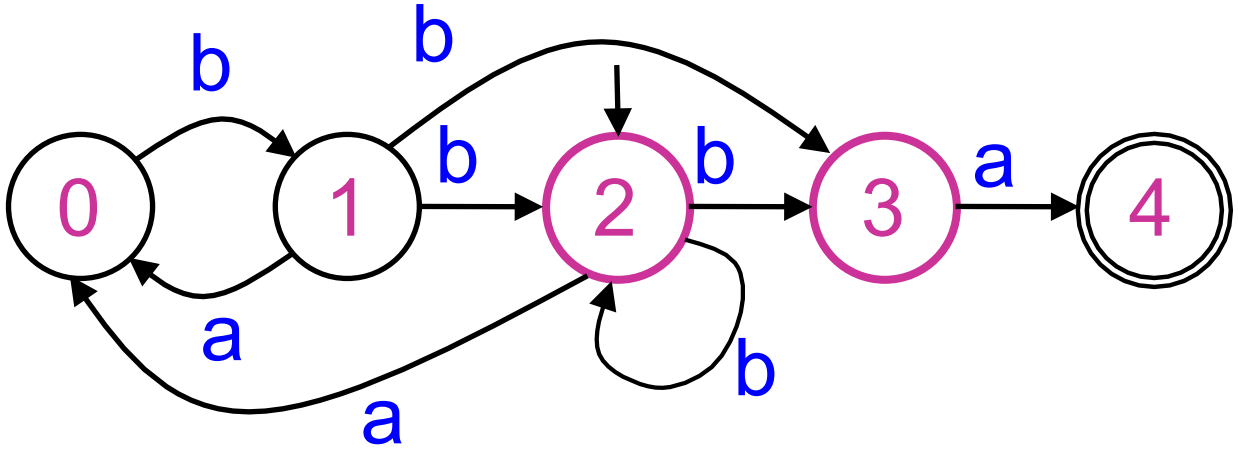
e = (ab+b)* ba

00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → 10001
b →



Determinization

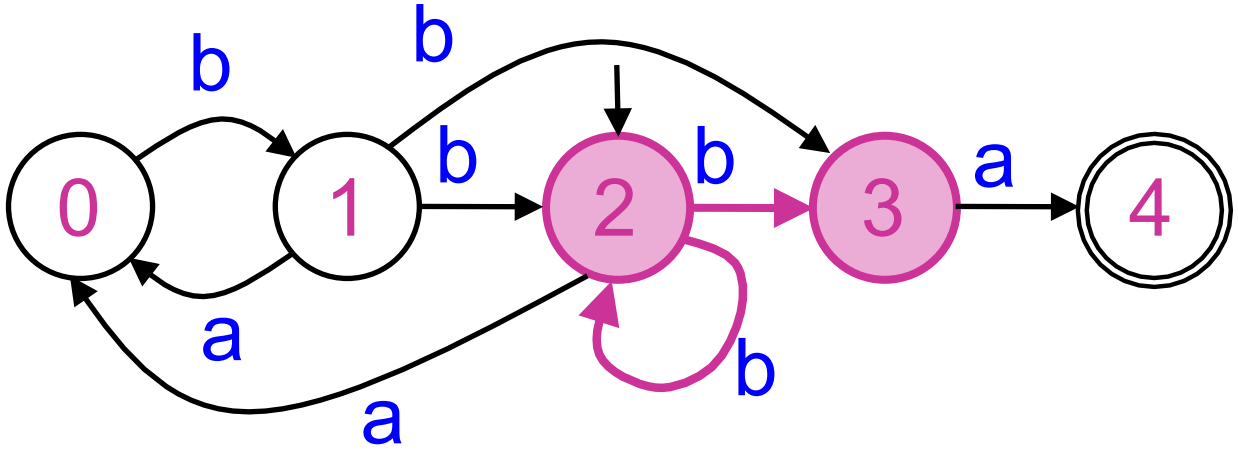
e = (ab+b)* ba

00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → 10001
b → 00110



Determinization

$e = (ab + b)^* ba$

00100 : initial

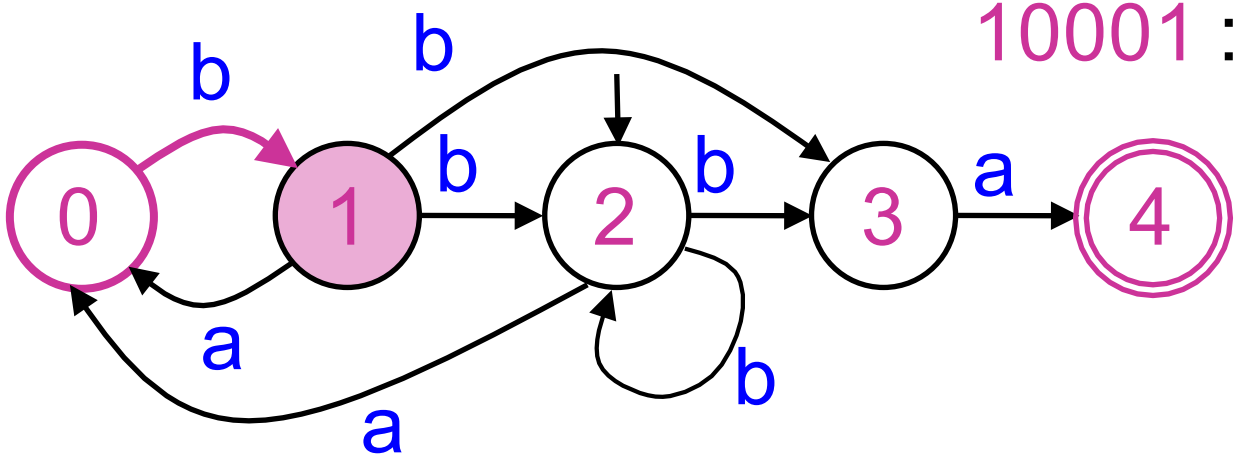
00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → 10001
b → 00110

