Cascade Decomposition of Asynchronous Zielonka Automata

Paul Gastin
LMF, ENS Paris-Saclay, IRL ReLaX

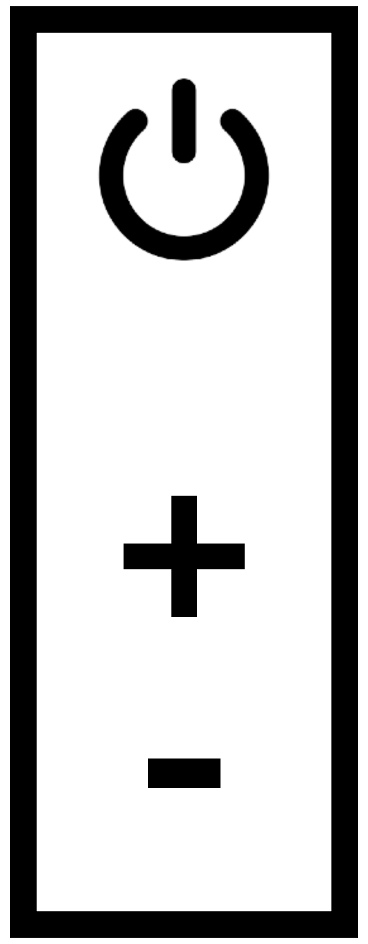
Joint Work with Bharat Adsul (IIT Bombay), Shantanu Kulkarni (IIT Bombay), Saptarshi Sarkar (IIT Bombay) and Pascal Weil (LaBRI, ReLaX)

Based on CONCUR'20, LMCS'22, CONCUR'22 and submitted work

Outline

- Labelling functions, sequential transducers and cascade product
- Krohn-Rhodes theorem for aperiodic/regular word languages
- Model of concurrency: Mazurkiewicz traces and asynchronous Zielonka automata
- Asynchronous labelling functions, transducers and cascade product
- Propositional dynamic logic for traces
- Conclusion

Labelling function $\theta\colon \Sigma^* \to \Gamma^*$

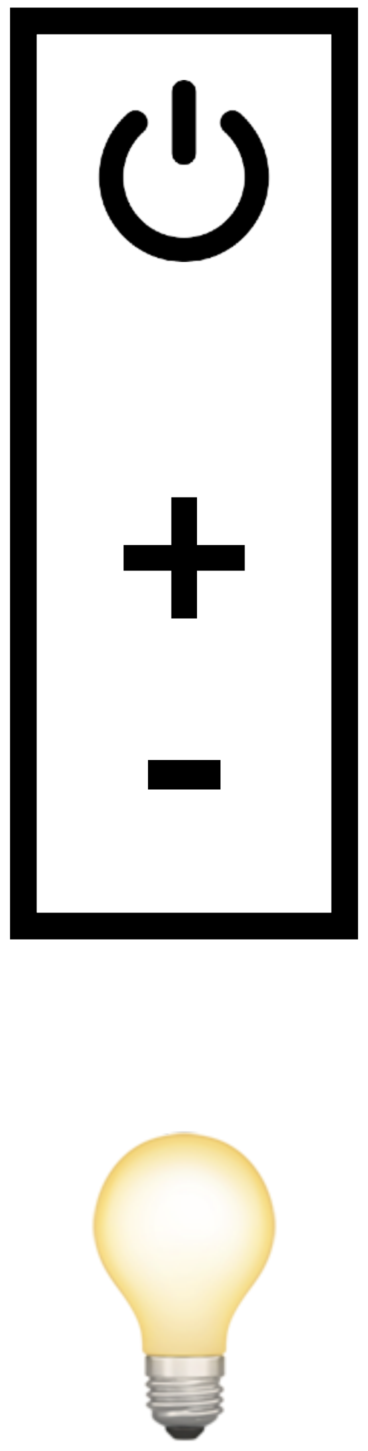


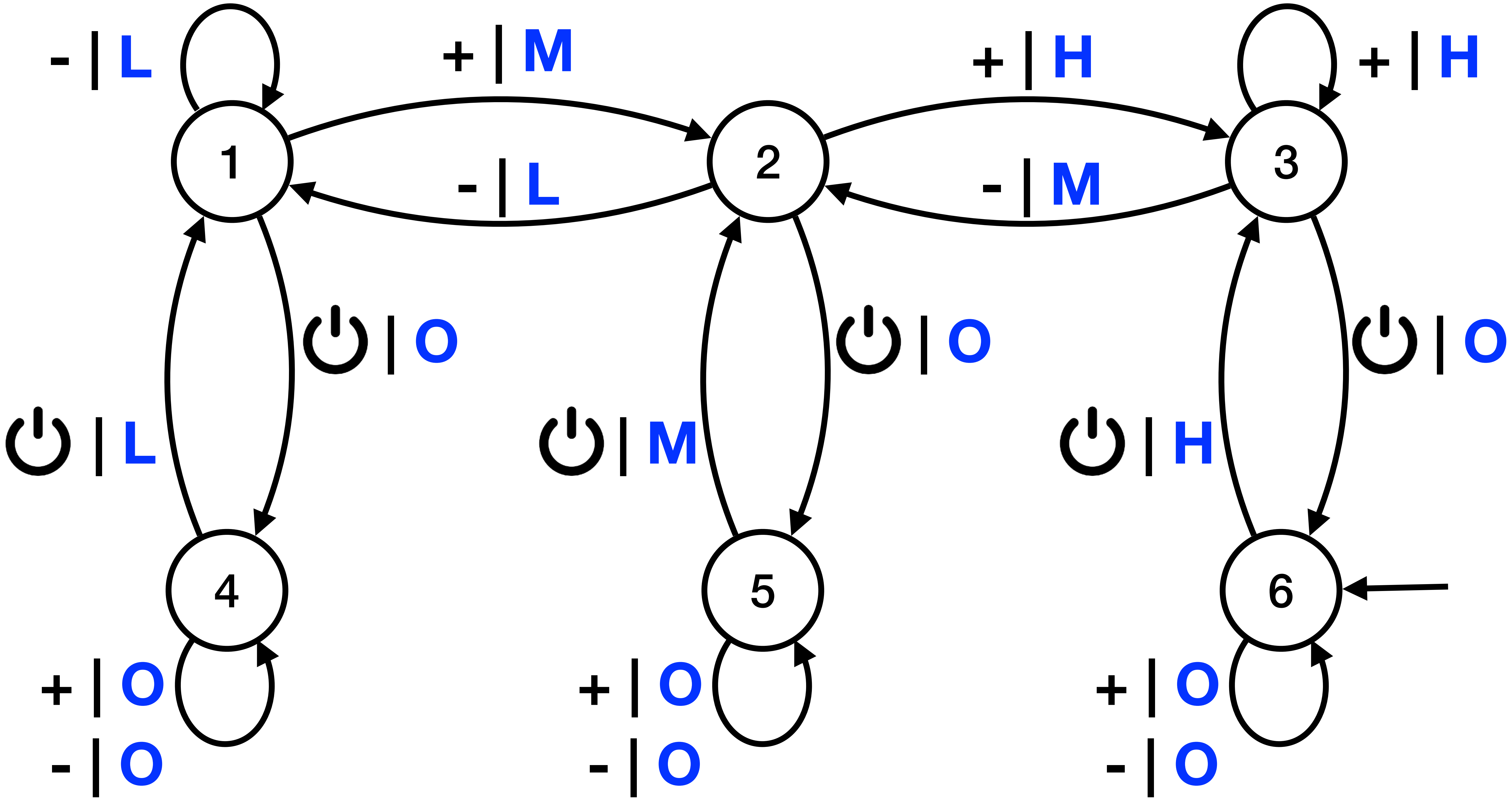
-	⏻	-	⏻	+	⏻	-	-	⏻	⏻	+	+	+	⏻
O	H	M	O	O	M	L	L	O	L	M	H	H	O

$\Sigma = \{$ ⏻ $, -, + \}$ and $\Gamma = \{\mathrm{O}, \mathrm{L}, \mathrm{M}, \mathrm{H}\}$

- ⏻ - ⏻ + ⏻ - - ⏻ ⏻ + + + ⏻

O H M O O M L L O L M H H O





letter-to-letter sequential transducer $\mathscr{T} = (Q, q_0, \Sigma, \delta, \Gamma, \mu)$

Composition - Cascade product

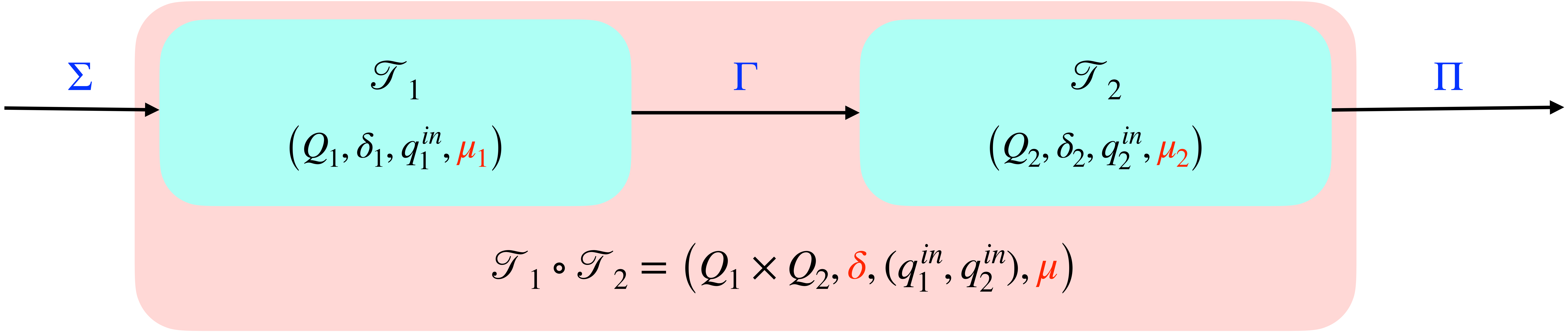
- Composition of labelling functions

$$\Sigma^* \xrightarrow{\theta_1} \Gamma^* \xrightarrow{\theta_2} \Pi^*$$

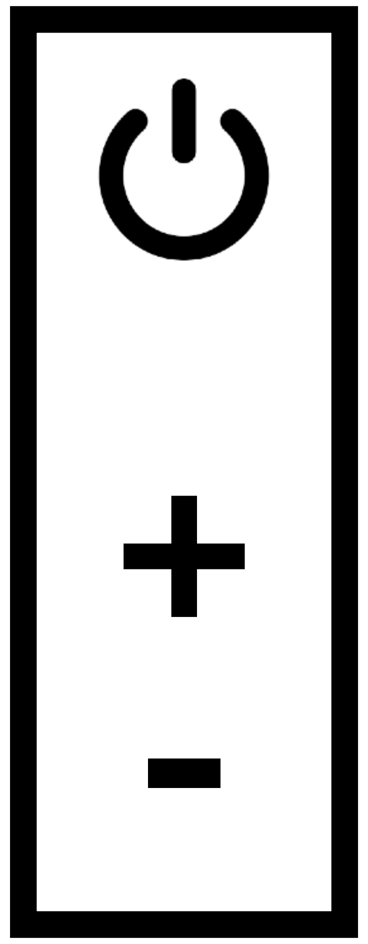
$$\boxed{\delta((p,q),a) = \big(\delta_1(p,a), \delta_2(q, \mu_1(p,a))\big)}$$