01000 : a → 10000
b → 00110

10001 : b → 01000



Determinization

$e = (ab + b)^* ba$

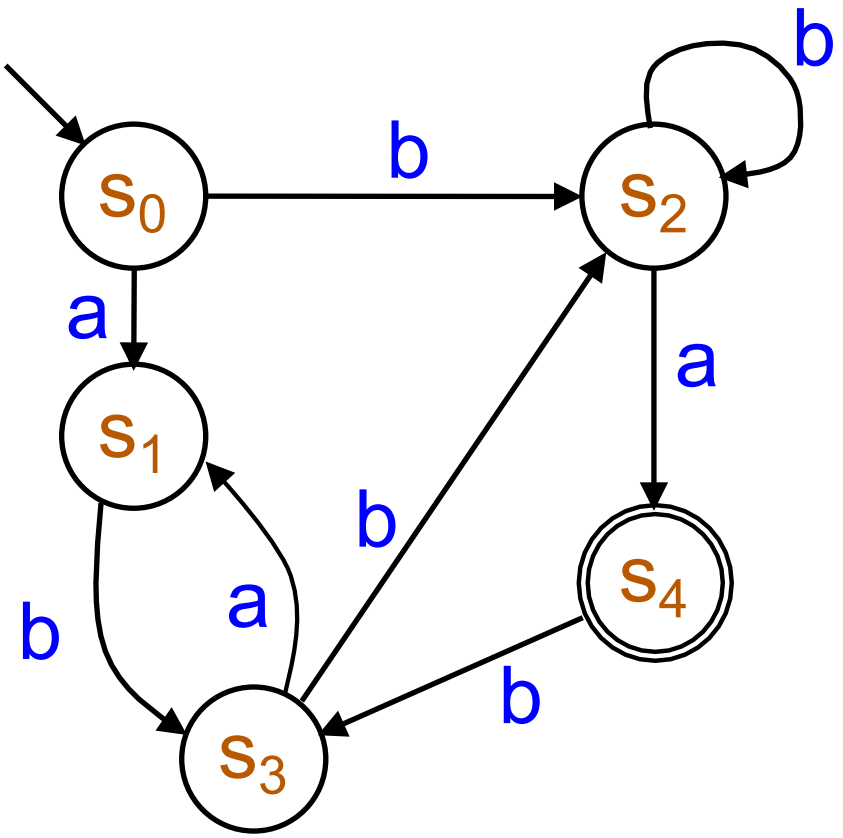
$s_0 = 00100$

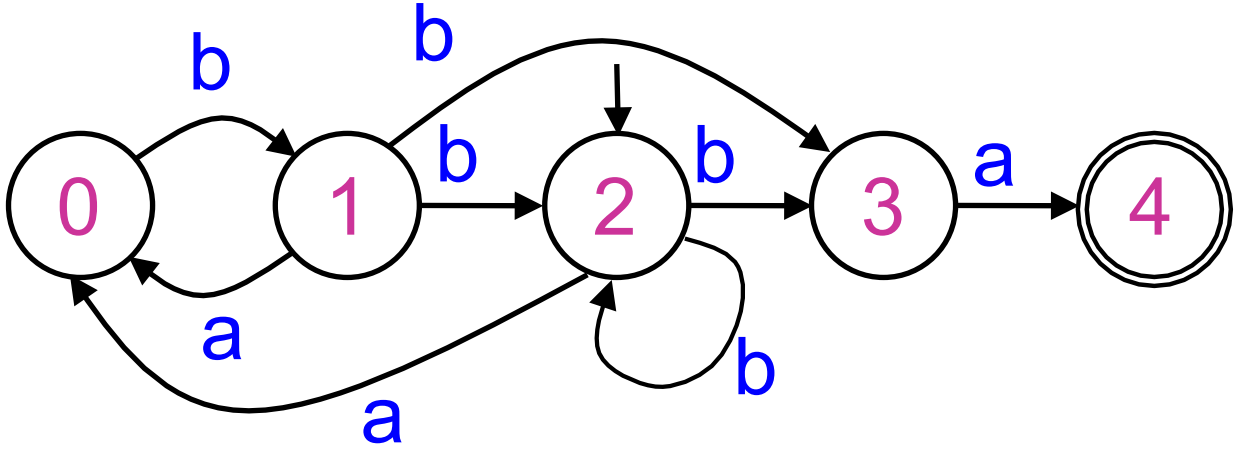
$s_1 = 10000$

$s_2 = 00110$

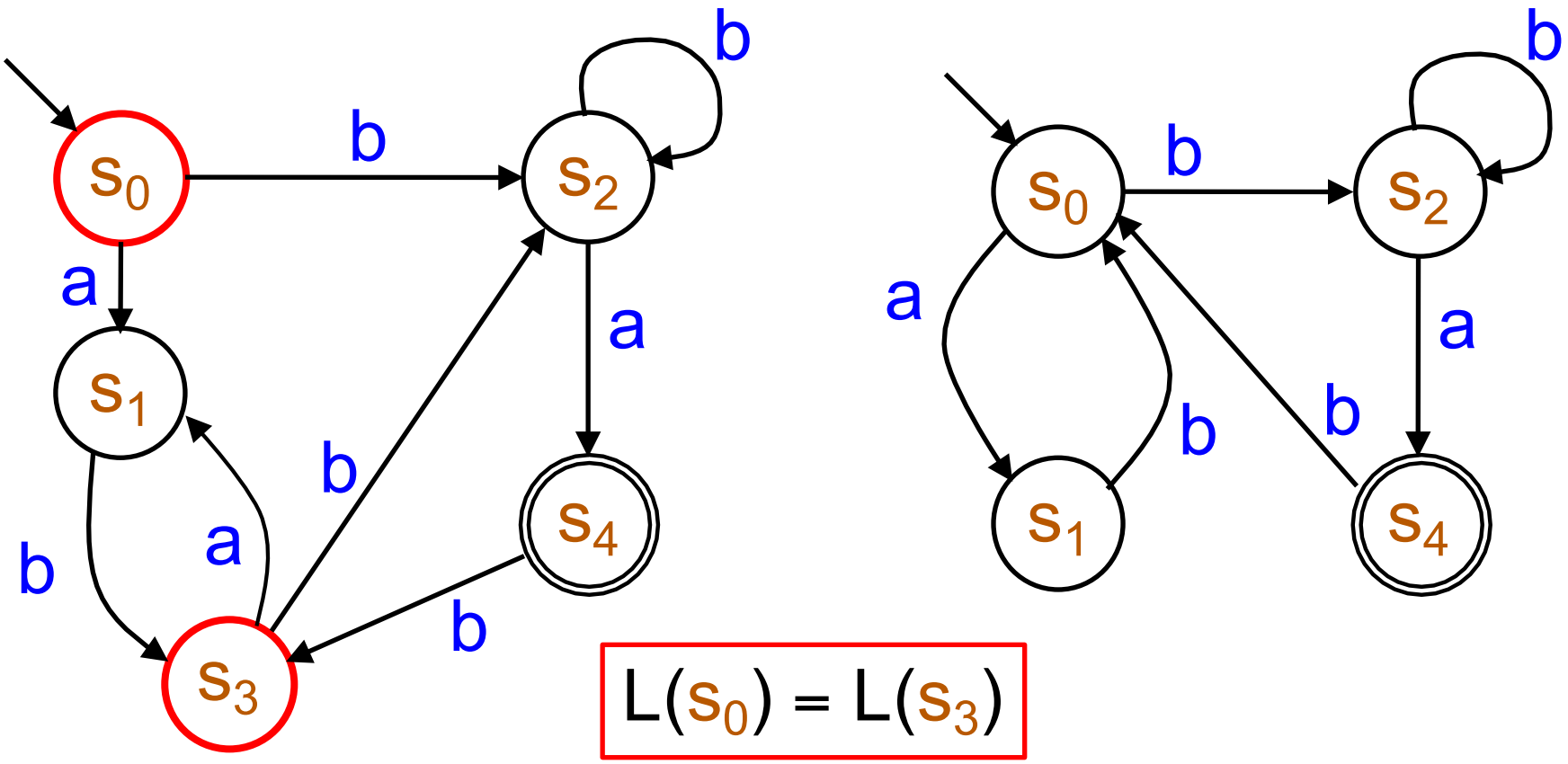
$s_3 = 01000$

$s_4 = 10001$



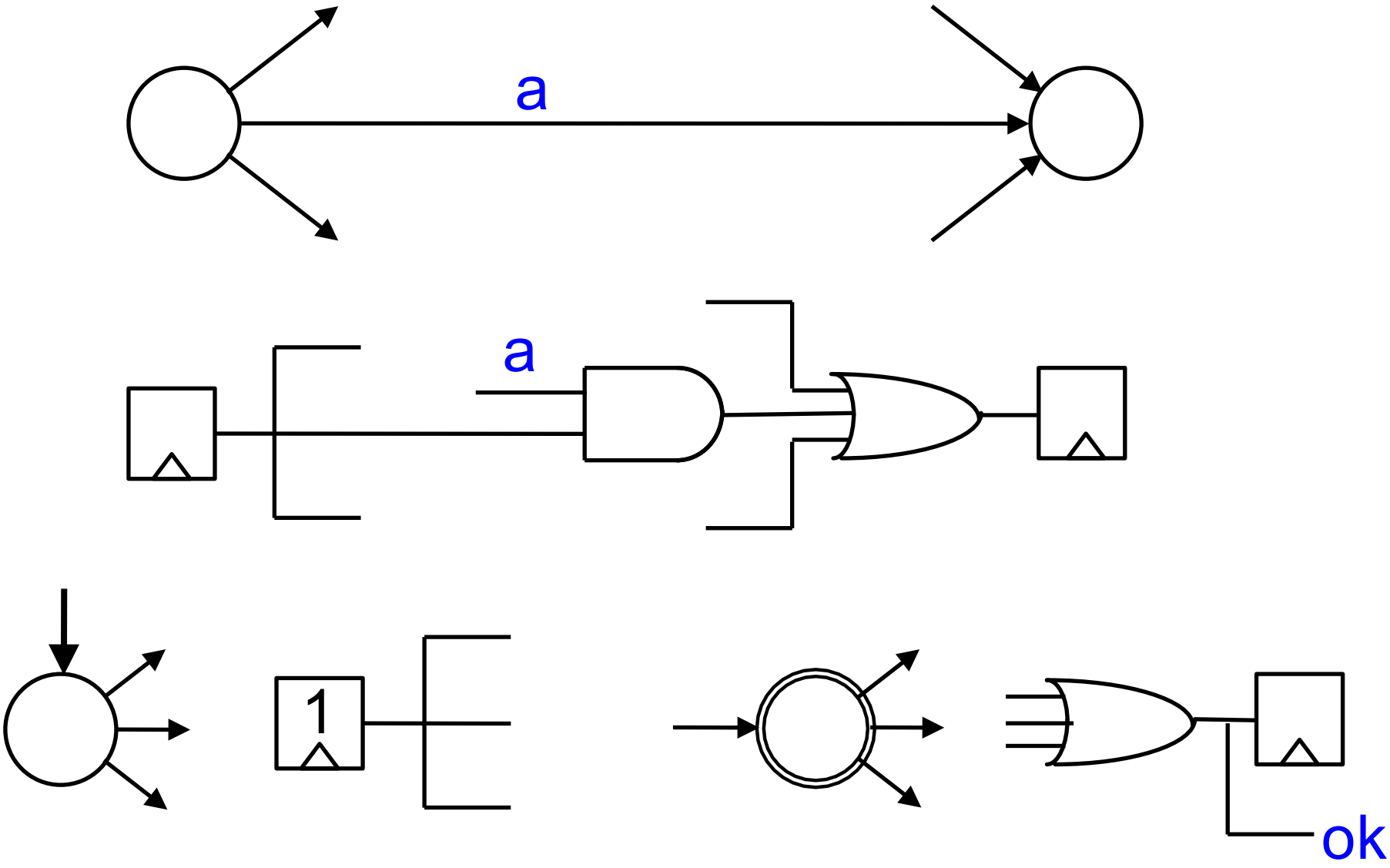


Minimizing Deterministic Automata



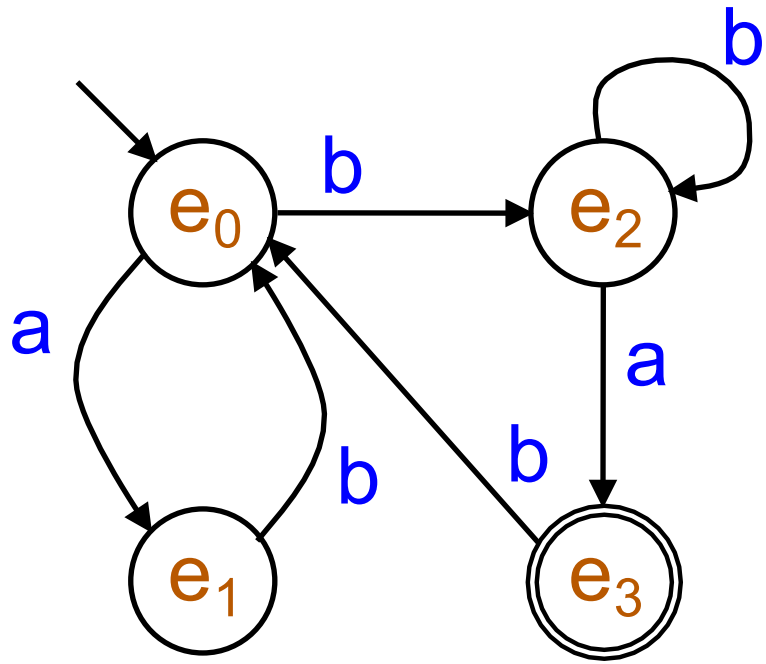
- Theorem : for any regular language L, there exists a minimal deterministic automaton recognizing L.
  Idea : states= nonempty $u^{-1}(L)$
- Bonus : there exists a fast minimization algorithm

Implementation by Boolean Circuits

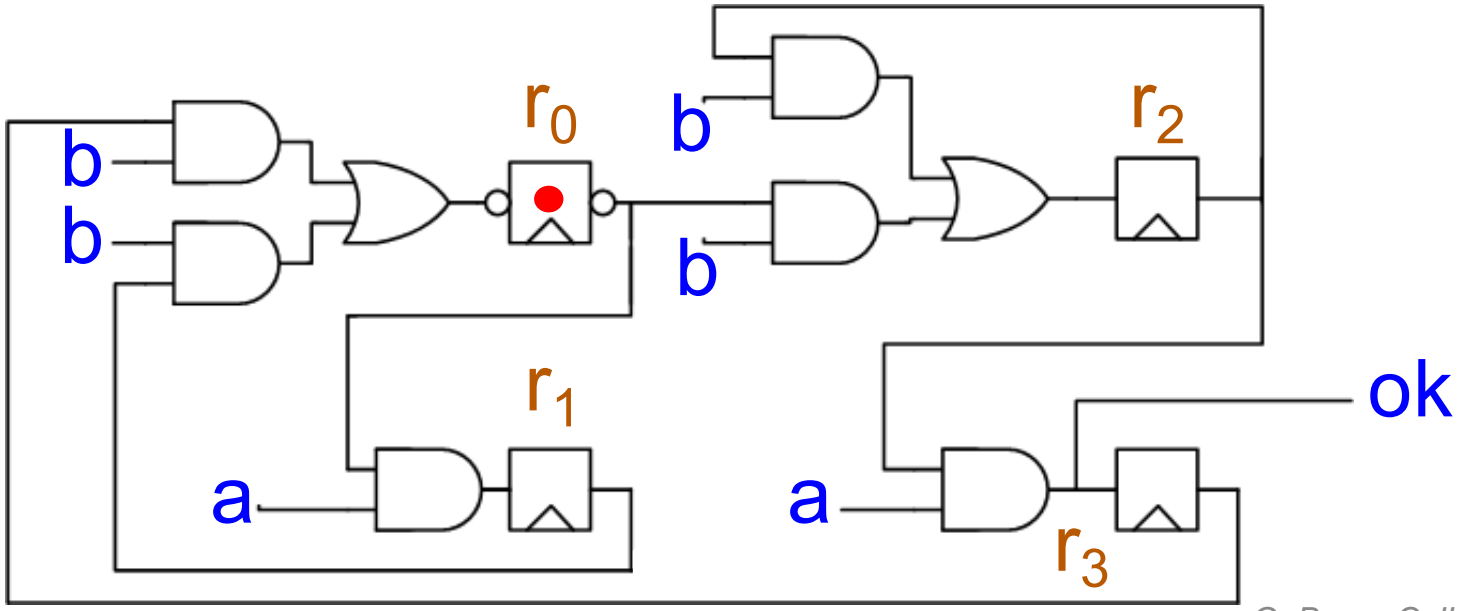


The Deterministic Case : 1-hot encoding

(ab+b)*ba

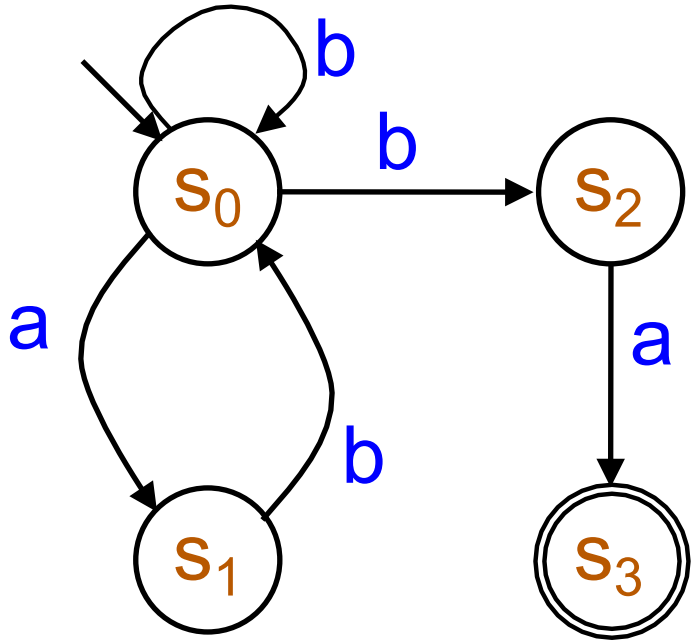


1-hot encoding
(only one r_i to 1)
size explosion!
fanout explosion !



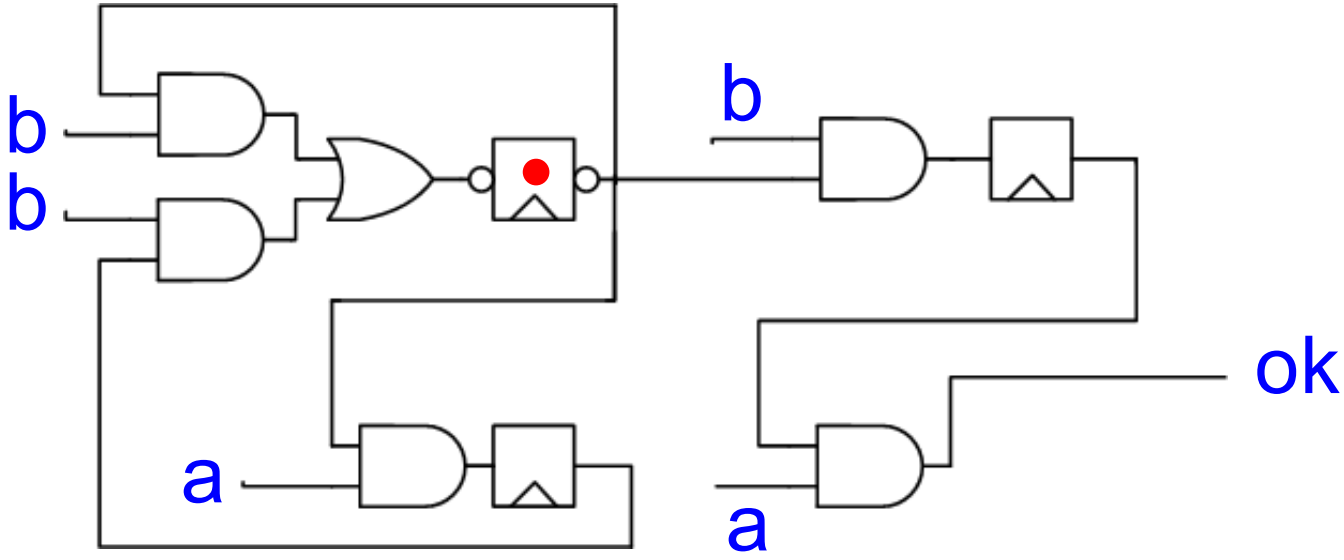
The Non-Deterministic Case

(ab+b)*ba



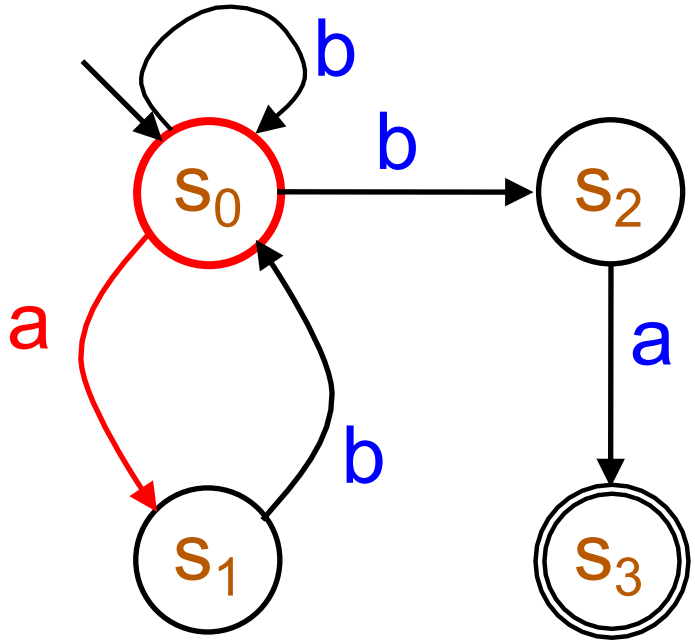
no size explosion

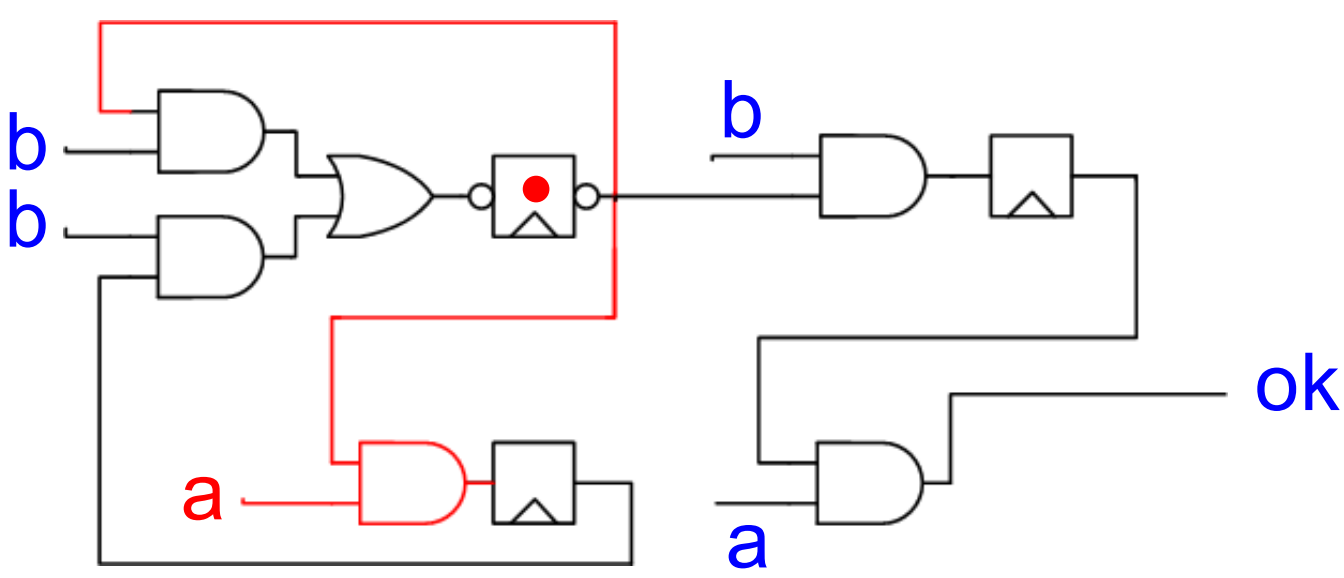
⇒ much better!



Light Speed Subset Construction

(ab+b)*ba

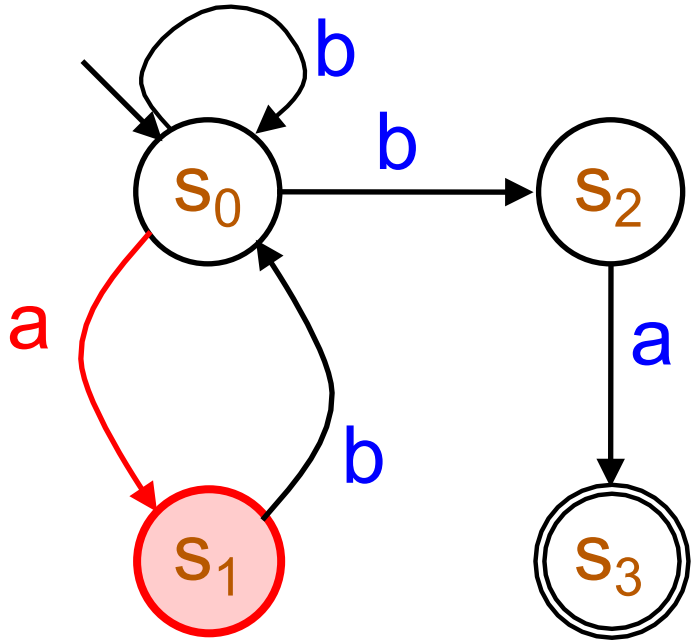




a

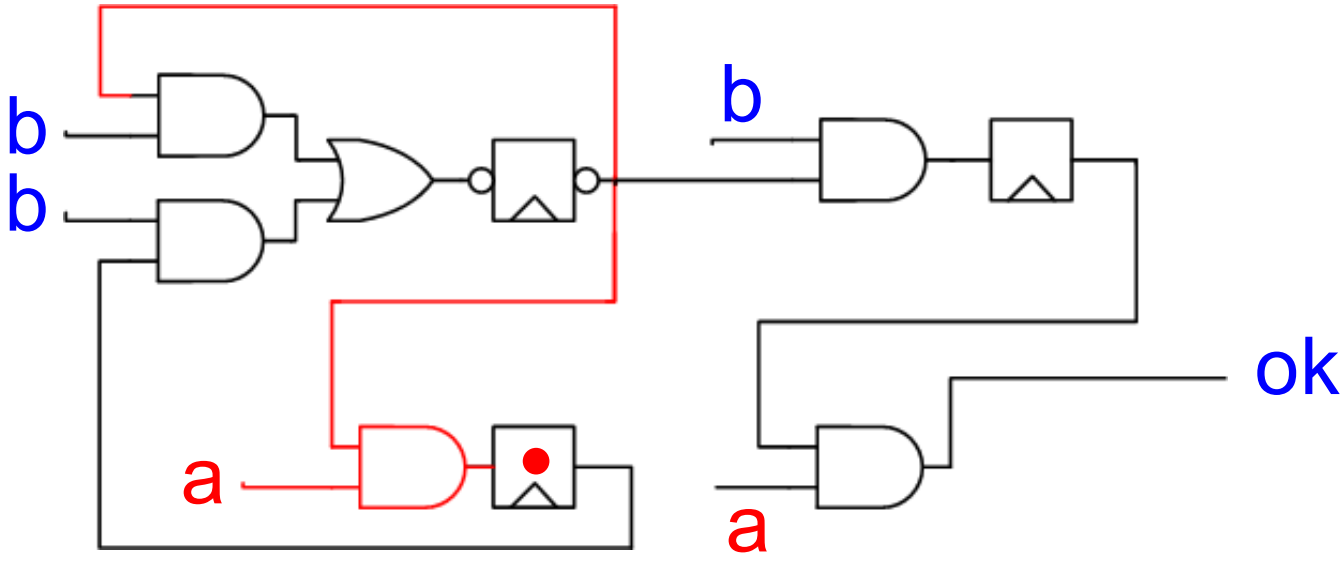
Light Speed Subset Construction

(ab+b)*ba



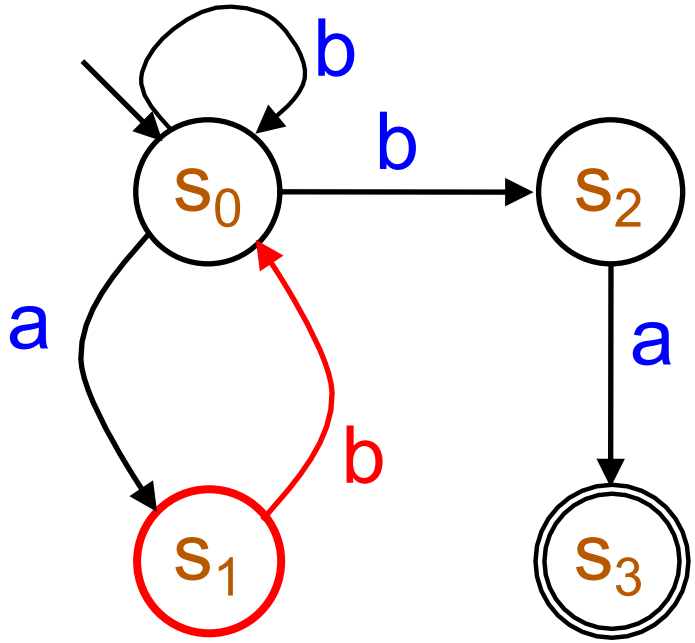
tick!

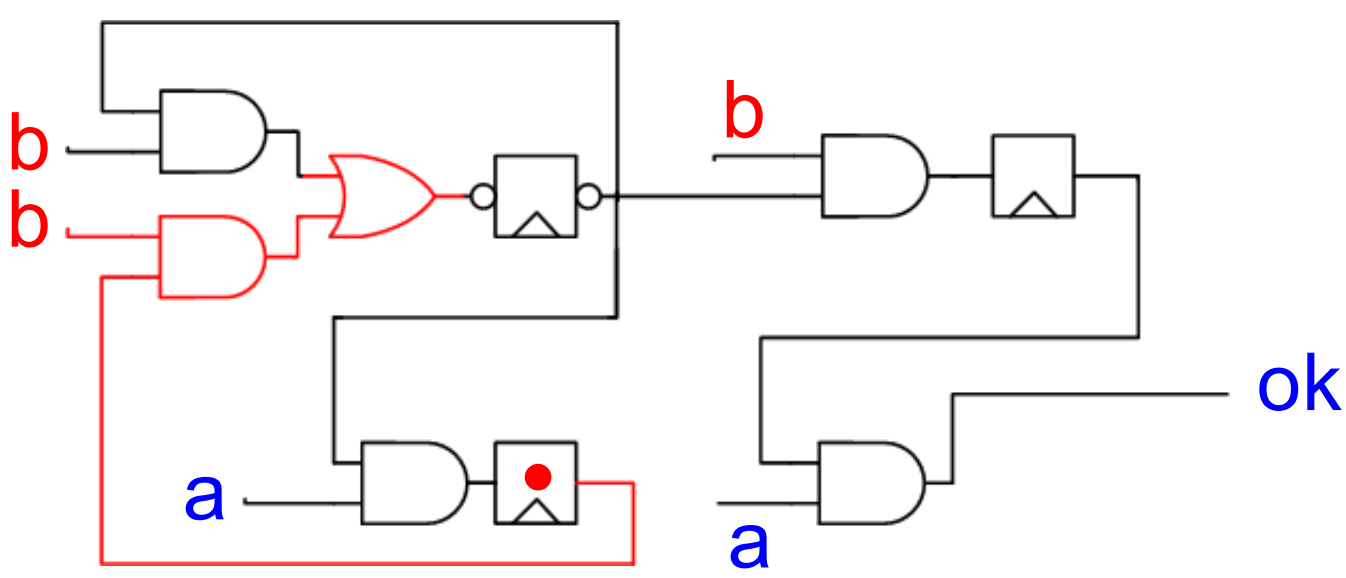
a



Light Speed Subset Construction

(ab+b)*ba

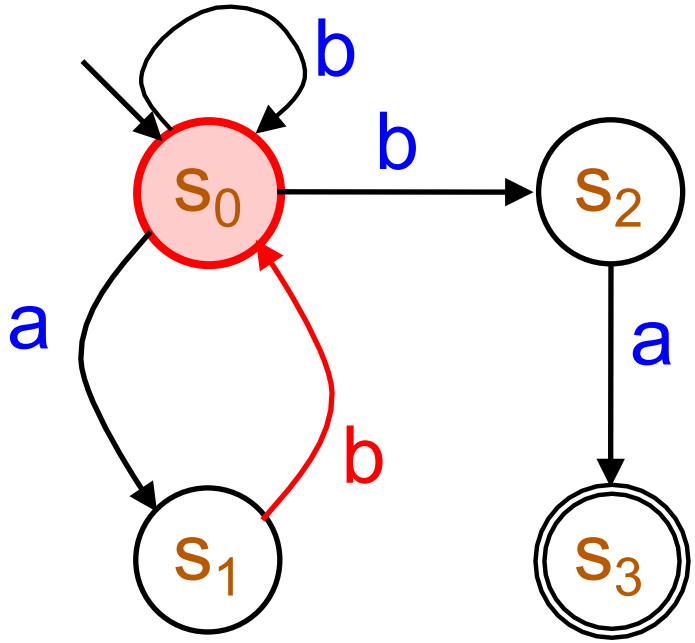




ab

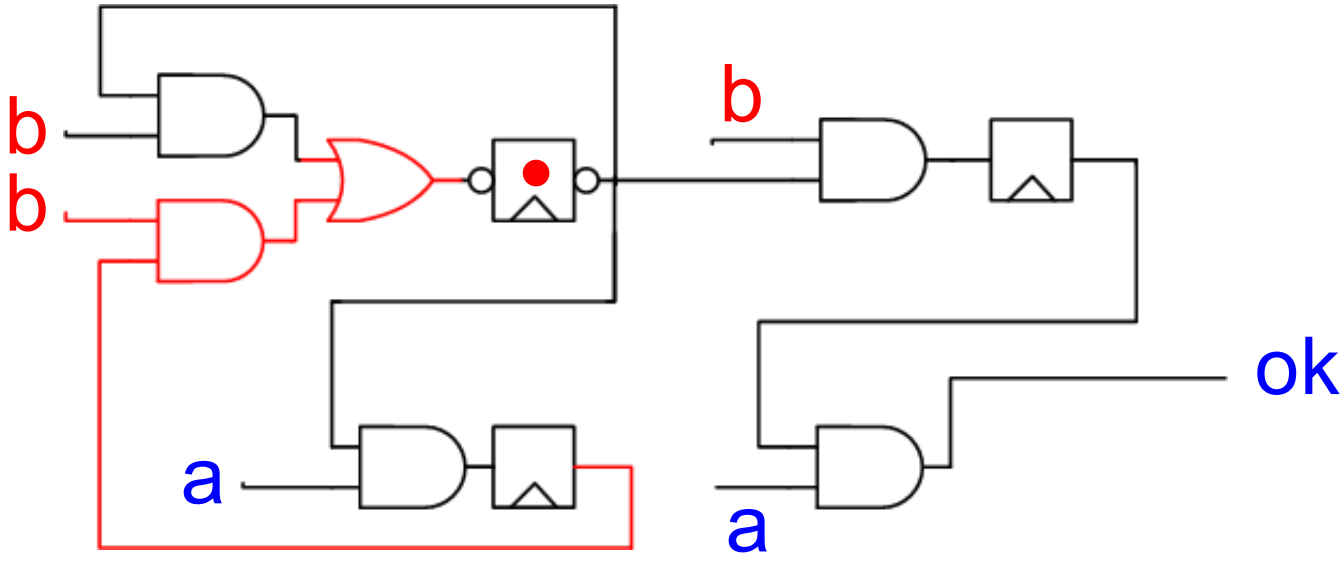
Light Speed Subset Construction

(ab+b)*ba



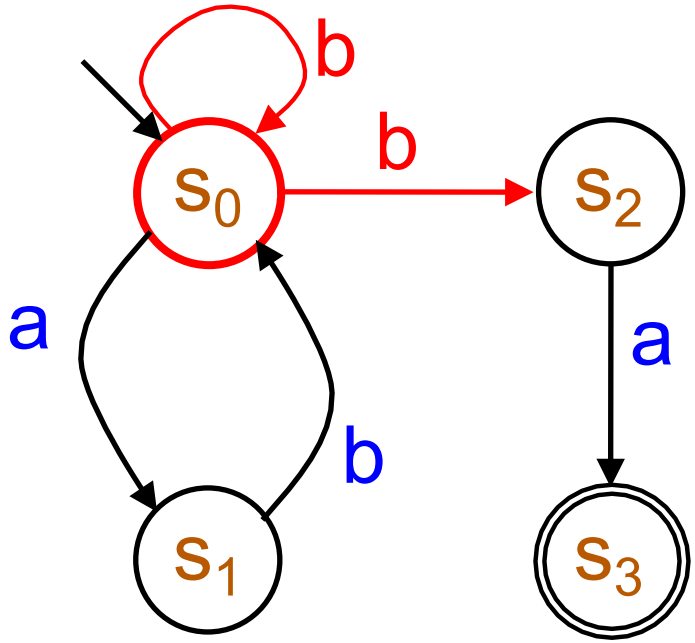
tick!