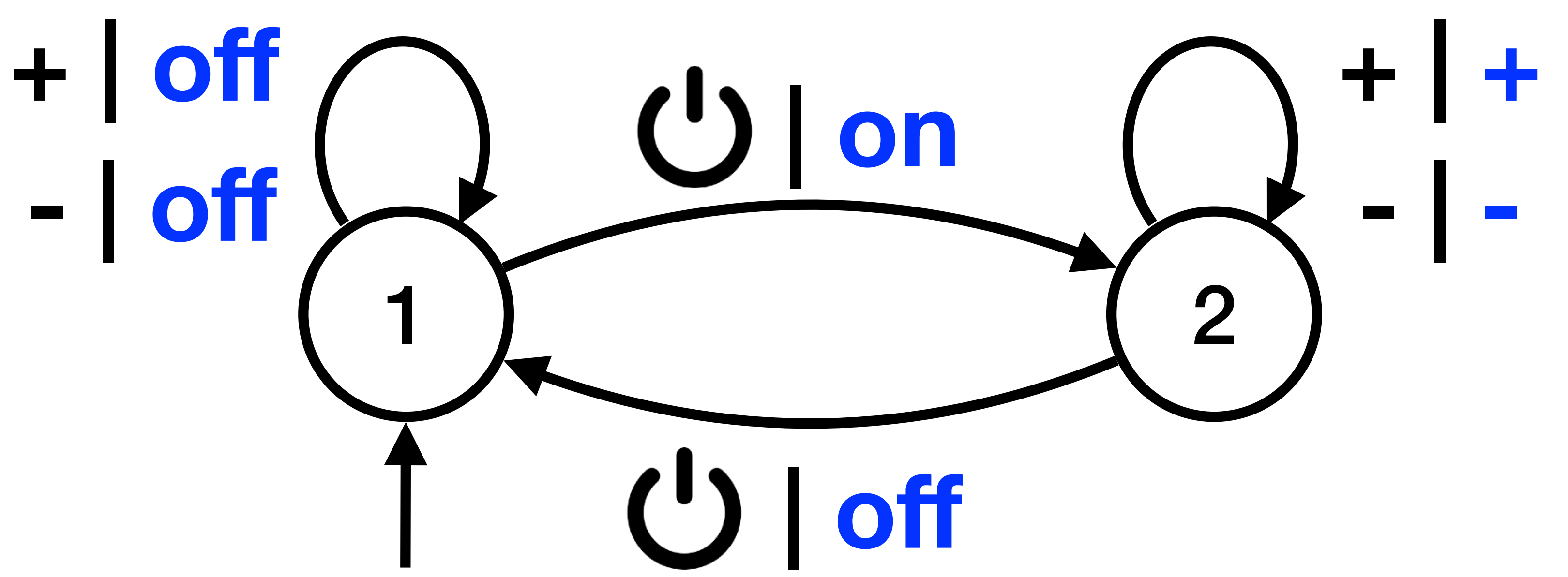
$$\boxed{\mu((p,q),a) = \mu_2(q, \mu_1(p,a))}$$

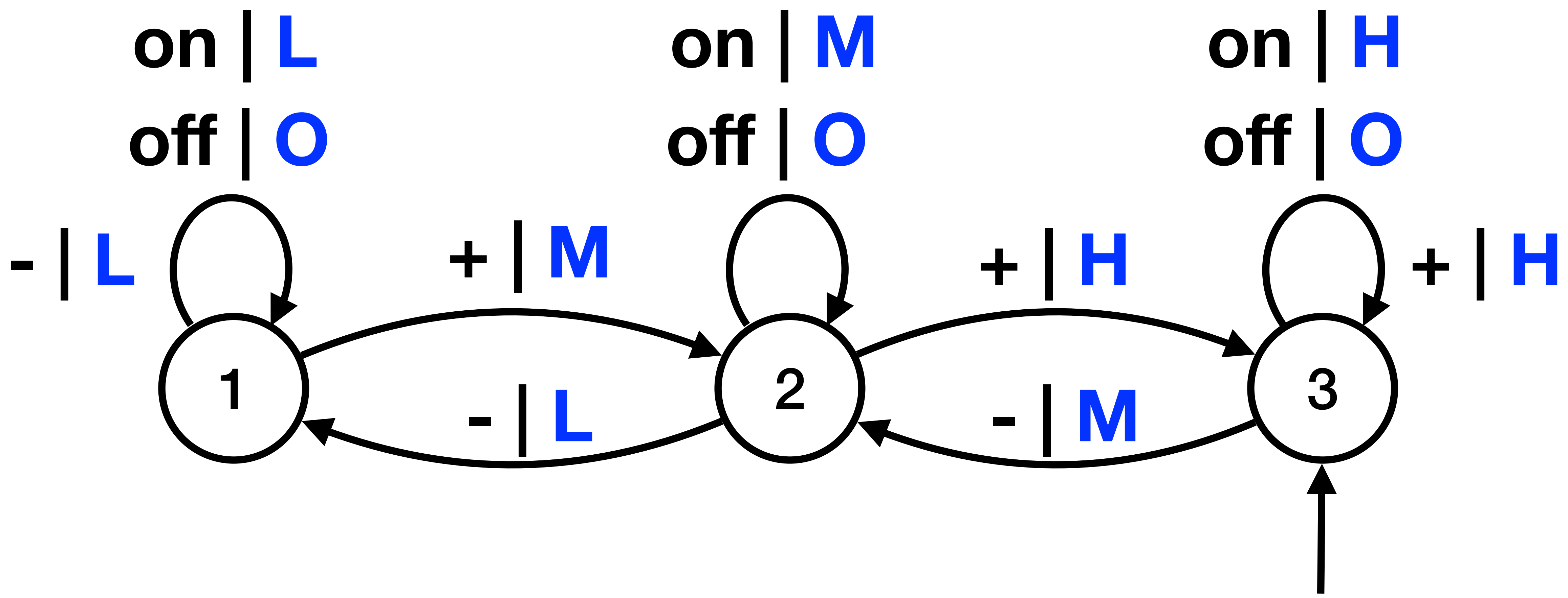
- Cascade product of (letter-to-letter) sequential transducers



-	⏻	-	⏻	+	⏻	-	-	⏻	⏻	+	+	+	⏻
O	H	M	O	O	M	L	L	O	L	M	H	H	O







Outline

- Labelling functions, sequential transducers and cascade product
- Krohn-Rhodes theorem for aperiodic/regular word languages
- Model of concurrency: Mazurkiewicz traces and asynchronous Zielonka automata
- Asynchronous labelling functions, transducers and cascade product
- Propositional dynamic logic for traces
- Conclusion

Krohn-Rhodes

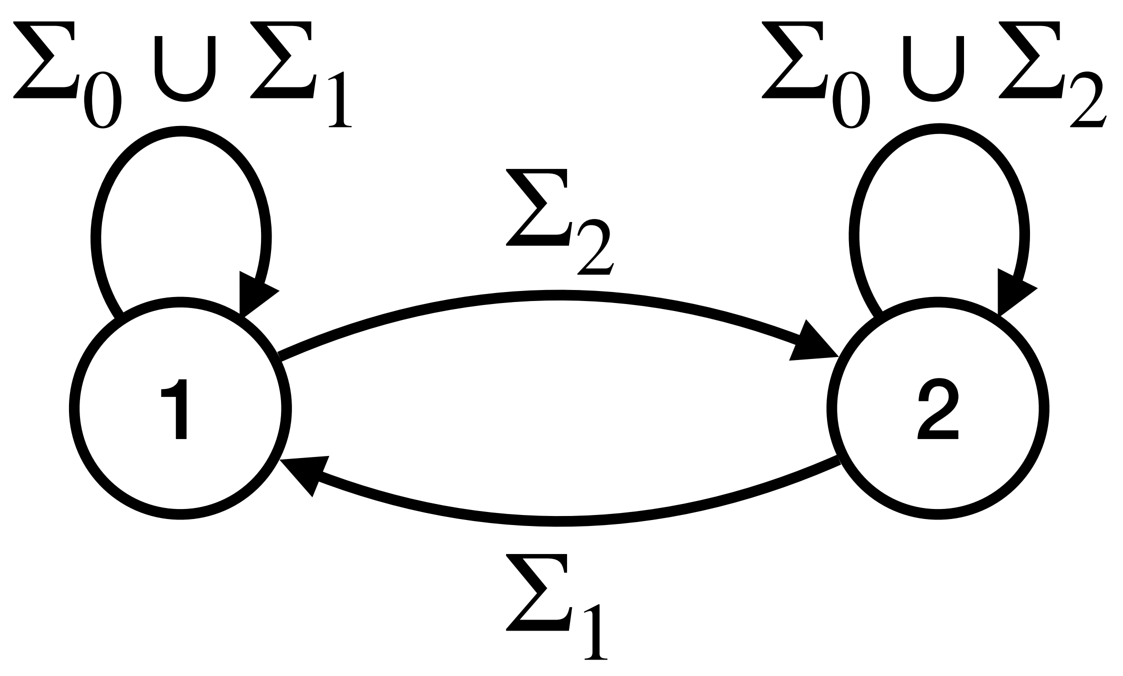
Theorem

Any (letter-to-letter) sequential transducer $\mathcal{T}$ can be realised by a cascade product of reset or permutation transducers:

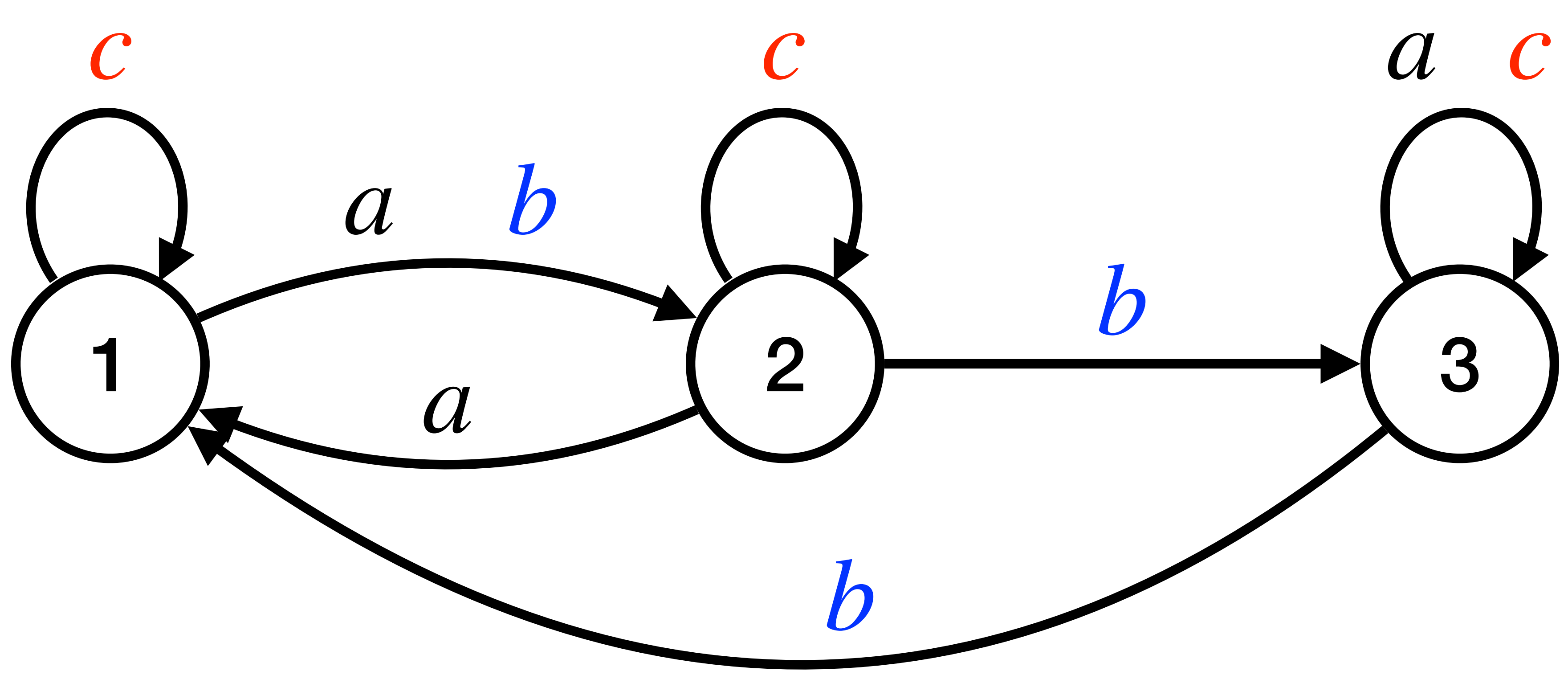
$$\mathcal{T} \equiv \mathcal{T}_1 \circ \mathcal{T}_2 \circ \cdots \circ \mathcal{T}_n$$

Reset or Permutation is a property of the underlying input automaton:

Reset - $\mathcal{U}_2$

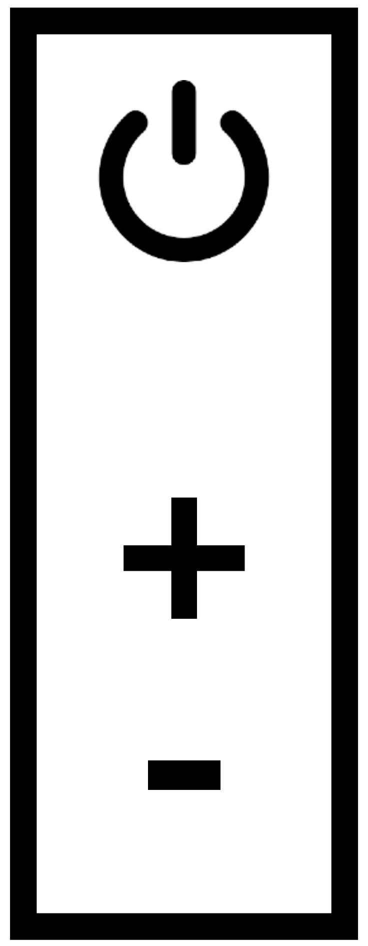


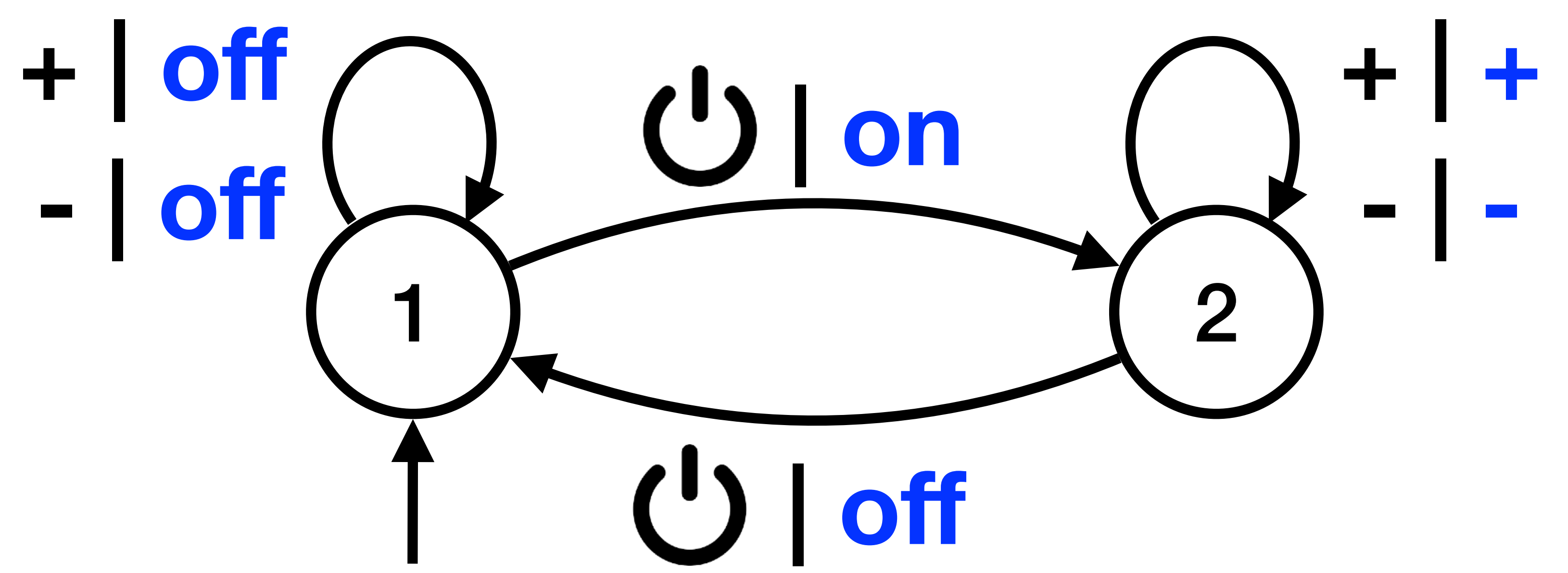
Permutation



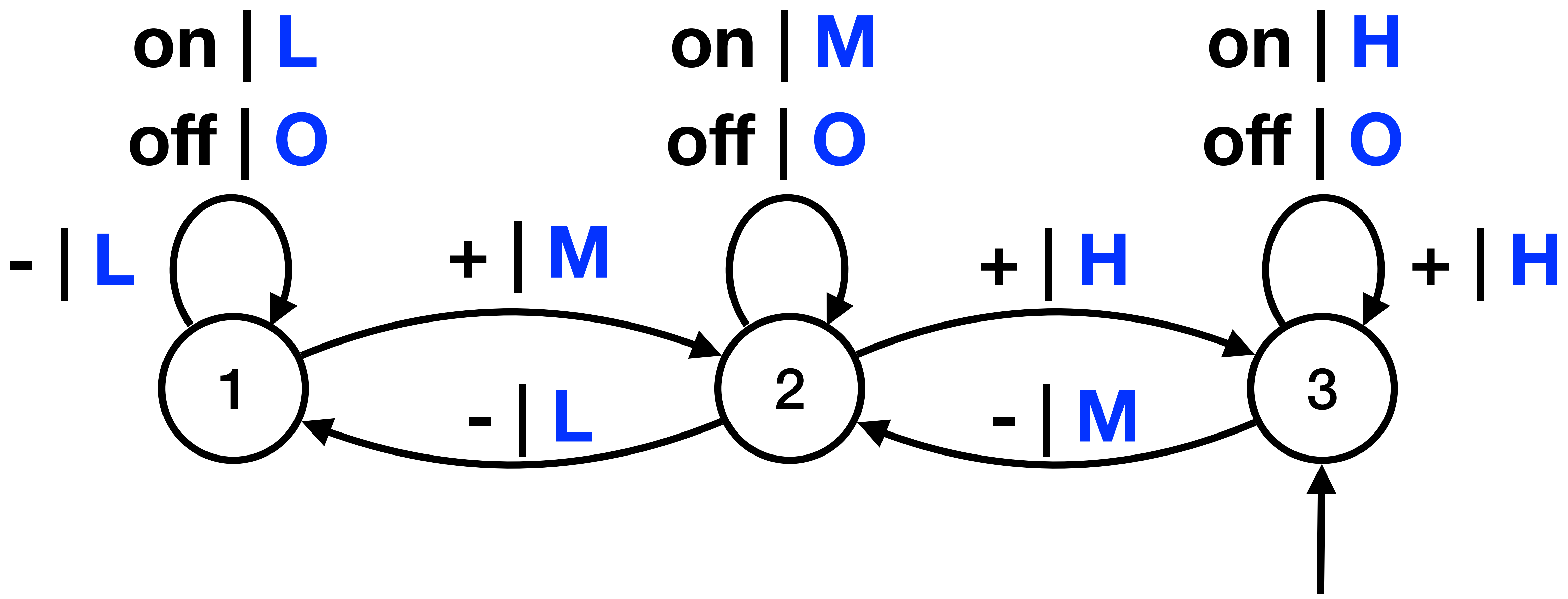
- ⏻ - ⏻ + ⏻ - - ⏻ ⏻ + + + ⏻

O H M O O M L L O L M H H O





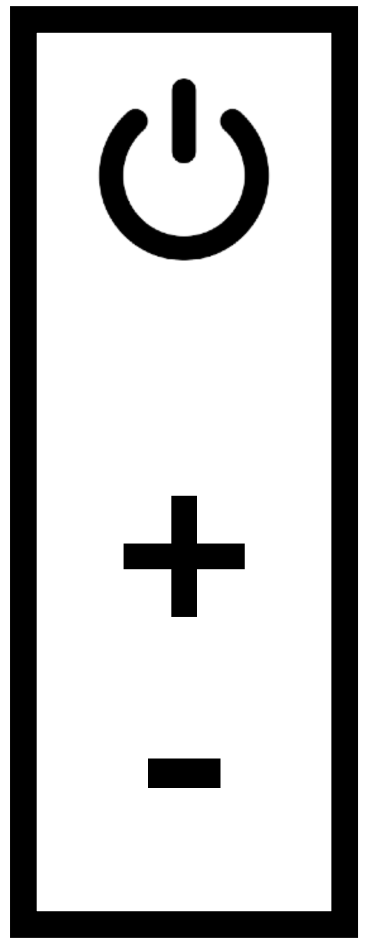
Permutation

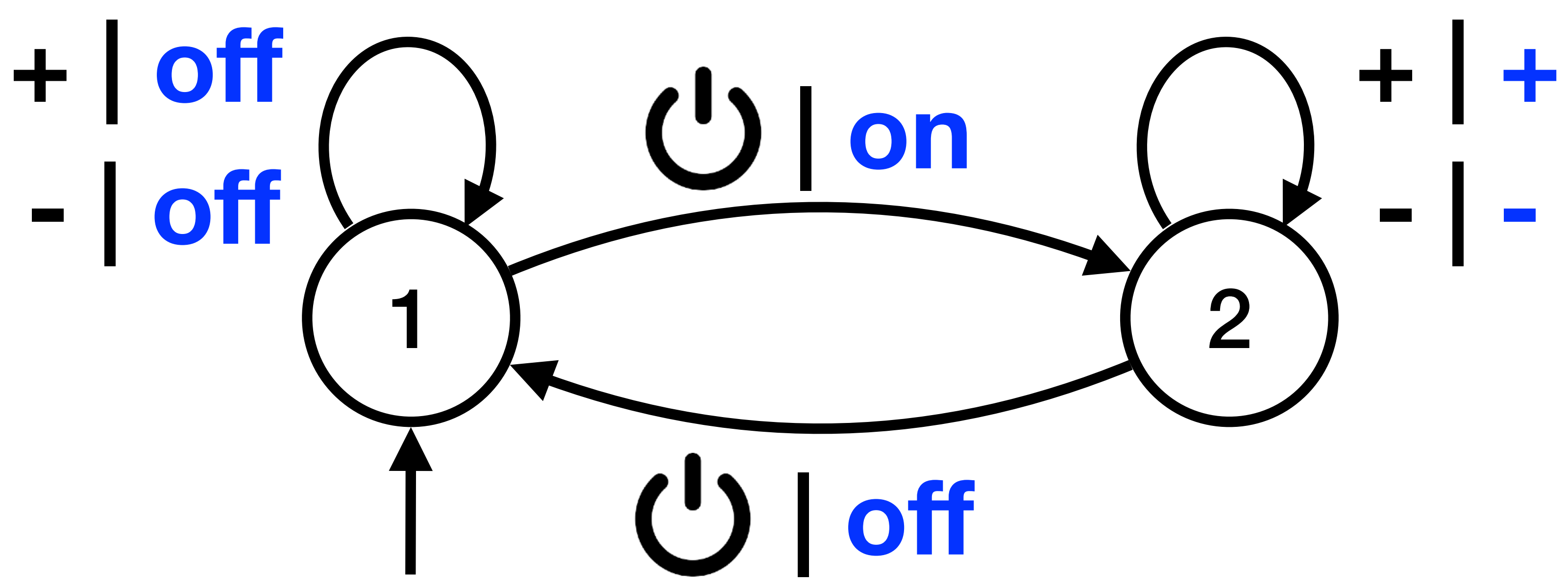


Neither Reset nor Permutation

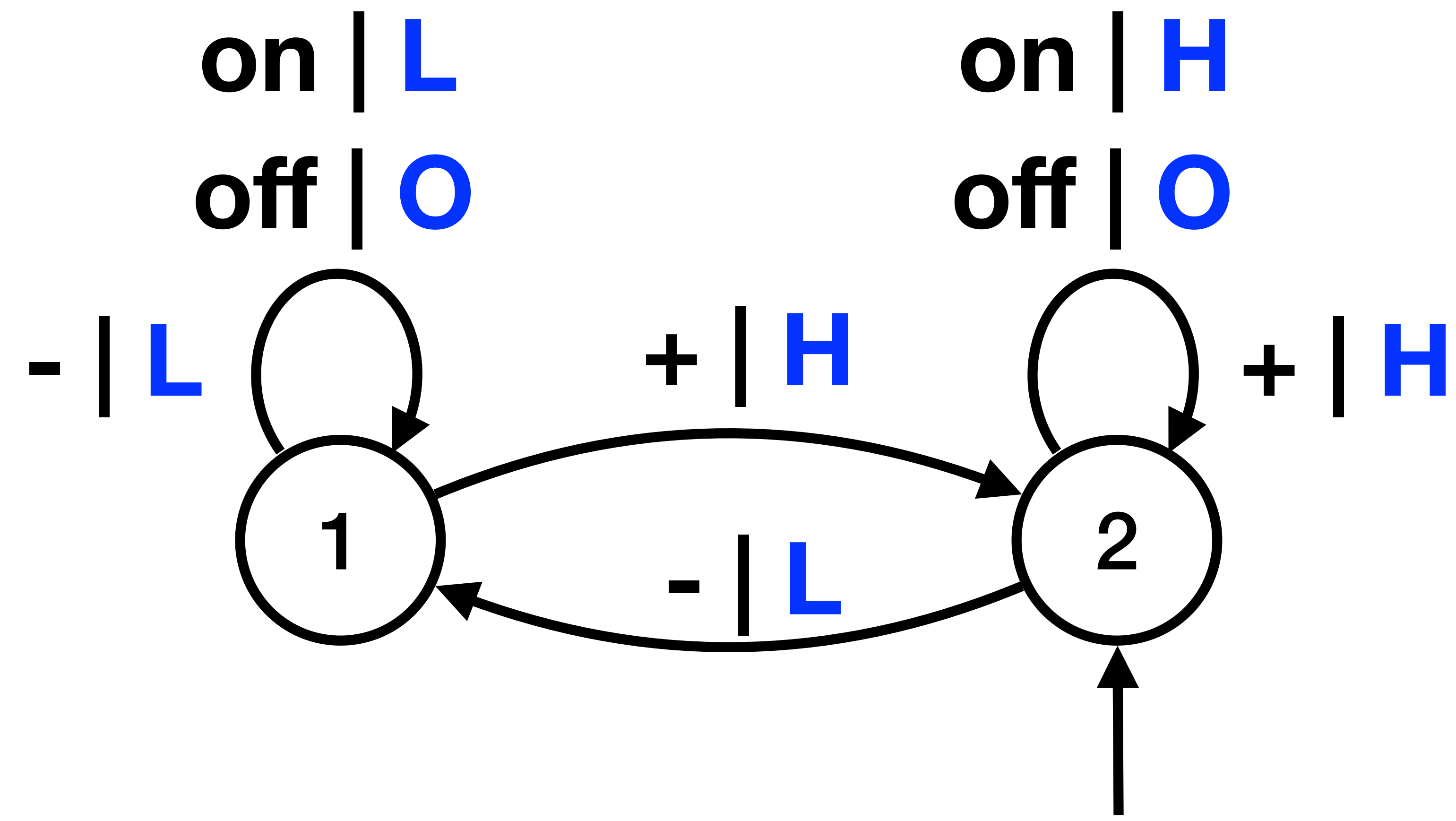
- ⏻ - ⏻ + ⏻ - + ⏻ ⏻ + - - ⏻

O H L O O L L H O H H L L O





Permutation



Reset

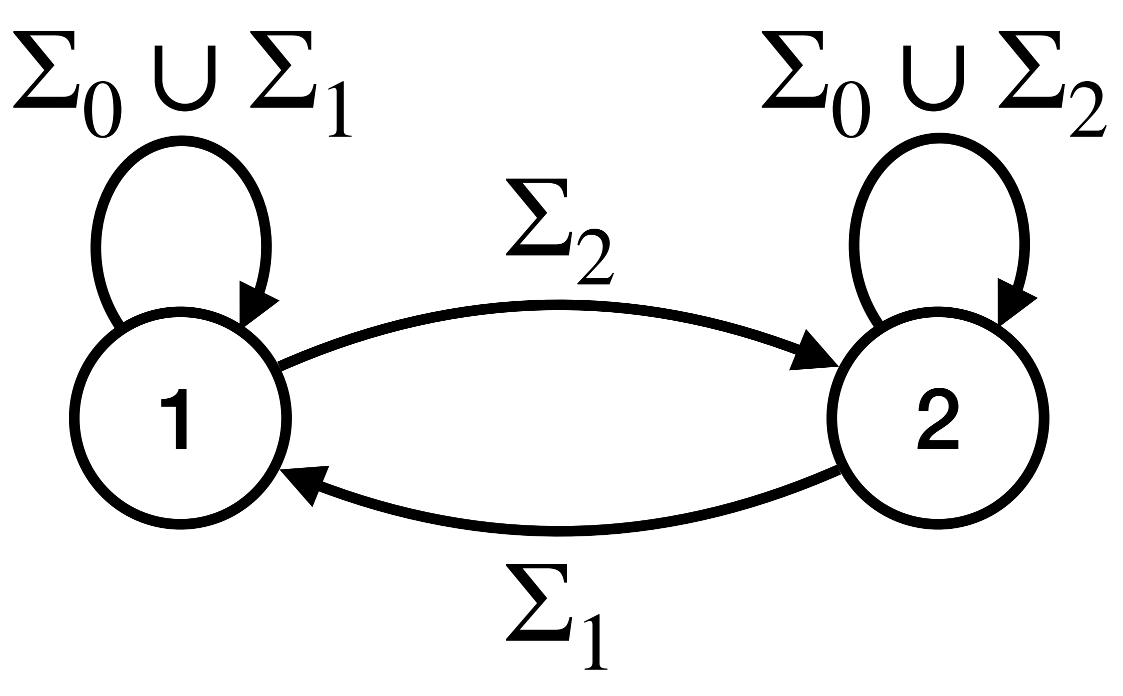
Krohn-Rhodes

Theorem

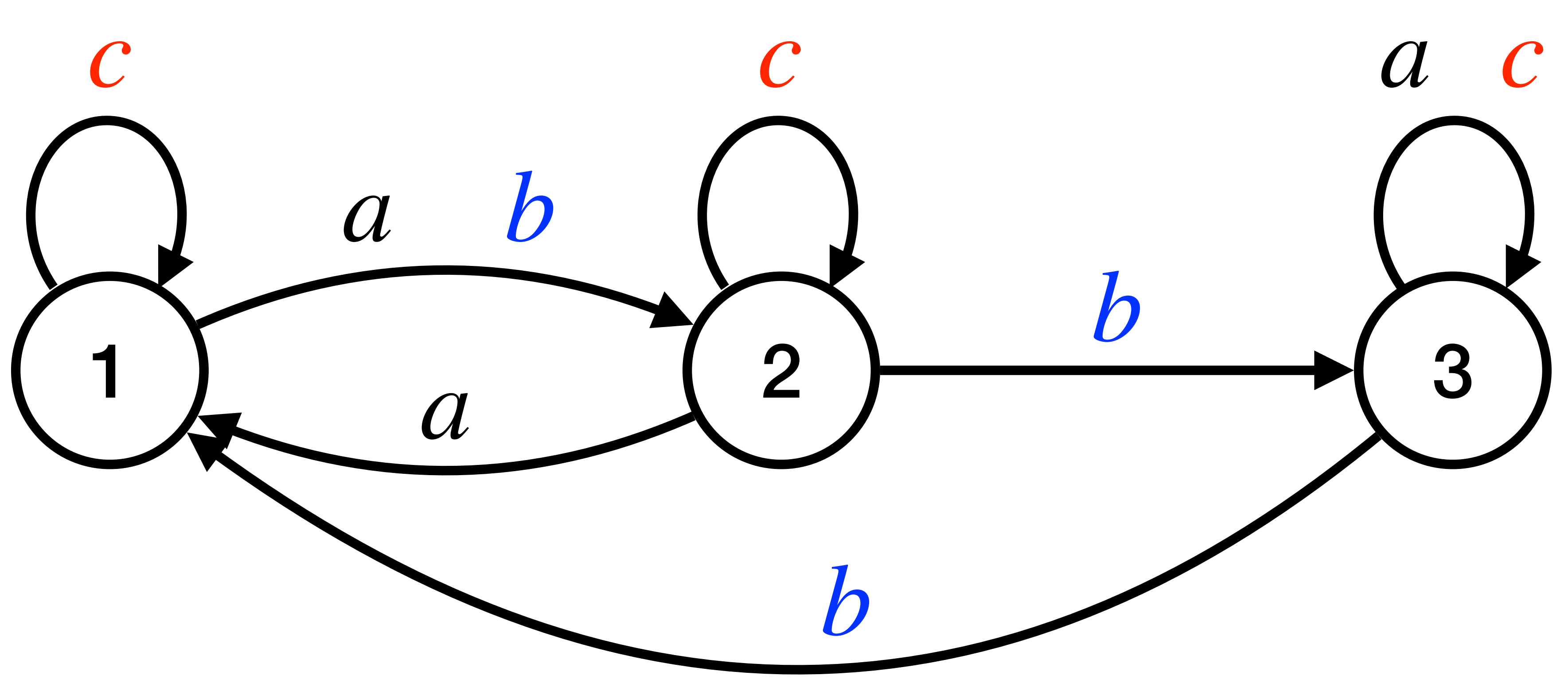
- Any regular language can be accepted by a cascade product of reset or permutation automata.
- Any aperiodic language can be accepted by a cascade product of reset automata.

Reset or Permutation automata:

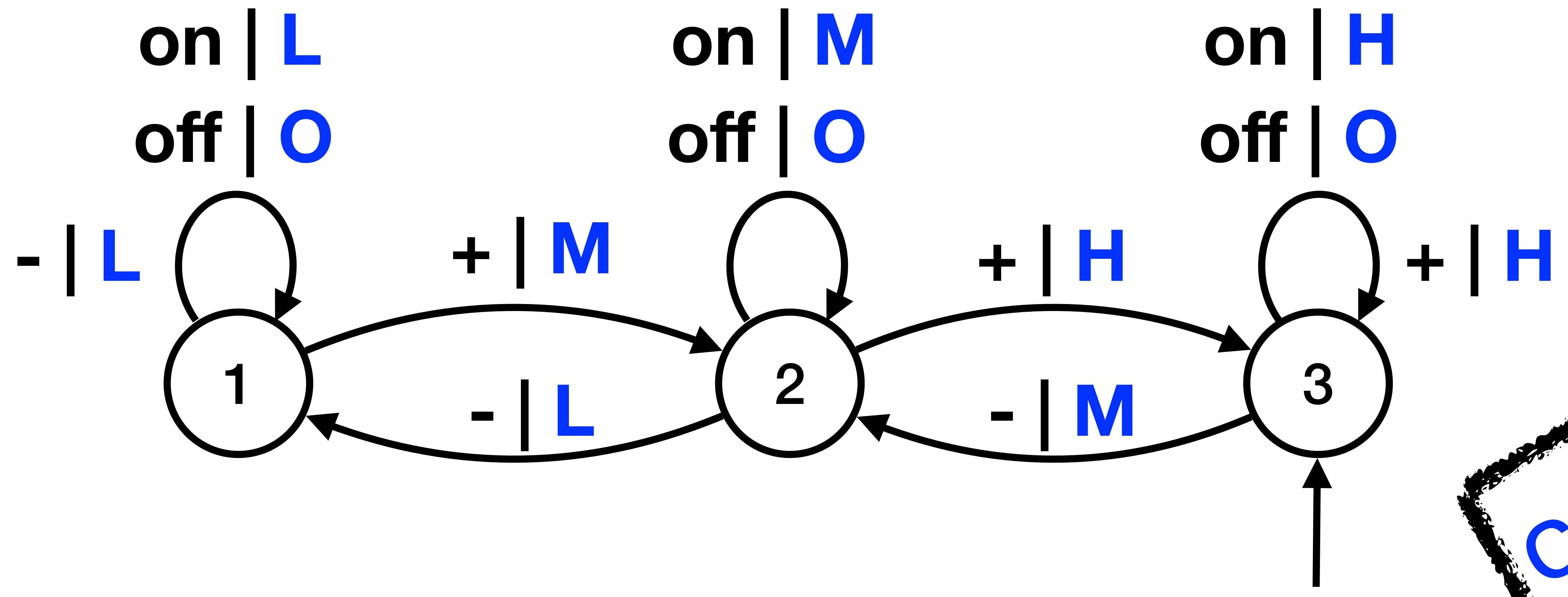
Reset - $\mathcal{U}_2$



Permutation



KR proof for aperiodic



cascade product of reset transducers

- If we ignore on and off

State_1 = $\{ + , - \}^* - - (+ -)^*$