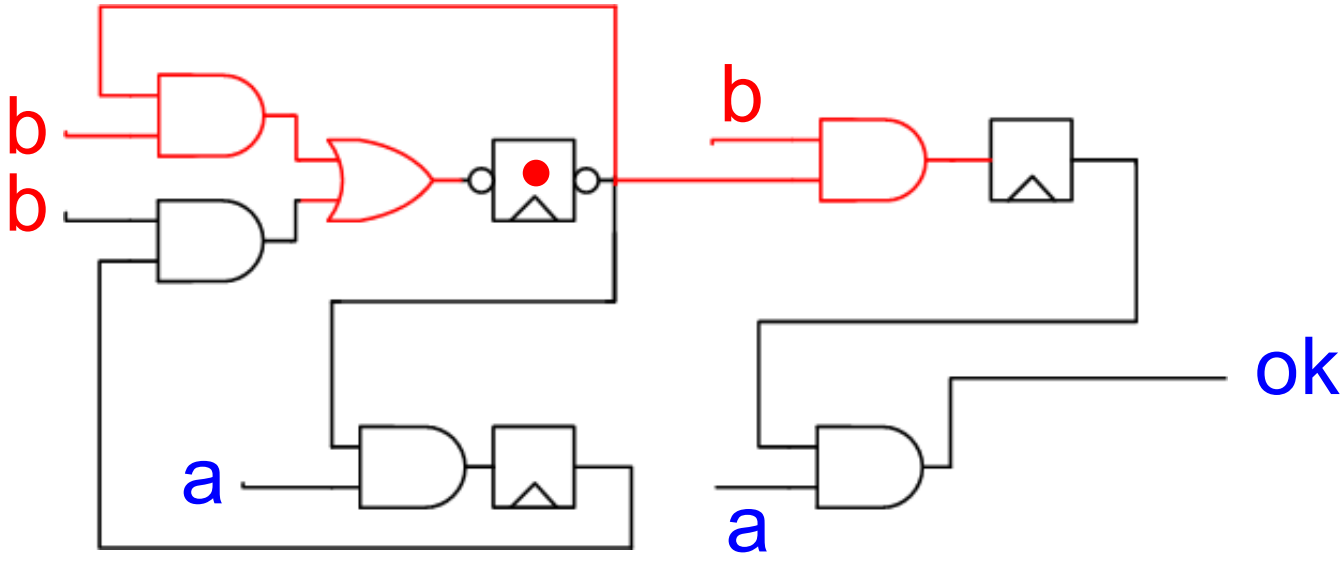
ab



Light Speed Subset Construction

(ab+b)*ba

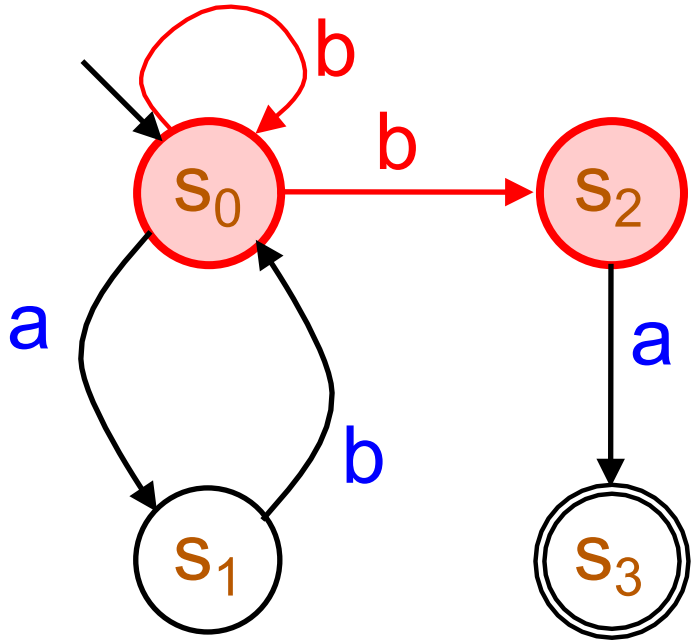




abb

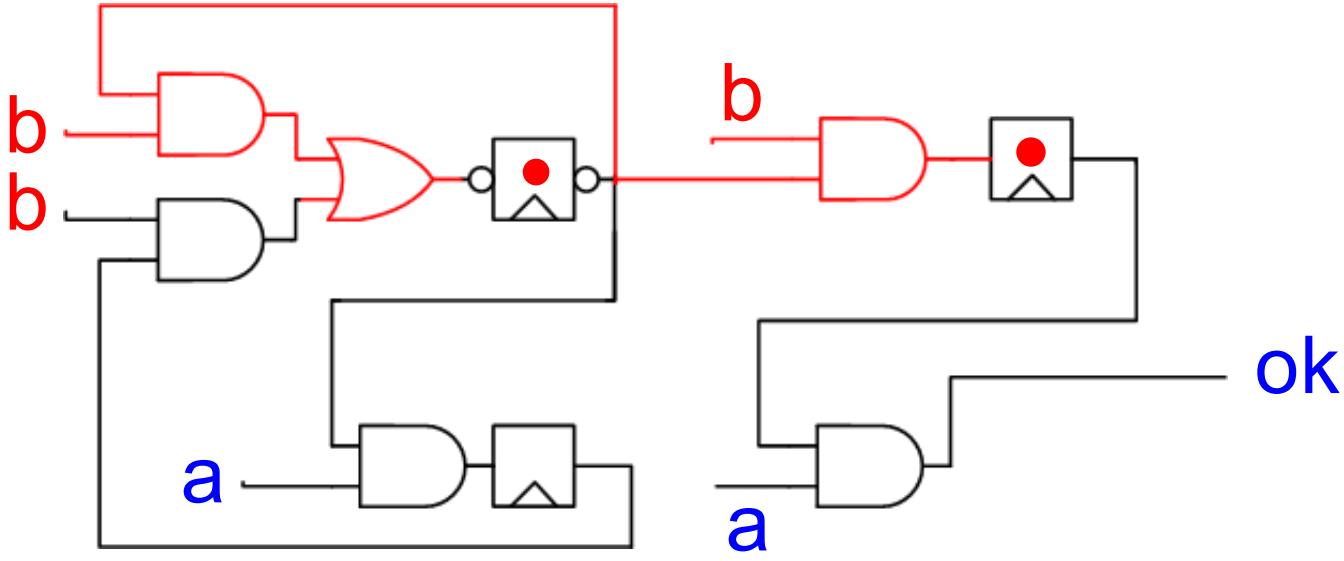
Light Speed Subset Construction

(ab+b)*ba



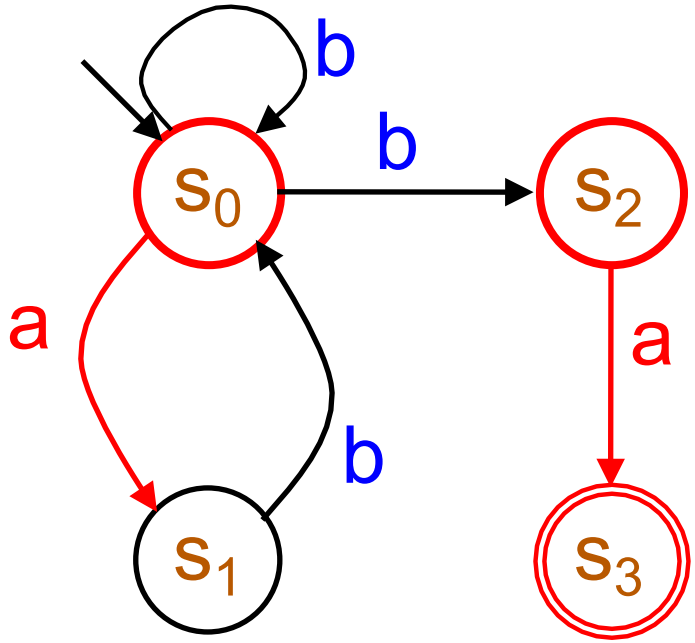
tick!

abb

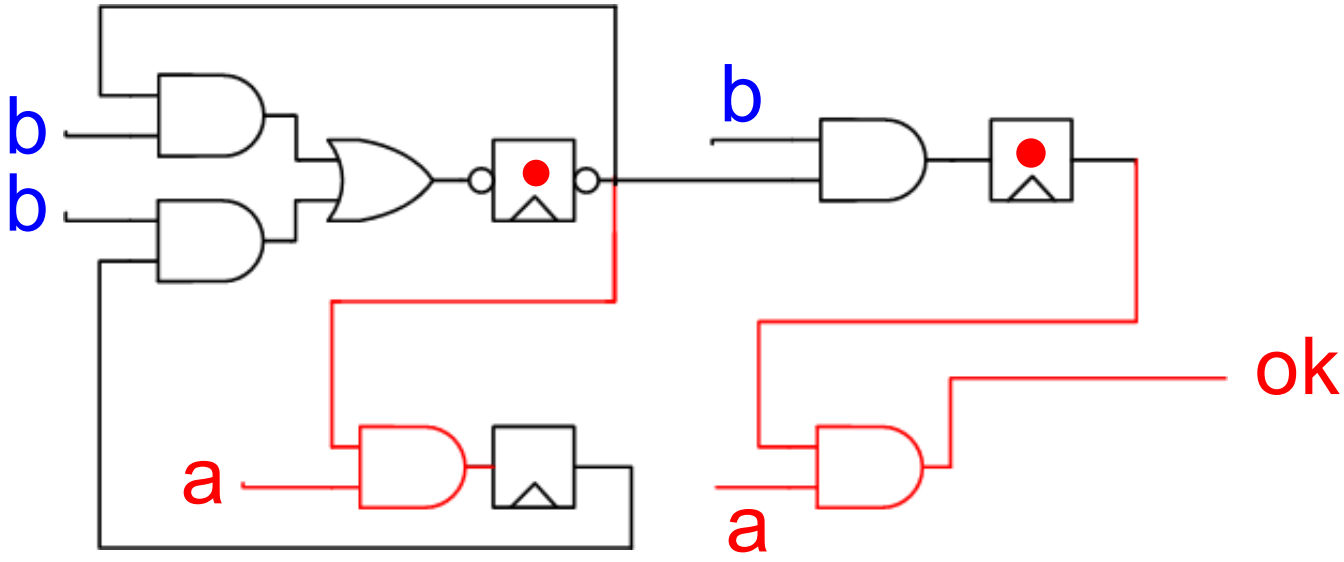


Light Speed Subset Construction

(ab+b)*ba

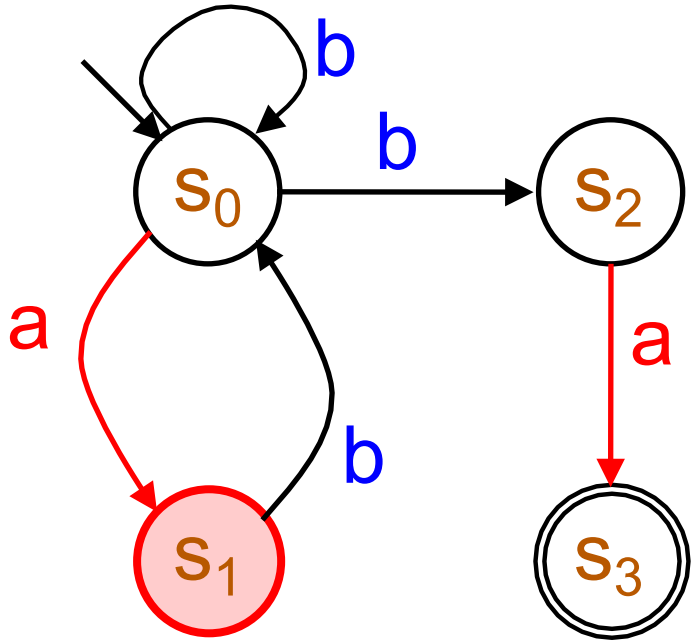


abba



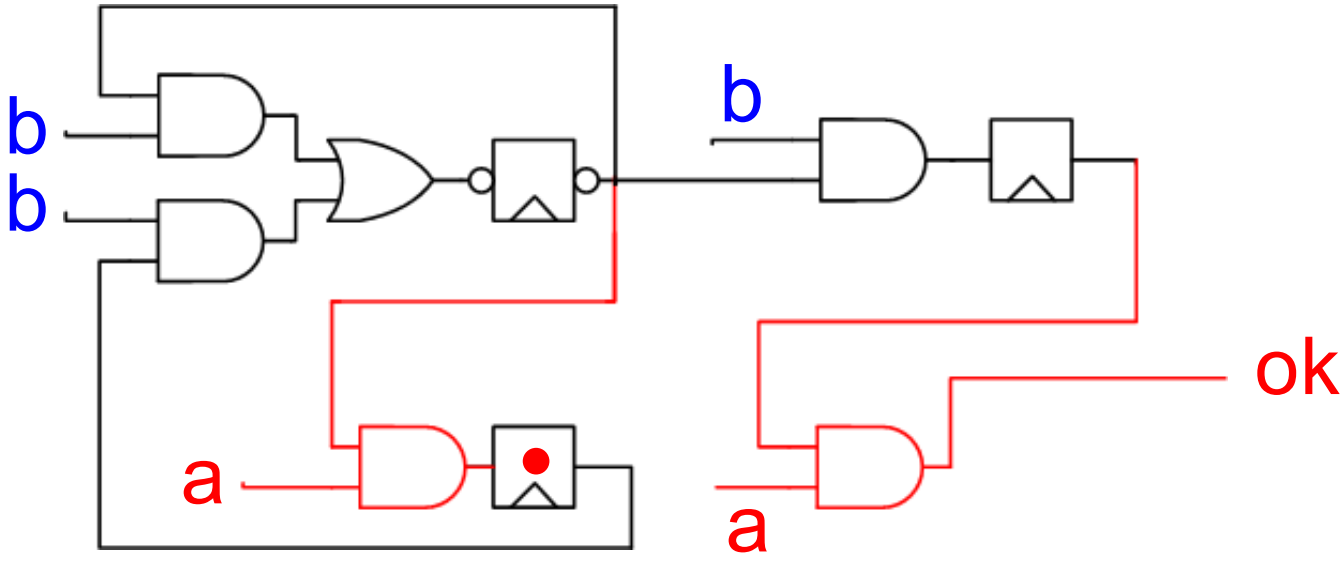
Light Speed Subset Construction

(ab+b)*ba



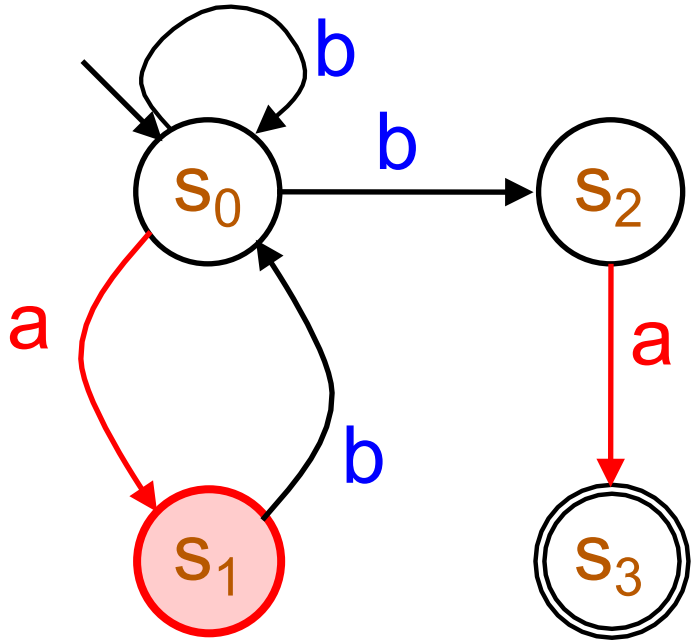
tick!