- With $A = \{\text{on}, \text{off}, + , - \}$ and $B = \{\text{on}, \text{off}\}$

State_1 = $A^* - B^* - B^* (+ \; B^* - B^*)^*$

KR proof for aperiodic

Theorem (Kamp)

Aperiodic = Past Temporal Logic

- If we ignore on and off

$$\text{State\_1} = \{ +, - \}^* - - (+ -)^*$$

$$- \wedge \Big((+ \to \mathsf{Y} -) \ \mathsf{S} \ (- \wedge \mathsf{Y} -) \Big)$$

KR proof for aperiodic

Theorem (Kamp)

Aperiodic = Past Temporal Logic

Each PastLTL formula φ defines a boolean labelling function

$$\theta_\varphi : \Sigma^* \to \{0,1\}$$

Each position is labelled with the truth value of φ at this position.

Example $\varphi = \mathsf{Y}\, a$

a b b b a a b a b b a
0 1 0 0 0 1 1 0 1 0 0

Example $\varphi = a \;\mathsf{S}\; b$

a b b a a c c b a a c
0 1 1 1 1 0 0 1 1 1 0

KR proof for aperiodic

Theorem (Kamp)

Aperiodic = Past Temporal Logic

Each PastLTL formula φ defines a boolean labelling function

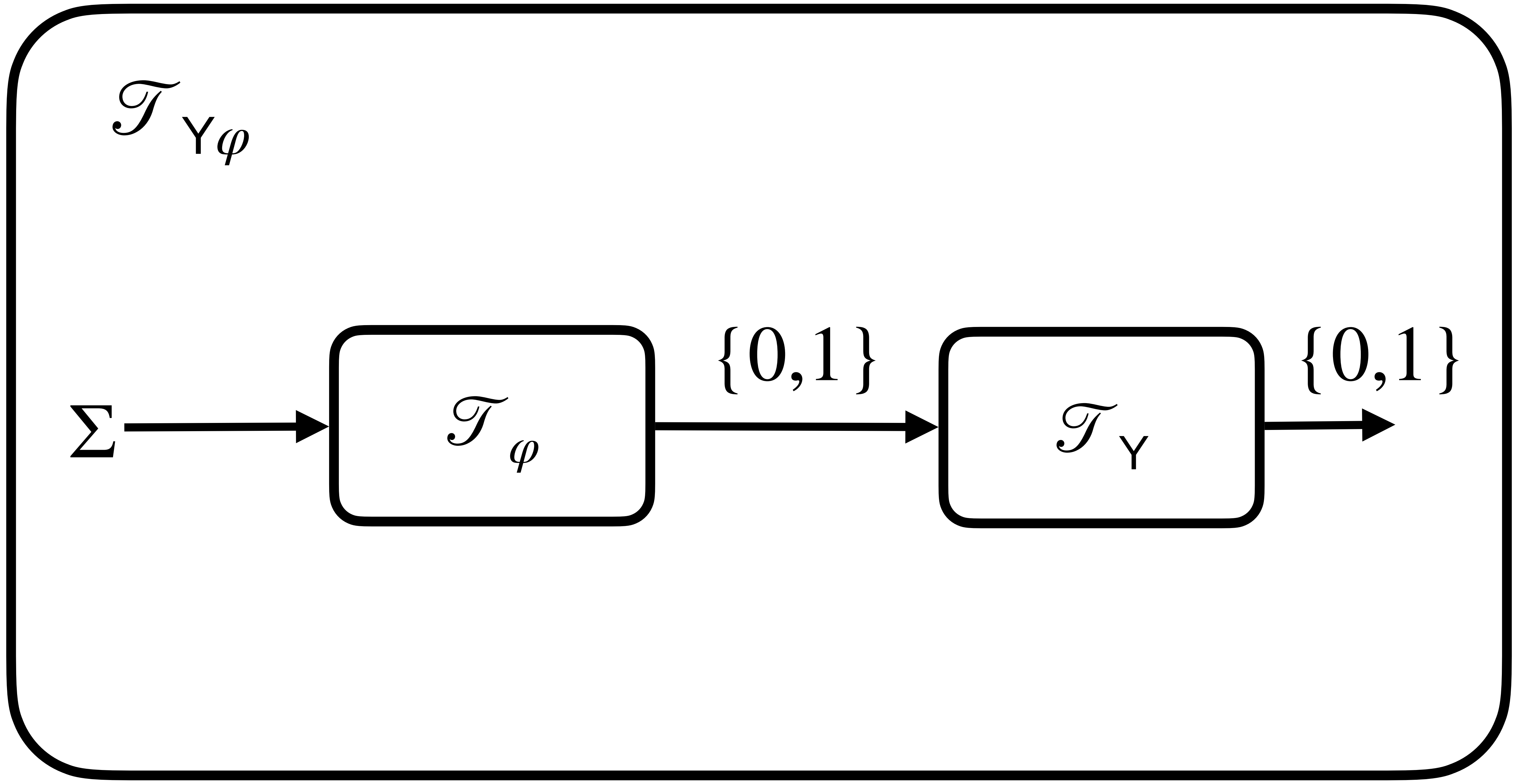
$$\theta_\varphi : \Sigma^* \to \{0,1\}$$