abba



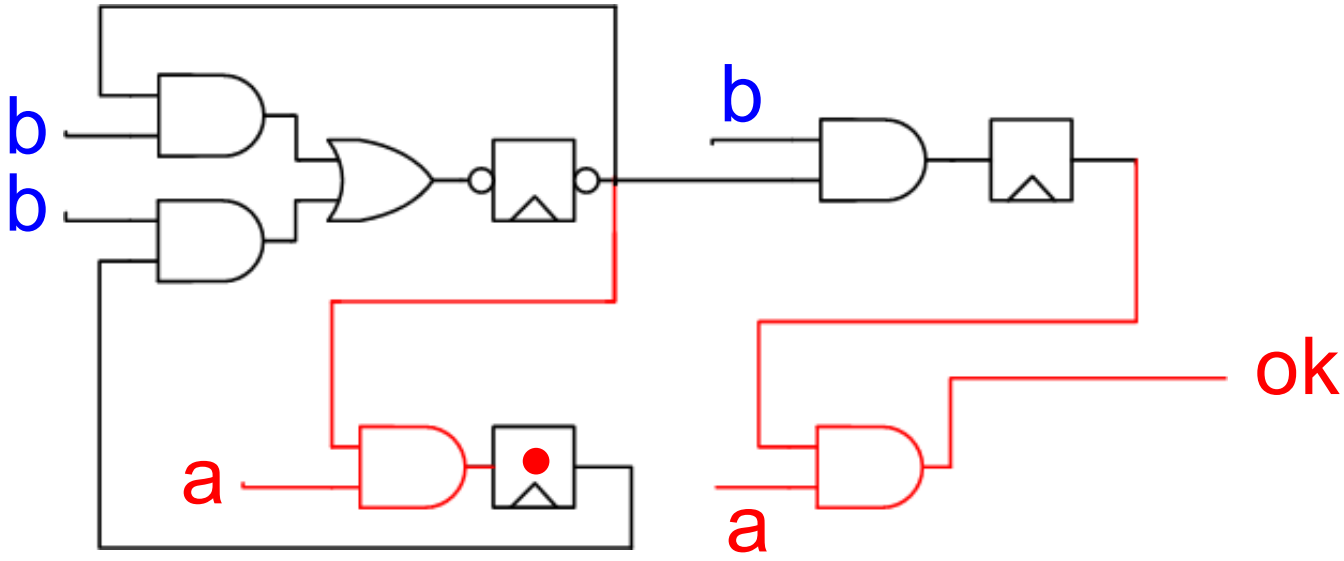
Light Speed Subset Construction

(ab+b)*ba



tick!

abba



Fundamental Practical Result

Any regular expression of size n is recognized
by a circuit with n+1 registers and at most n^2 gates

- Deterministic, superfast, scales up in size
- Almost always better than determinization
- Can be cleverly optimized and formally verified (using BDDs, SAT, SMT)

Can do better for regular expression (linear),
See Reglo by Pascal Raymond

Alternative: Dense Encoding ?

When coding the DFA in HW, why one register per state? Number states in binary → log(n) regs !

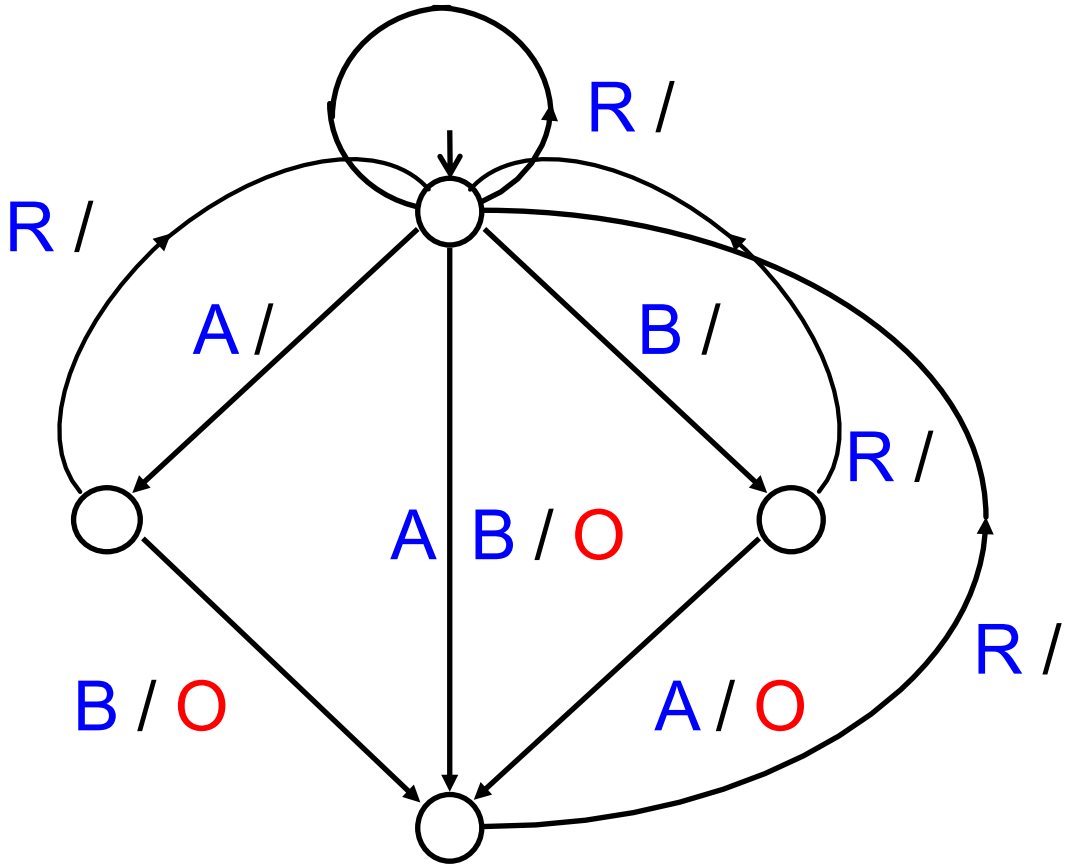
Not Quite ! State transition logic can be exponential in the number of registers.

Furthermore, n! numberings to try, and no good heuristics for that!

Expensive commercial systems cannot handle really useful DFAs with 12 states

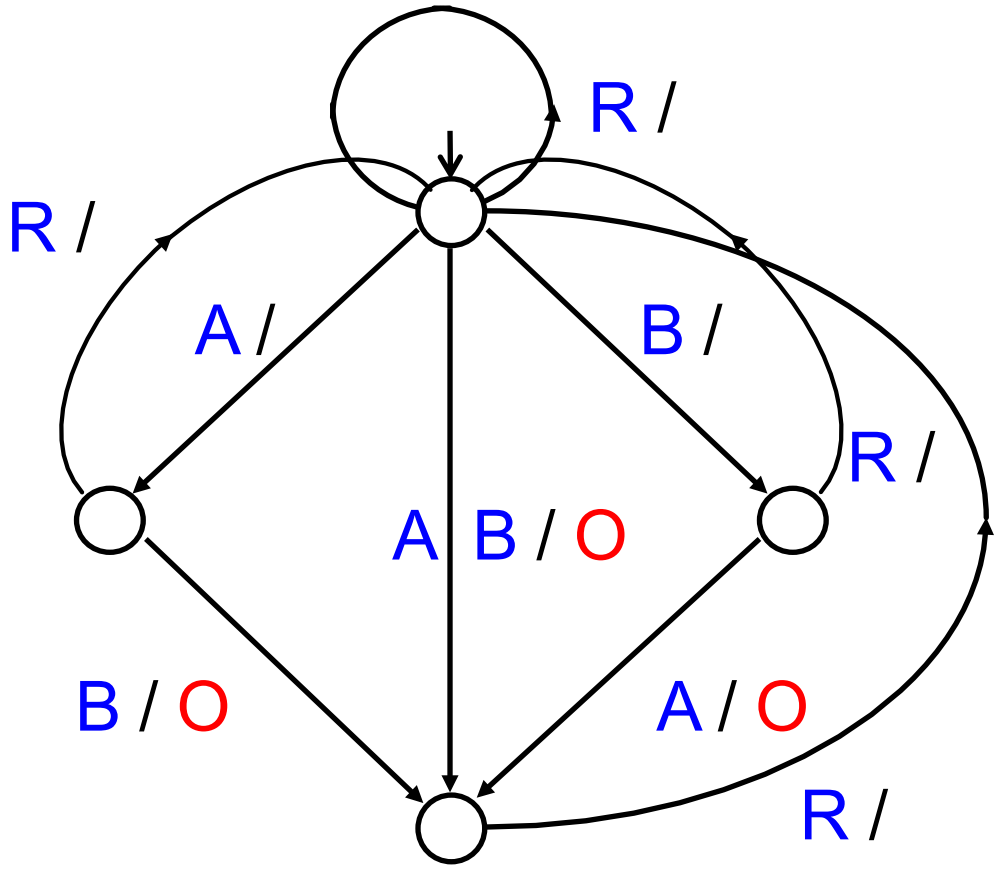
Saving One More Exponential : ABRO

Emit O as soon as A and B have arrived
Reset behavior each time R is received



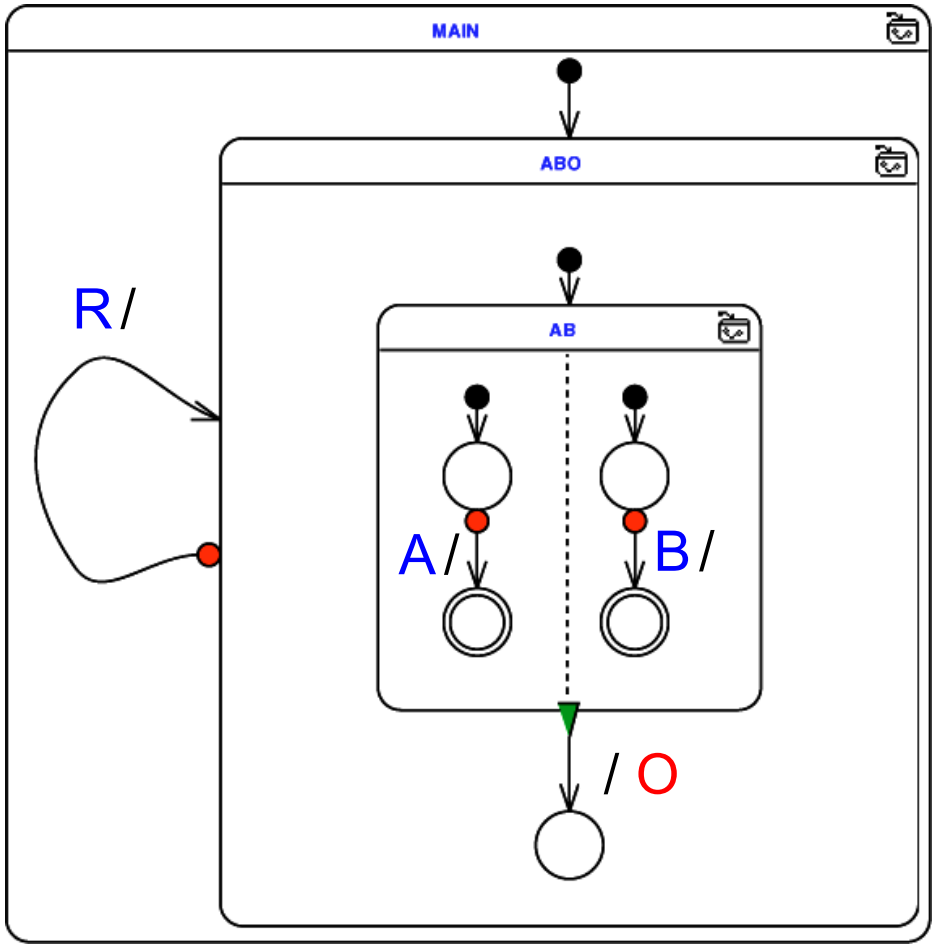
Memory write
R : Request
A : Address
B : Data
O : Write

Esterel = Linear Specification




```
loop
   abort
        { await A || await B };
        emit O ;
        halt
   when R
end loop
```

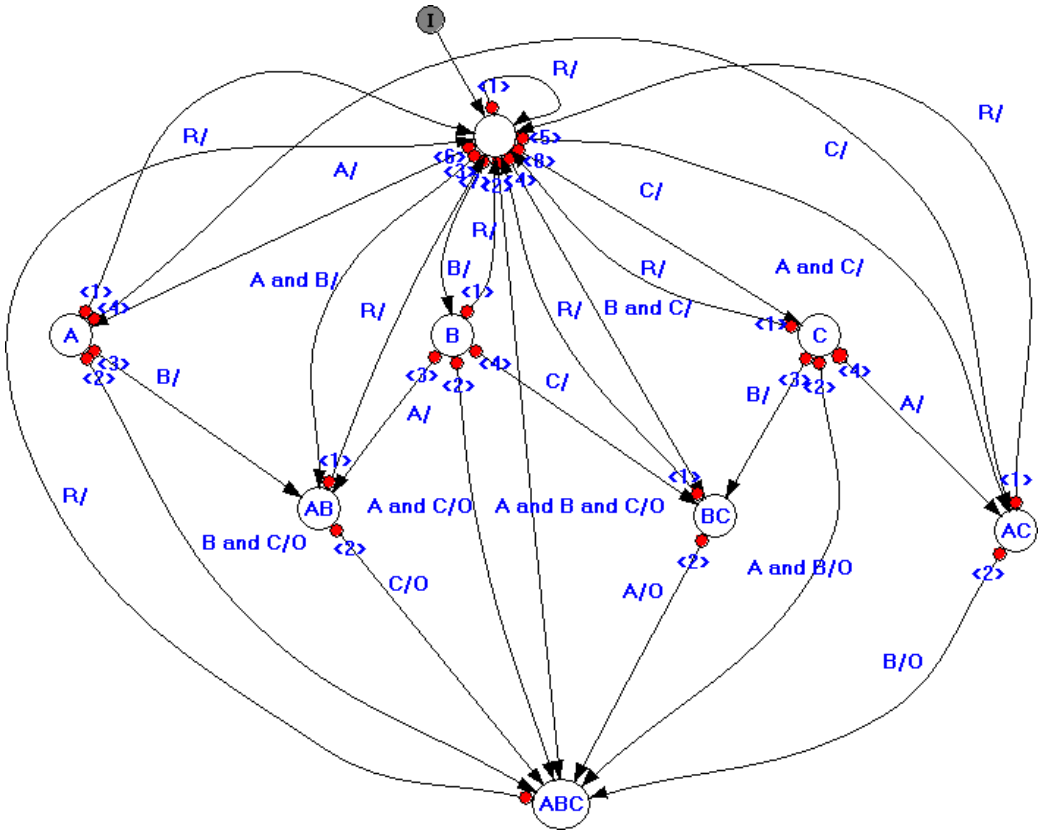
SyncCharts (C. André)