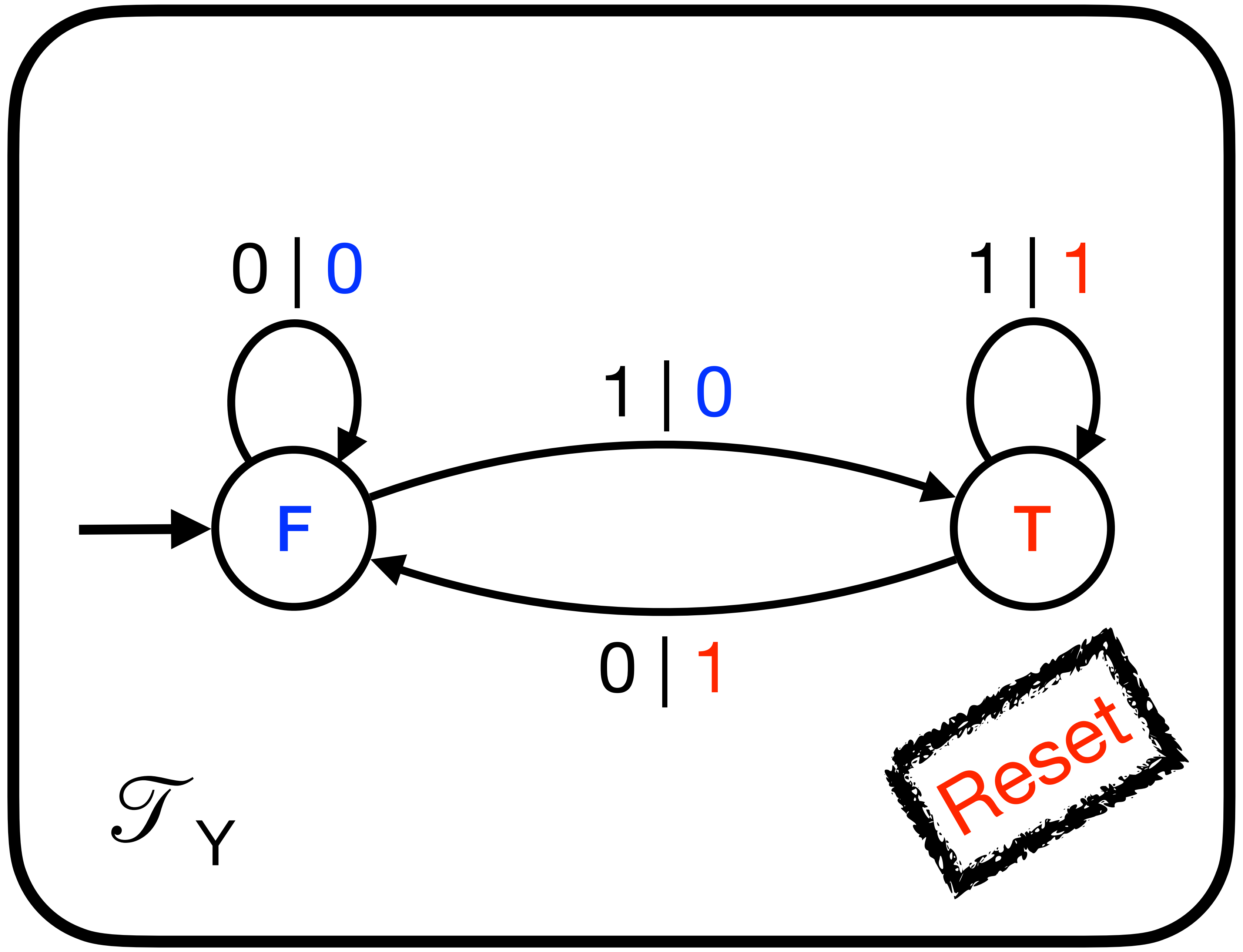
Each position is labelled with the truth value of φ at this position.

Given a PastLTL formula φ, we will implement θ_φ with a transducer $\mathcal{T}_\varphi$ constructed inductively as a cascade product of reset transducers.

PastLTL: boolean connectives, Yesterday and Since

KR proof for aperiodic





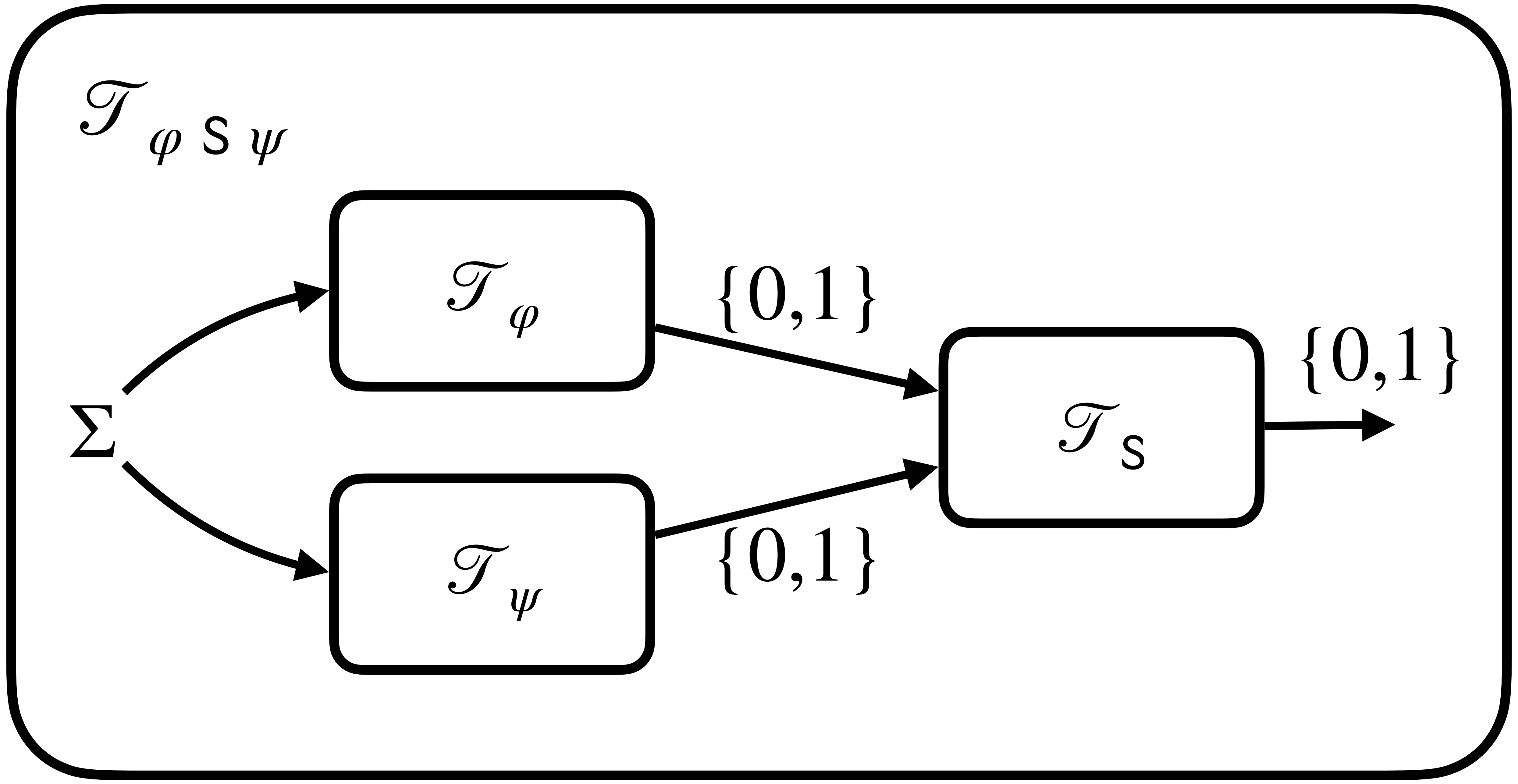
Given a PastLTL formula φ, we will implement θ_φ with a transducer $\mathscr{T}_\varphi$ constructed inductively as a cascade product of reset transducers.

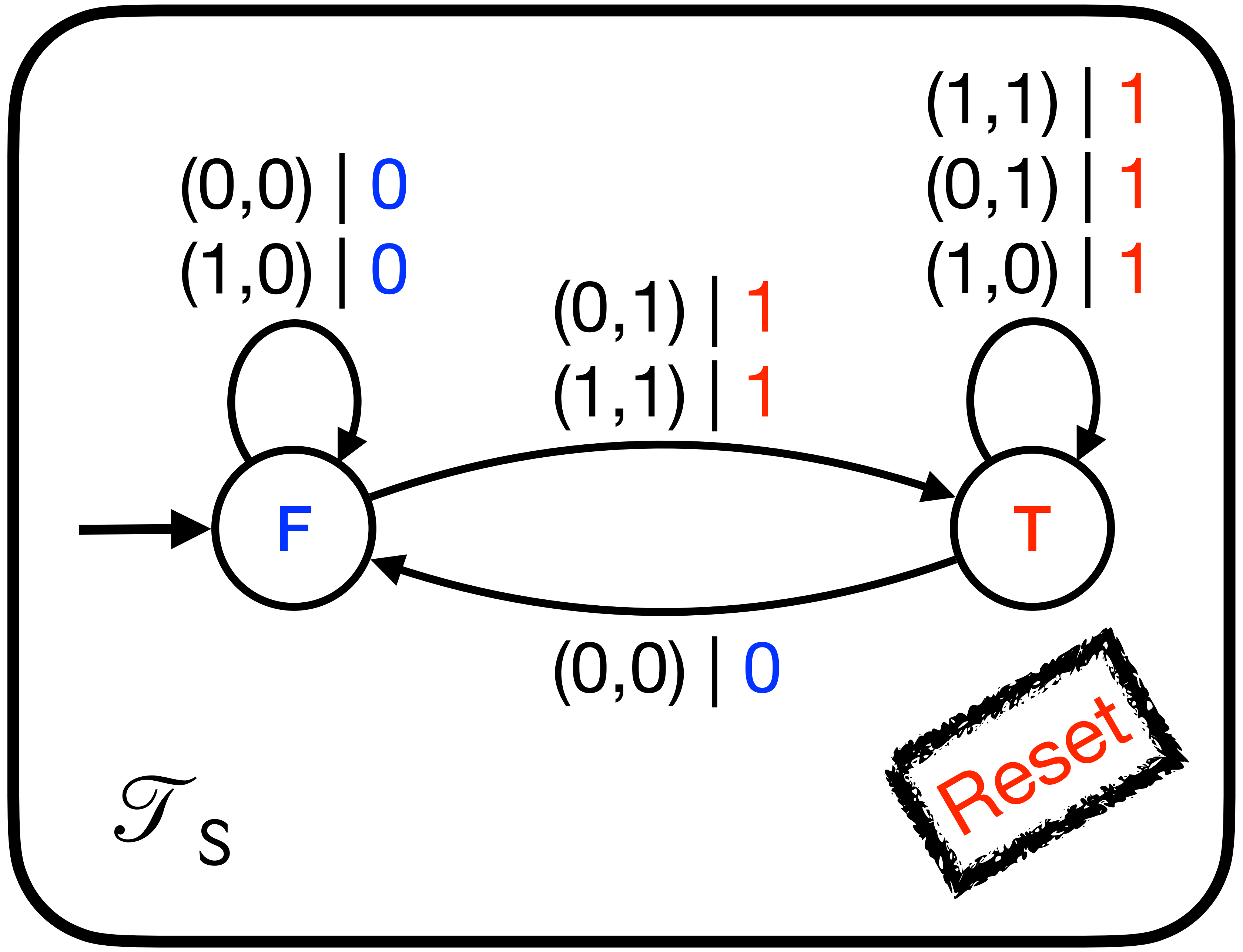
PastLTL: boolean connectives, Yesterday and Since

Example $\varphi = \mathsf{Y}\ a$

a b b a a c c b a a c
0 1 0 0 1 1 0 0 0 1 1

KR proof for aperiodic





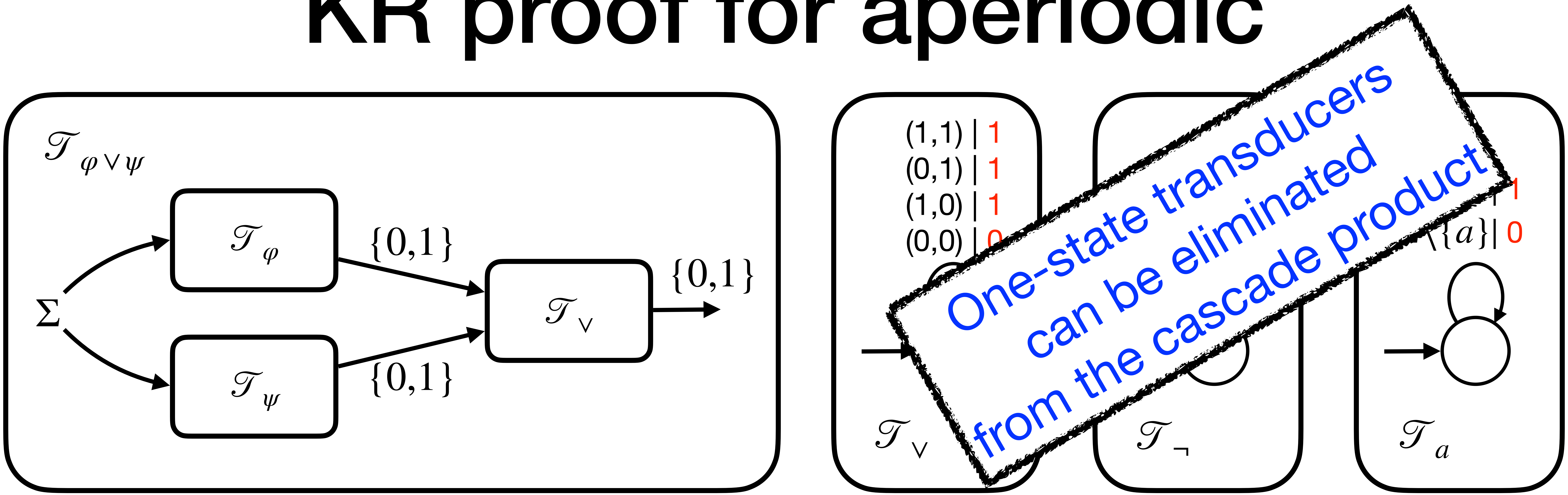
Given a PastLTL formula φ, we will implement θ_φ with a transducer $\mathscr{T}_\varphi$ constructed inductively as a cascade product of reset transducers.

PastLTL: boolean connectives, Yesterday and Since

Example $\varphi = a \text{ S } b$

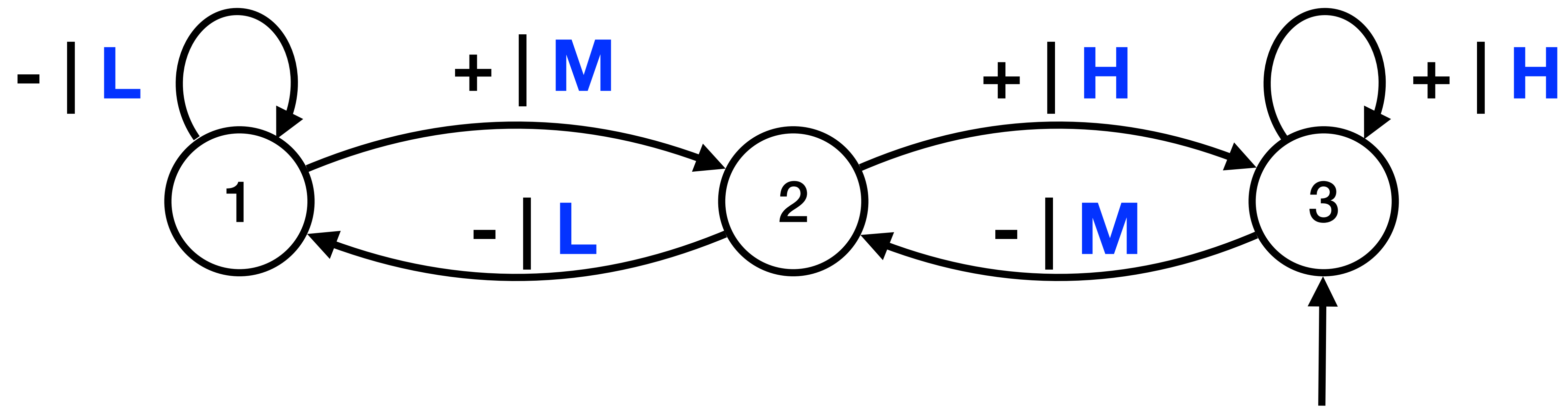
a b b a a c c b a a c
0 1 1 1 1 0 0 1 1 1 0

KR proof for aperiodic

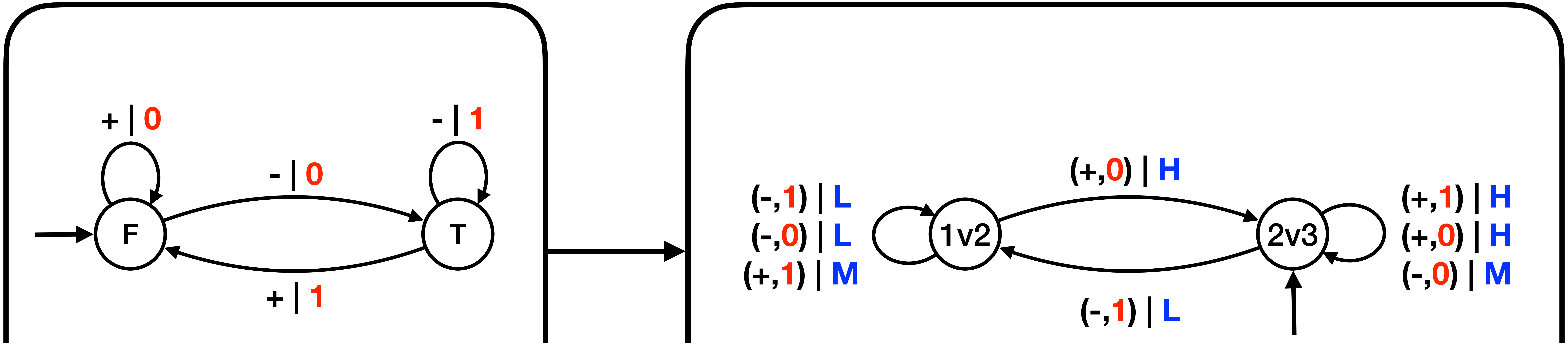


Given a PastLTL formula φ, we will implement θ_φ with a transducer $\mathscr{T}_\varphi$ constructed inductively as a cascade product of reset transducers.