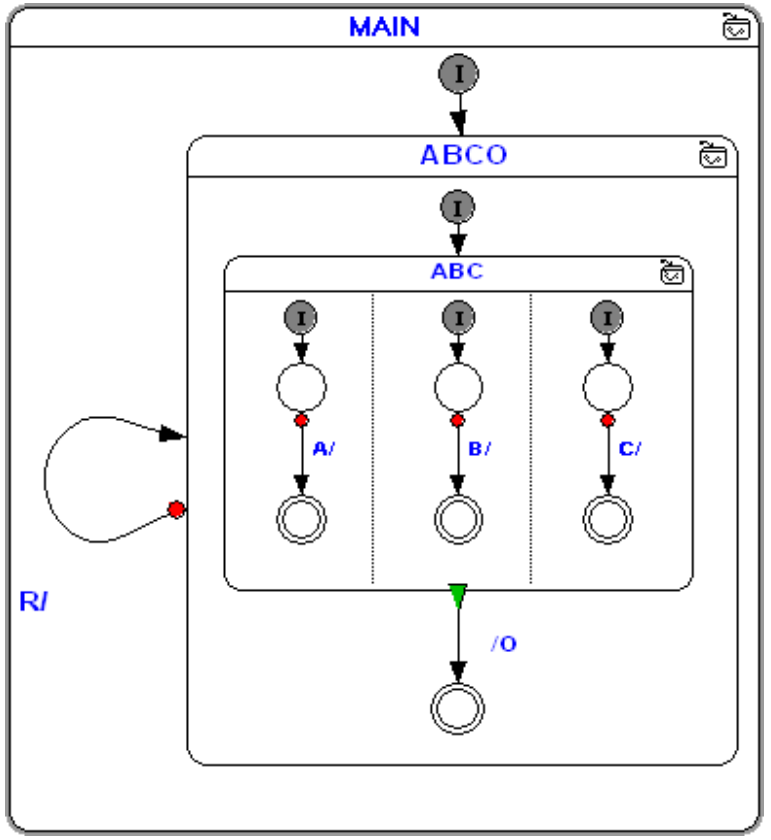

```
loop
   abort
        { await A || await B };
        emit O ;
        halt
   when R
end loop
```

Hierarchical synchronous concurrent automata (Synchronous Statecharts)

ABCRO : from exponential to linear



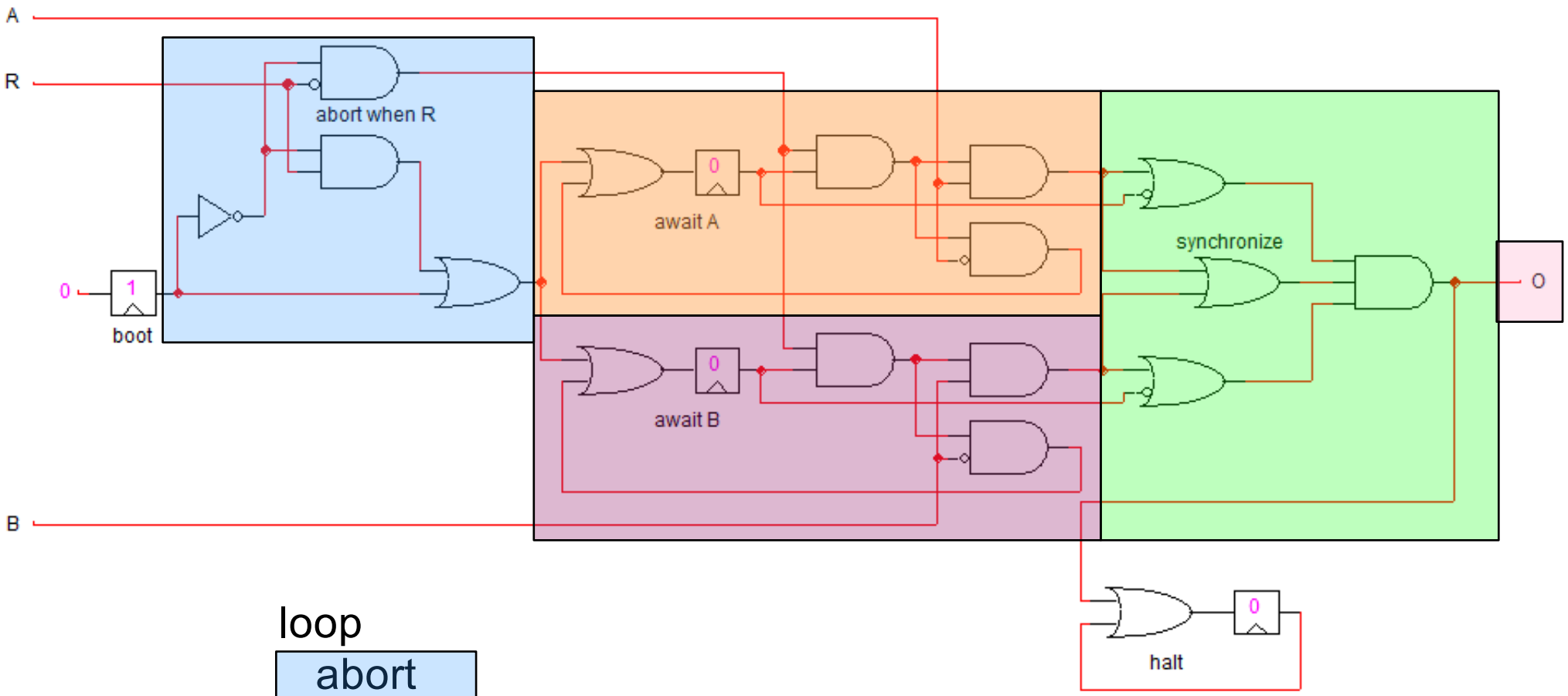
flat automaton



Hierarchical automaton

linear

The Hierarchical ABRO Circuit



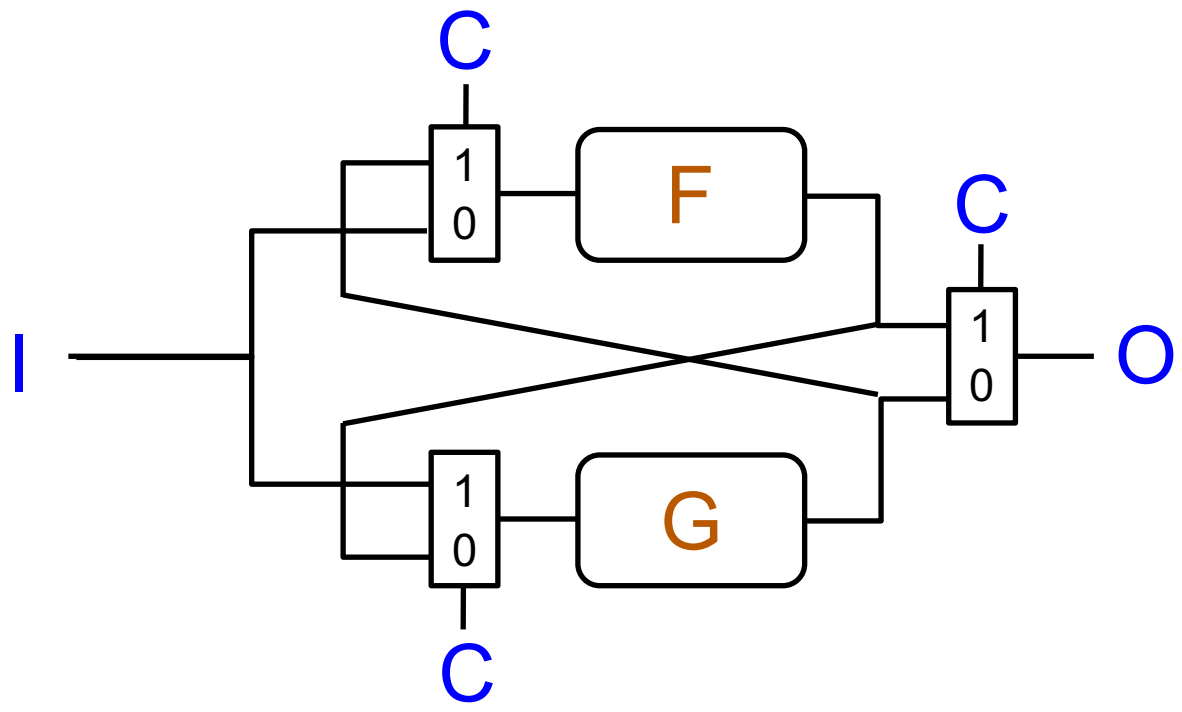

```
loop
   abort
      { await A || await B };
      emit O;
      halt
   when R
end loop
```

Conclusion 1

- Should we still promote DFAs?
- NFAs are deterministic if use in the proper way !
  … and they save an exponential
- Synchronous languages s.t. Esterel / SyncCharts
  … save another exponential (see D. Harel)
- NFA circuits can be efficiently optimized (HW and SW)
  systematically better than human designs !
- Large-scale analysis and verification can be performed by symbolic techniques (BDDs, SAT, SMT)
  …which might be exponential but do work very well in practice

Resource Sharing ⇒ Combinational Cycles

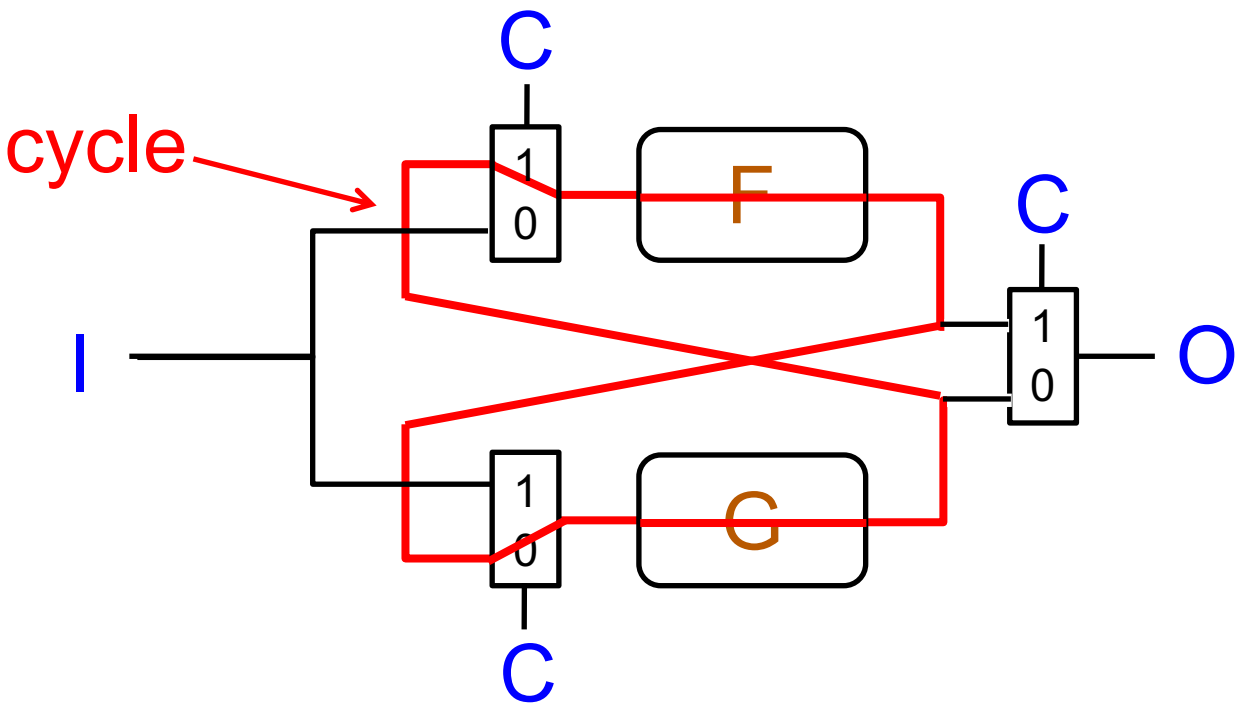
O := if C then F(G(I)) else G(F(I))



Sharad Malik, *Analysis of Cyclic Combinational Circuits*
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 13, no. 7, July 1994

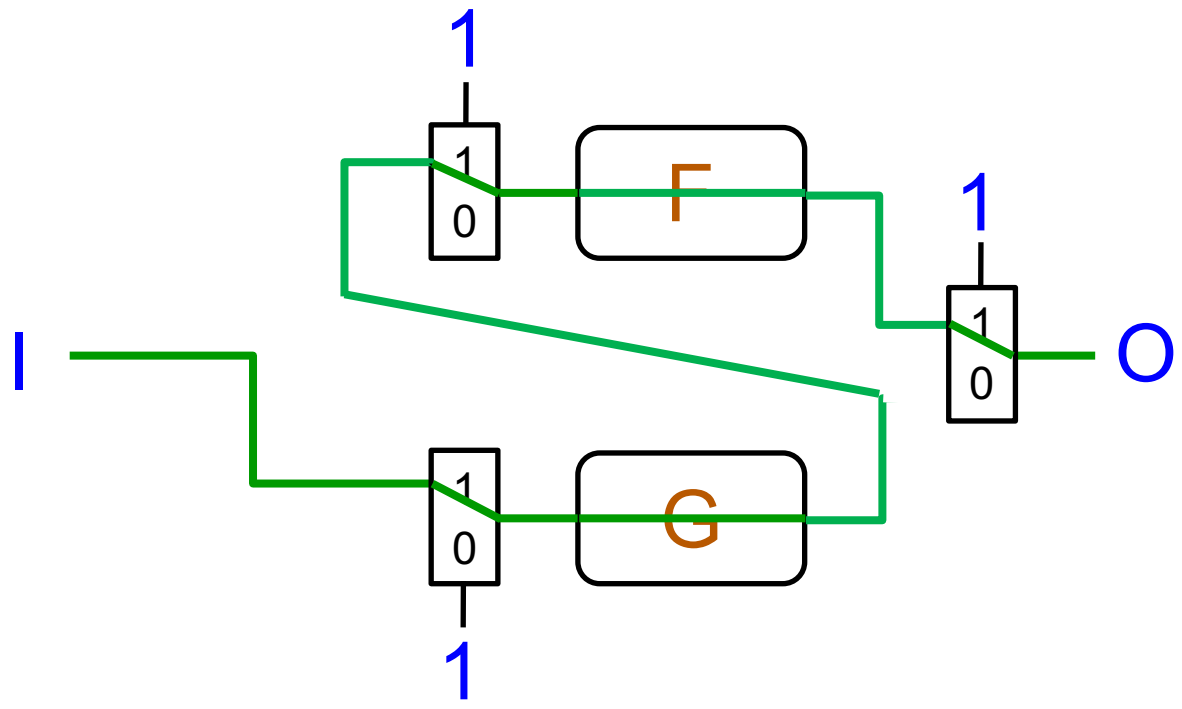
Resource Sharing ⇒ Combinational Cycles

O := if C then F(G(I)) else G(F(I))



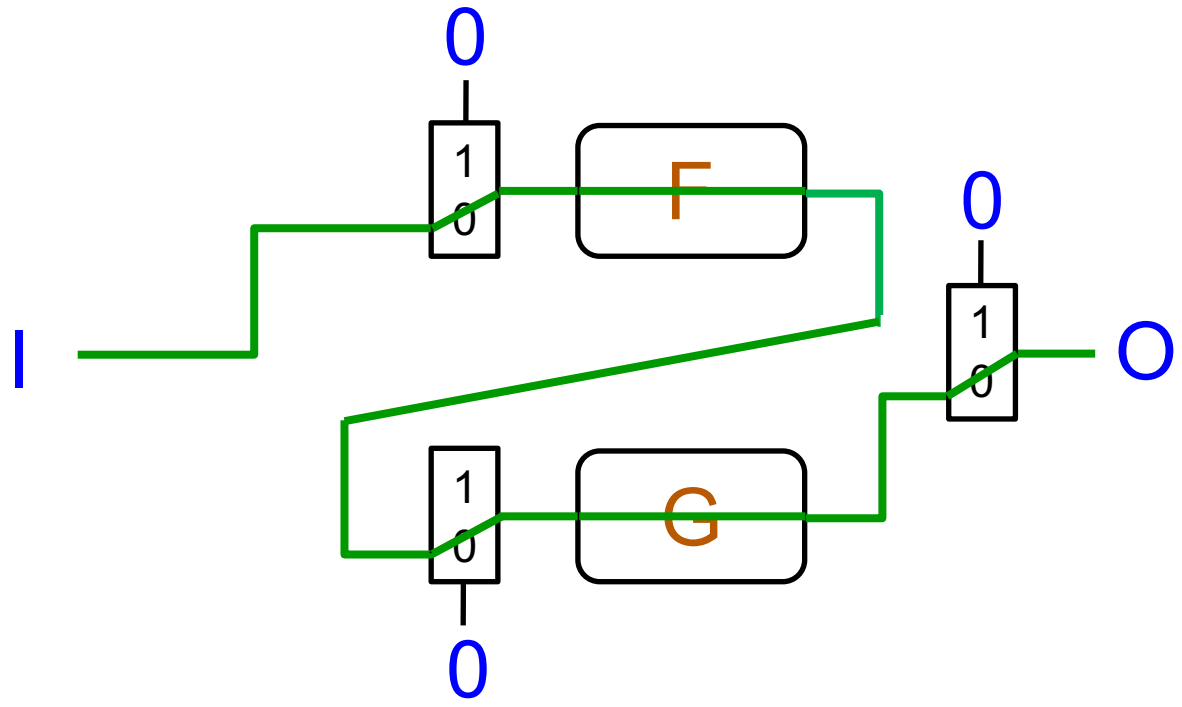
Resource Sharing ⇒ Combinational Cycles

C = 1 ⇒ O = if C then F(G(I)) else G(F(I))



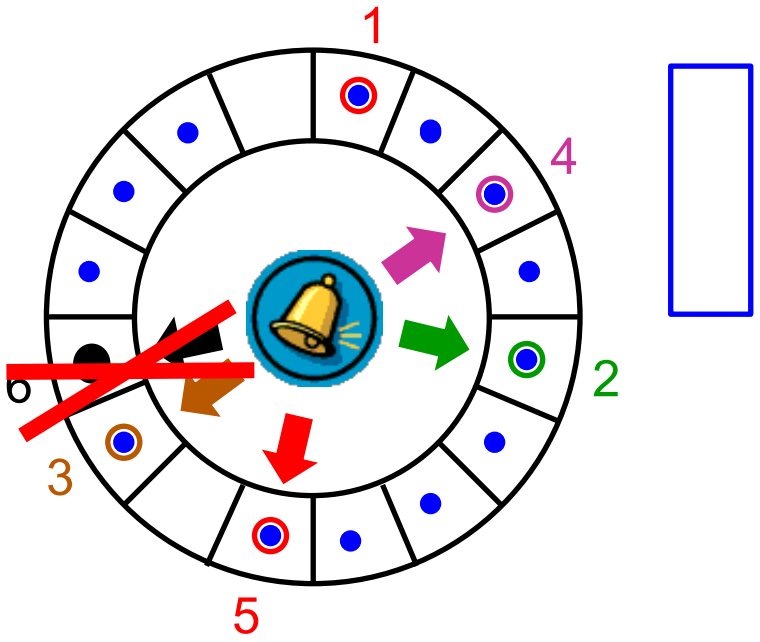
Resource Sharing ⇒ Combinational Cycles

C = 0 ⇒ O = if C then F(G(I)) else G(F(I))



The cycle is logically sound and electrically sound !

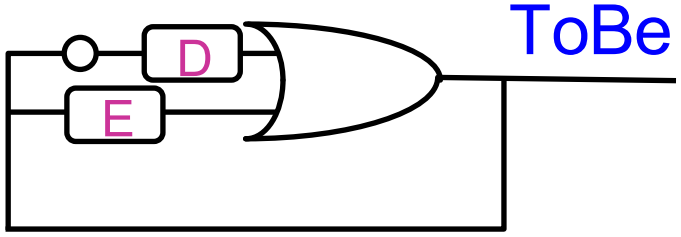
ILD = Instruction Length Decoder



- 16-bytes circular buffer, bytes coming in randomly
- Instruction length depending on the first byte and a variable number of other bytes in the instruction text
- Instruction length potentially arbitrary
- Naturally cyclic design, hard to make acyclic

The Three Kinds of Cyclic Circuits

- Good : electrical stabilization, logical consistency
  previous examples
- Bad : no electrical stabilization, no logical consistency
  X = X
  X = ¬X
- Weird : unique logical solution, but electrical stabilization depending on wire and gate delays
  Hamlet : ToBe = ToBe or not ToBe



no electrical stabilization when starting from ToBe =0 with D=2 and E=5

→ Intuitionistic logic !

Constructive Boolean Circuit Logic

- Circuit C, input vector : I = inputs $\rightarrow$ {0,1}
- formulae : $I \vdash e = b$, written $e = b$ when I constant

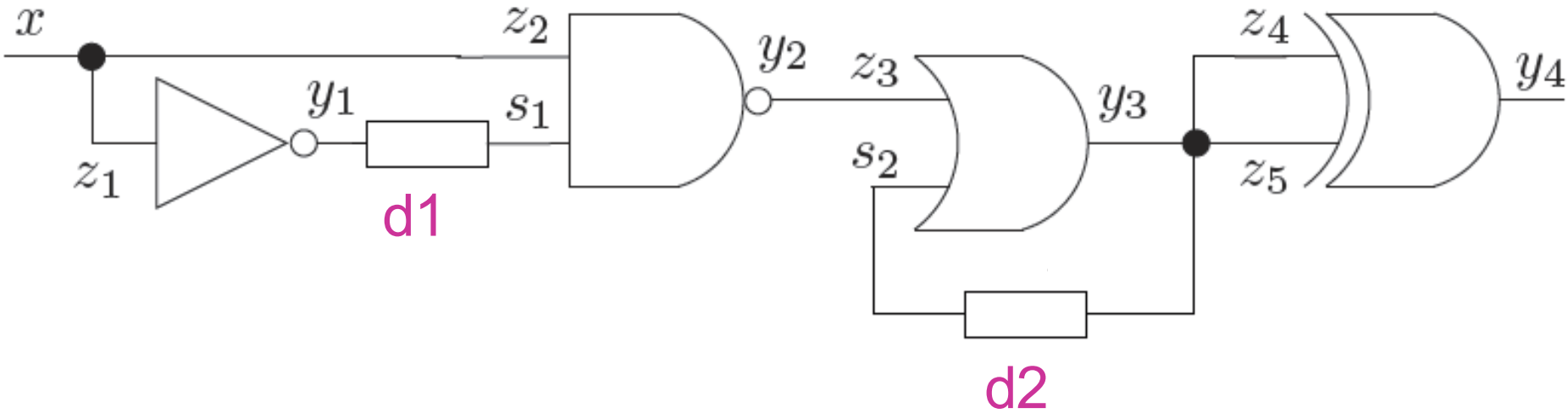
$$\frac{l \text{ input}}{l = I(l)} \qquad \frac{e = 0}{\neg e = 1} \qquad \frac{e = 1}{\neg e = 0}$$

$$\frac{e = 0}{e \wedge e' = 0} \qquad \frac{e' = 0}{e \wedge e' = 0} \qquad \frac{e = 1 \quad e' = 1}{e \wedge e' = 1}$$

$$\frac{e = 1}{e \vee e' = 1} \qquad \frac{e' = 1}{e \vee e' = 1} \qquad \frac{e = 0 \quad e' = 0}{e \vee e' = 0}$$

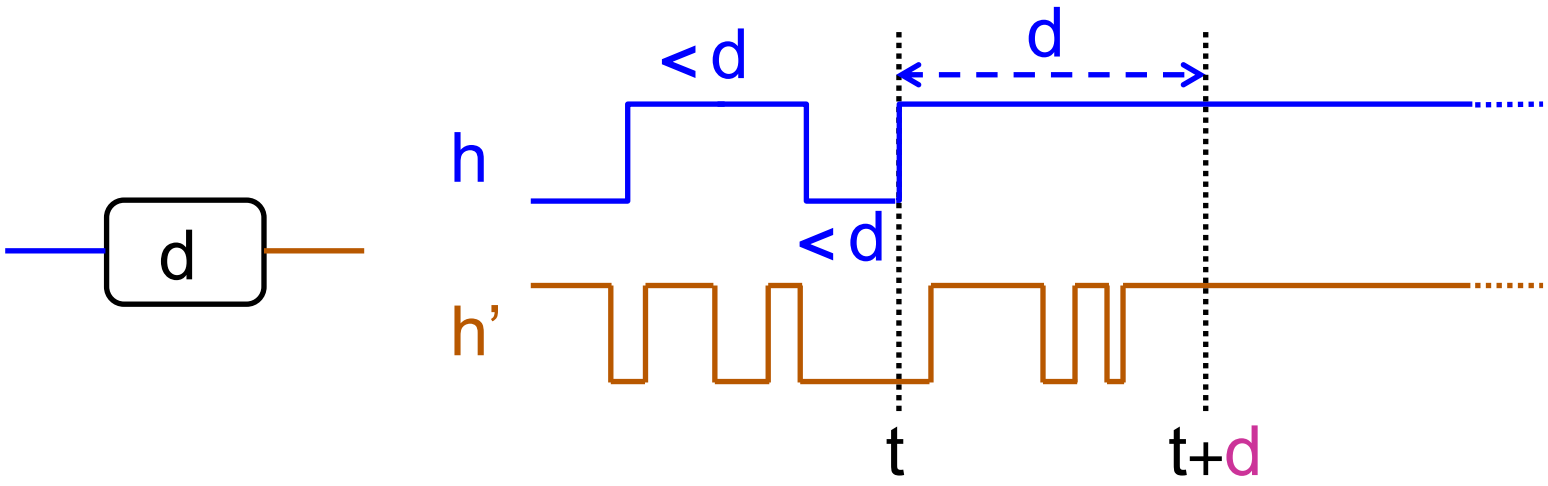
$$\frac{x := e \in C \quad e = b}{x = b}$$

Circuits With Delays



- Logical gates : zero-delay, grouping possible
  - polynomial notation : $y_1 = \overline{x}$, $s_2 = \overline{s_1 x} + s_2 = \overline{s_1} + \overline{x} + s_2$
- Explicit delay nodes
- At least one electrical delay per cycle

UN-Delay and Stability

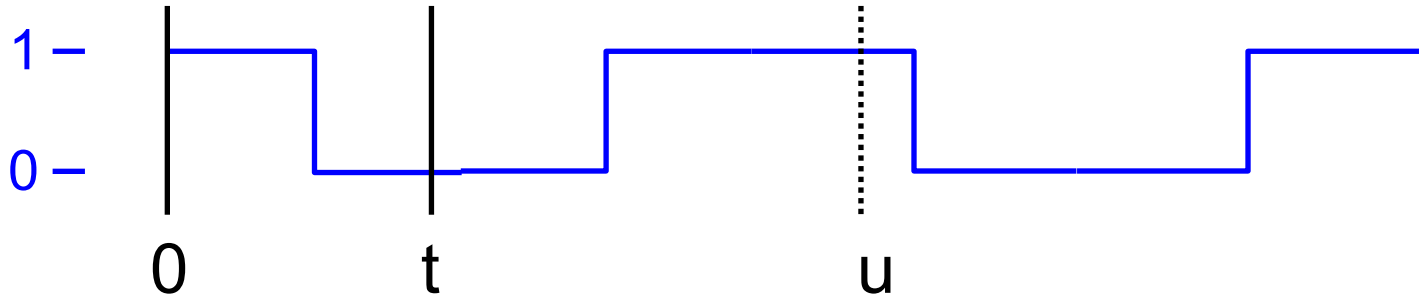


UN-delay : $d \in \mathbb{R}^+$ $\quad h[t,u)=b$ and $t+d < u \Rightarrow h'[t+d,u)=b$

- A history h is stable at b after a delay d if $h[d,\infty) = b$
- Otherwise, h is called unstable or oscillating

Intuitionistic Negation

- Goal : to represent the not gate
  - x stable to 1 : $h[t,u) \models x$
  - logical opposite : $h[t,u) \not\models x$
  - but the logical opposite is satisfied by any unstable signal
  - we want x stable to 0, i.e. $\neg x \equiv \overline{x}$, which is different!



neither $h[t,u) \models x$ nor $h[t,u) \models \overline{x}$

$h[t,u) \models \neg\phi$ iff $\forall [t',u') \subset [t,u).\ h[t',u') \not\models \phi$

ϕ is never satisfied by h on [t,u)

different from ϕ is not satisfied by h !

Summary of UN-Logic Definition

$h[t,u) \models R$ if $\forall \tau \in [t,u).\ h(\tau) \in R$

$h[t,u) \models \phi \wedge \psi$ if $h[t,u) \models \phi$ and $h[t,u) \models \psi$

$h[t,u) \models \phi \vee \psi$ if $h[t,u) \models \phi$ or $h[t,u) \models \psi$

$h[t,u) \models \neg\phi$ if $\forall [t',u') \subset [t,u).\ h[t',u') \not\models \phi$

$h[t,u) \models \phi \supset \psi$ if $\forall [t',u') \subset [t,u).\ h[t',u') \models \phi \Rightarrow h[t',u') \models \psi$

$h[t,u) \models \phi \equiv \psi$ if $\forall [t',u') \subset [t,u).\ h[t',u') \models \phi \Leftrightarrow h[t',u') \models \psi$

$h[t,u) \models \Diamond_d \phi$ if $t+d < u$ if $h[t+d,u) \models \phi$

Notation : $\phi \models \psi$ iff $\forall h,t,u.\ h[t,u) \models \phi \Rightarrow h[t,u) \models \psi$

Deduction Rules

◊true $\dfrac{}{\Diamond_d 1}$

for $\Phi(C,I)$ fixed,
$\Phi(C,I) \vdash$ ... implicit everywhere

◊bool $\dfrac{\Diamond_d R \quad d \le e \quad R \subseteq S}{\Diamond_e S}$ weakening

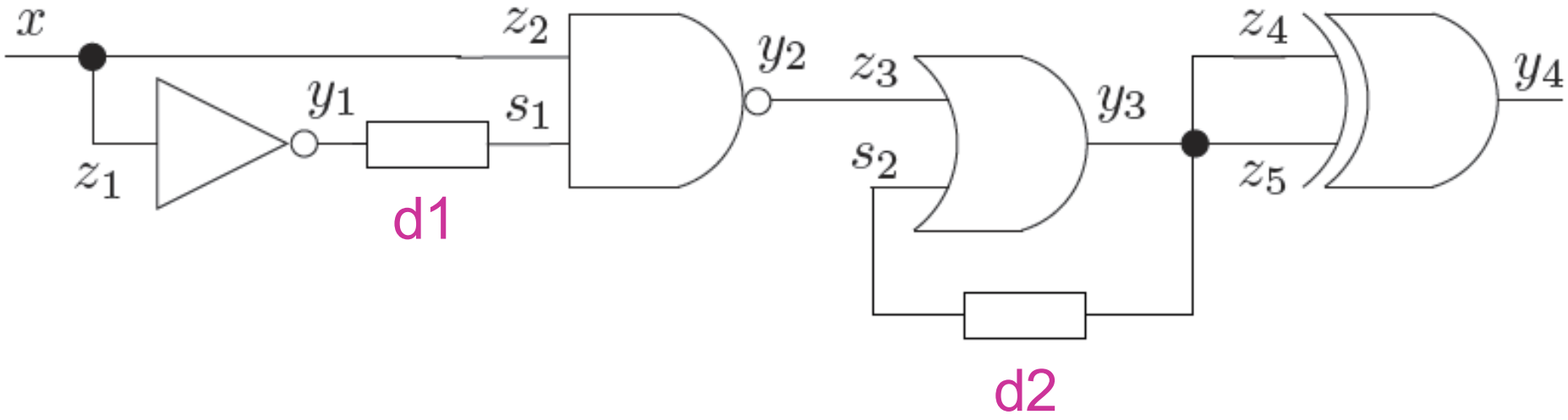
◊chain $\dfrac{\Diamond_d R \quad R \supset \Diamond_e S}{\Diamond_{d+e} S}$

transition chaining
$y :=_e x \Leftrightarrow (x \supset \Diamond_e y) \vee (\bar{x} \supset \Diamond_e \bar{y})$

◊join $\dfrac{\Diamond_d S \qquad \Diamond_e T}{\Diamond_{max(d,e)} S \cap T}$ gate inputs gathering

+ classical Boolean rules for regions
  (OK since applied only to stable signals)
+ arithmetic operations on delays

Deduction at Work



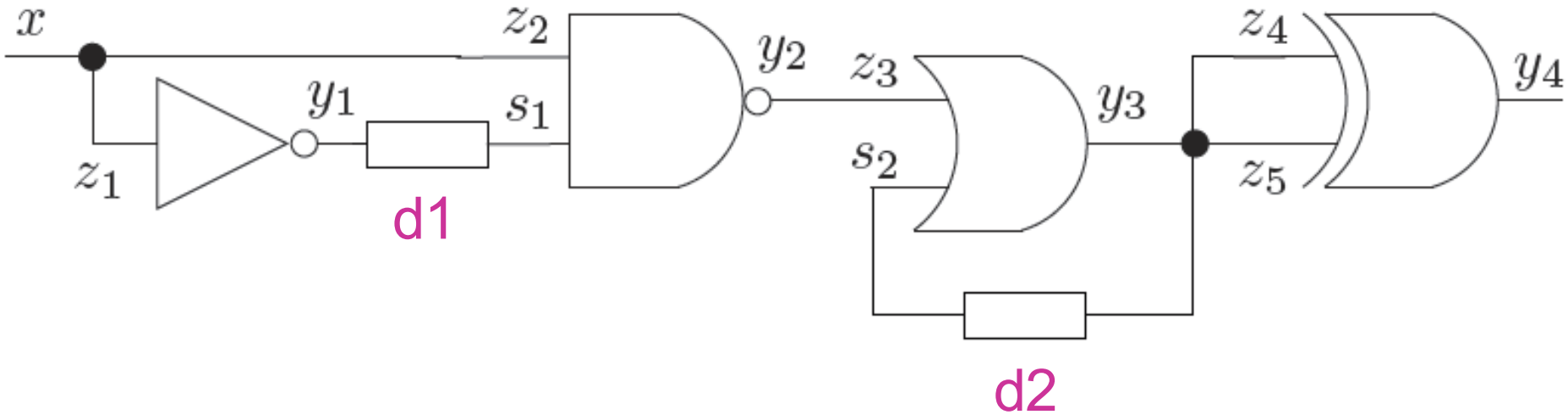
$\Phi(C) = (s_1 :=_{d1} \bar{x}) \wedge (s_2 :=_{d2} \bar{x}+\bar{s_1}+s_2)$

case x=0
i.e. $\Diamond_0 \bar{x}$

$$\dfrac{\dfrac{\dfrac{\Diamond_0 \bar{x}}{\Diamond_{d1} s_1}\,\Diamond\text{chain} \qquad \dfrac{\dfrac{\Diamond_0 \bar{x}}{\Diamond_0(\bar{x}+\bar{s_1}+s_2)}\,\Diamond\text{bool}}{\Diamond_{d2} s_2}\,\Diamond\text{chain}}{\Diamond_{max(d1,d2)}\, s_1 s_2}\,\Diamond\text{join}$$

x=0 ⇒ region $s_1 s_2$ reached in time max(d1,d2)

Deduction at Work



$\Phi(C) = (s_1 :=_{d1} \overline{x}) \wedge (s_2 :=_{d2} \overline{x}+\overline{s_1}+s_2)$

case x=1
i.e. $\Diamond_0 x$

$$\dfrac{\dfrac{\dfrac{\Diamond_0 x}{\Diamond_{d1}\overline{s_1}}\,\Diamond\text{chain} \qquad \dfrac{\dfrac{\Diamond_0 x}{\Diamond_{d1}\overline{s_1}}\,\Diamond\text{chain}}{\Diamond_{d1}(\overline{x}+\overline{s_1}+s_2)}\,\Diamond\text{bool}}{\Diamond_{d1+d2} s_2}\,\Diamond\text{chain}}{\Diamond_{\max(d1,\, d1+d2)}\ \overline{s_1}s_2}\,\Diamond\text{join}$$

x=1 ⇒ region $\overline{s_1}s_2$ reached in time d1+d2

The Key Theorems

- Theorem 2 : equivalence of $\models$ and $\vdash$ for circuits

$$\forall\ C,I,\Theta.\ \ \Phi(C,I) \models \Theta \Leftrightarrow \Phi(C,I) \vdash \Theta$$

- Theorem 3 : Intuitionism of $\models$

$$\forall\ C,I,\Theta.\ \ \Phi(C,I) \models \Theta \Rightarrow \exists \theta \in \Theta.\ \ \Phi(C,I) \models \theta$$

A disjunction (even infinite) can only by validated by one of its members (immediate from Theorem 2 and definition of $\Phi(C,I) \vdash \Theta$)

The Central Result

Corollary : stabilization is deterministic

Let s a delay assignment for C and I an input vector. Then the histories h of s in (C,I) have only two possible behaviors:

1. all the h stabilize to the same value
2. there is at least one oscillating valid history h

$\Rightarrow$ A circuit is constructive iff its outputs cannot oscillate

Proof : let $\Theta = \Diamond_1 s \vee \Diamond_1 \bar{s} \vee \Diamond_2 s \vee \Diamond_2 \bar{s} \vee \Diamond_3 s \vee \Diamond_3 \bar{s} \vee \ldots$

then Θ (infinite) expresses that s stabilizes eventually

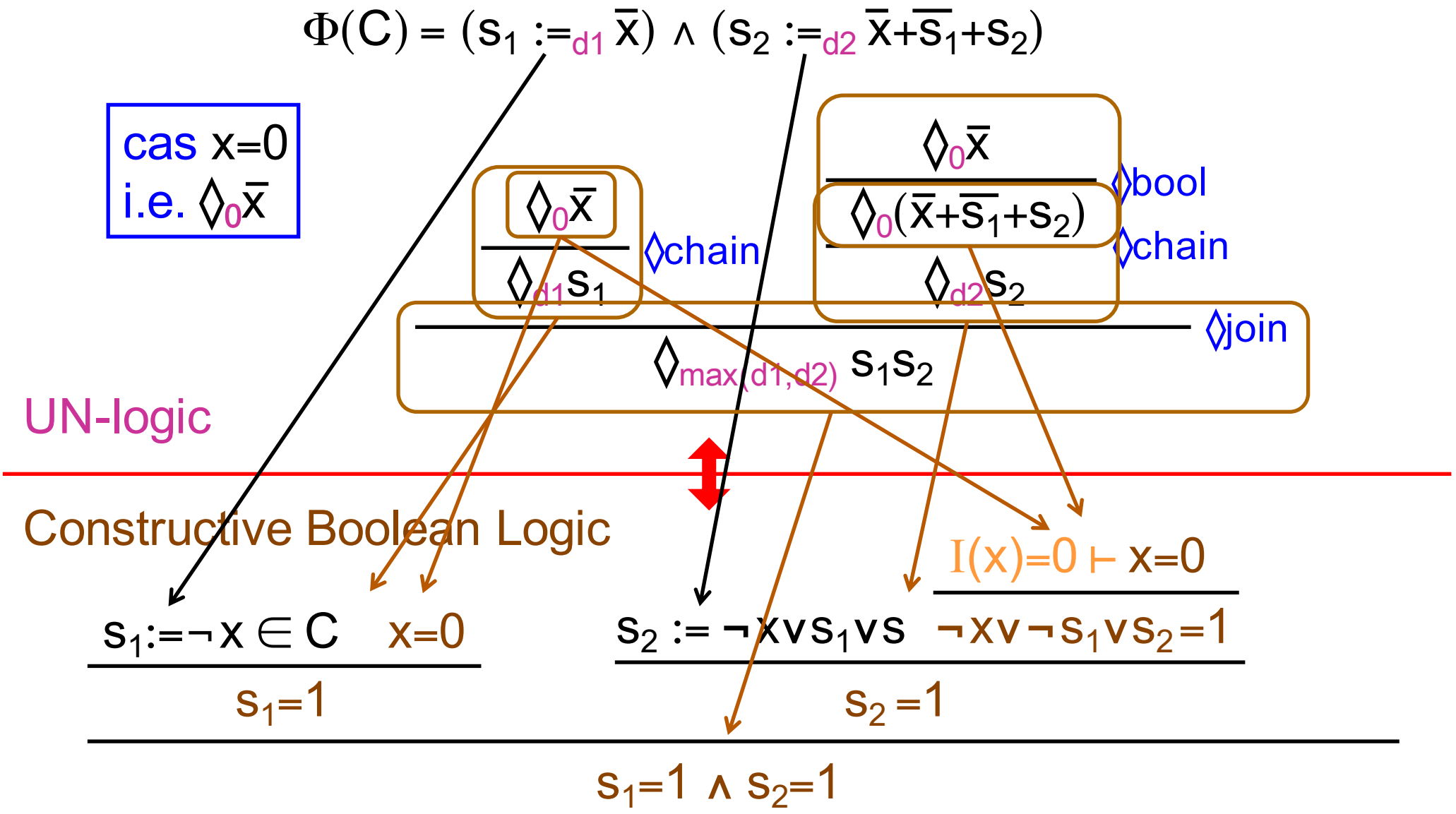
case 1 : $\Phi(C,I) \models \Theta$. Then $\Phi(C,I) \models \theta_k$ for some k by thm.3 (intuitionism), for instance $\theta_k = \Diamond_m s$.

Hence $\forall h.\ h \models \Phi(C,I) \Rightarrow h \vdash \Diamond_m s$, any h stabilizes to 1

case 2 : $\Phi(C,I) \not\models \Theta$. Then $\forall h.\ h \models \Phi(C,I) \Rightarrow h \models \Theta$ is impossible.

Hence $\exists h.\ h \models \Phi(C,I) \wedge h \not\models \Theta$, and this h oscillates for s

Proof Transformation Example



There We Are !

- For given delays, UN-provability vs. $\vdash$ is a necessary and sufficient condition for UN-stabilization vs. $\models$
- But any proof with delays can be transformed into a proof without delays, and conversely

Which means: Provability in Constructive Boolean Logic exactly reflects electrical constructivity for all delays

Bonus: given the delays, proof-construction based simulation computes the maximal reaction time w.r.t. each input

Conclusion

- Complete study of cyclic circuits in the UN-delay model,
- Showing equivalence between model ⊨ and deduction ⊢, ignoring transients, oscillations, metastability etc

Electrical stabilization ⇔ constructive boolean provability
(with delays) (without delays)

Light speed Curry-Howard :

Intuitionistic deduction reflects electrical propagation

References

Constructive Boolean Circuits and the Exactness of Timed Ternary Simulation
M. Mendler, T. Shiple et G. Berry.
Formal Methods in System Design, Vol.40, No.3, pp. 283-329, Springer (2012).

Constructive Analysis of Cyclic Circuits
T. Shiple, G. Berry et H. Touati.
Proc. *Int. Design and Testing Conferen*ce IDTC'96, Paris, France (1996).

Asynchronous Circuits
J. Brzozowski et C-J. Seger.
Springer-Verlag (1995).