PastLTL: boolean connectives, Yesterday and Since

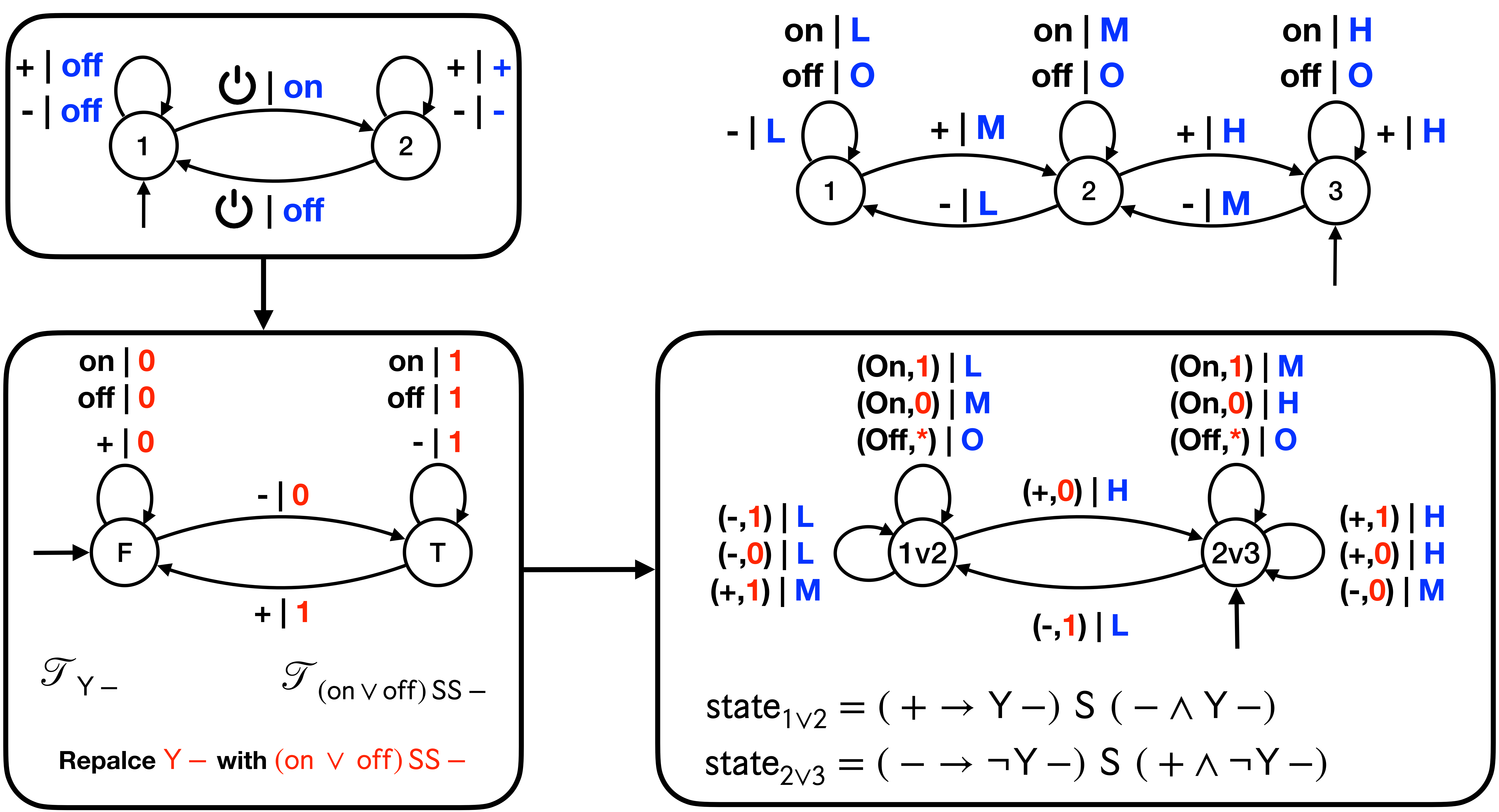


Neither Reset nor Permutation



$$\text{state}_{1\vee 2} = (+ \rightarrow Y-)\ S\ (- \wedge Y-)$$

$$\text{state}_{2\vee 3} = (- \rightarrow \neg Y-)\ S\ (+ \wedge \neg Y-)$$


+ | off
- | off
⏻ | on
⏻ | off
+ | +
- | -
1
2
on | L
off | O
on | M
off | O
on | H
off | O
- | L
+ | M
- | L
+ | H
- | M
+ | H
3
on | 0
off | 0
+ | 0
on | 1
off | 1
- | 1
- | 0
+ | 1
F
T
𝒯 Y −
𝒯 (on ∨ off) SS −
Repalce Y − with (on ∨ off) SS −
(On,1) | L
(On,0) | M
(Off,*) | O
(On,1) | M
(On,0) | H
(Off,*) | O
(+,0) | H
(-,1) | L
(-,0) | L
(+,1) | M
(+,1) | H
(+,0) | H
(-,0) | M
(-,1) | L
1v2
2v3
state1∨2 = (+ → Y −) S (− ∧ Y −)
state2∨3 = (− → ¬Y −) S (+ ∧ ¬Y −)

Outline

- Labelling functions, sequential transducers and cascade product
- Krohn-Rhodes theorem for aperiodic/regular word languages
- Model of concurrency: Mazurkiewicz traces and asynchronous Zielonka automata
- Asynchronous labelling functions, transducers and cascade product
- Propositional dynamic logic for traces
- Conclusion

Mazurkiewicz Traces

Architecture

$\mathscr{P} = \{1,2,3\}$

$\Sigma = \{a, b, c, d, e\}$

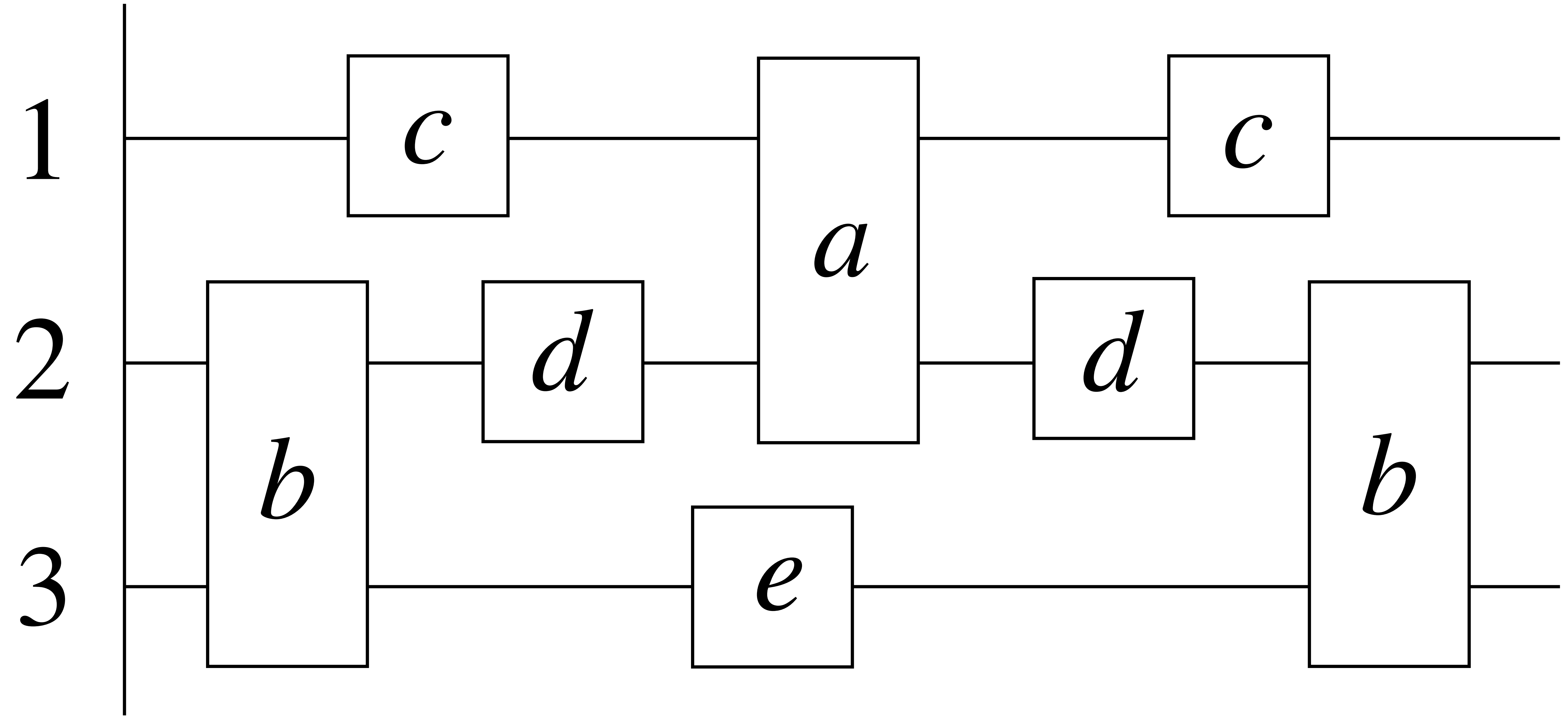
$\mathrm{loc}(a) = \{1,2\}$

$\mathrm{loc}(b) = \{2,3\}$

$\mathrm{loc}(c) = \{1\}$

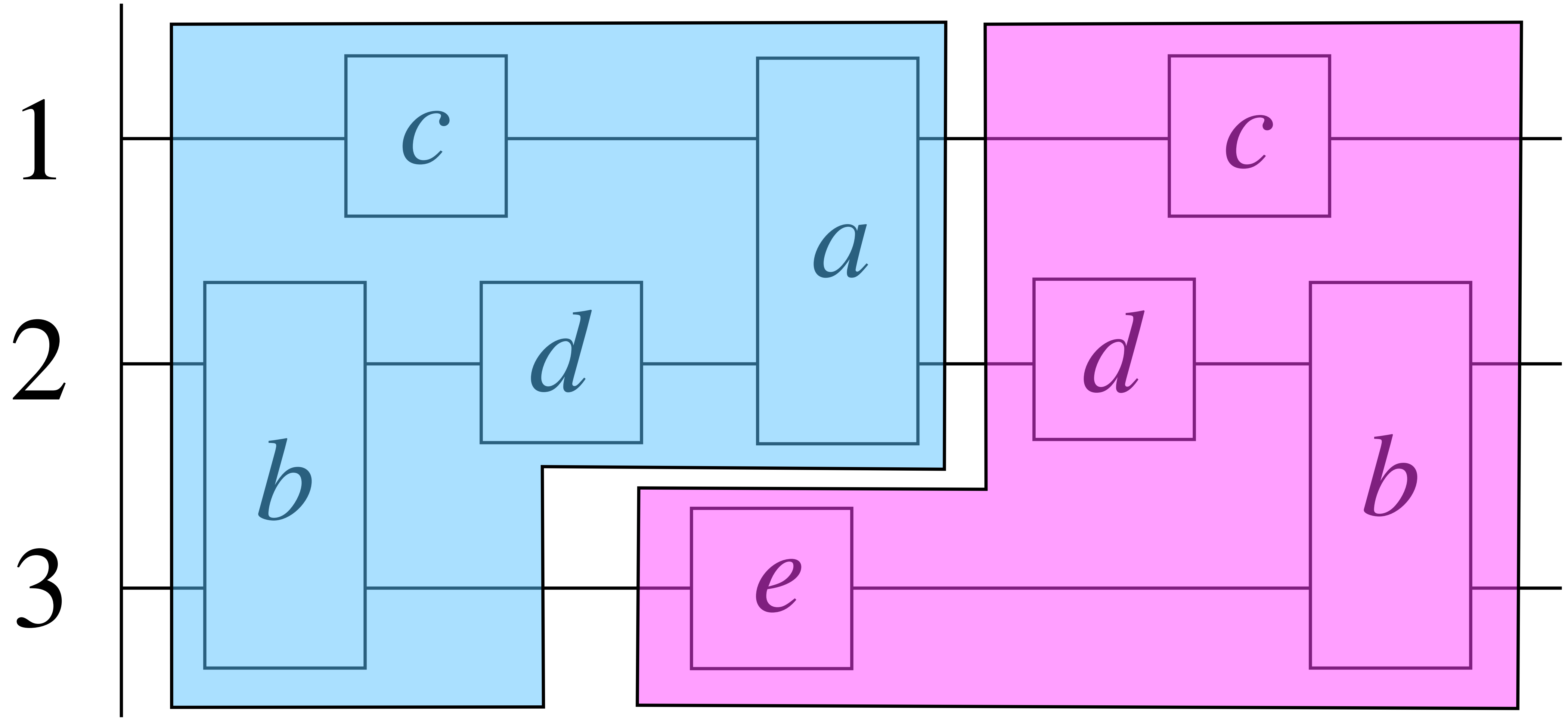
$\mathrm{loc}(d) = \{2\}$

$\mathrm{loc}(e) = \{3\}$



- *set of traces denoted* $Tr(\Sigma, \mathscr{P}, \mathrm{loc})$ *or simply* $Tr(\Sigma)$
- *Trace Language:* $L \subseteq Tr(\Sigma)$

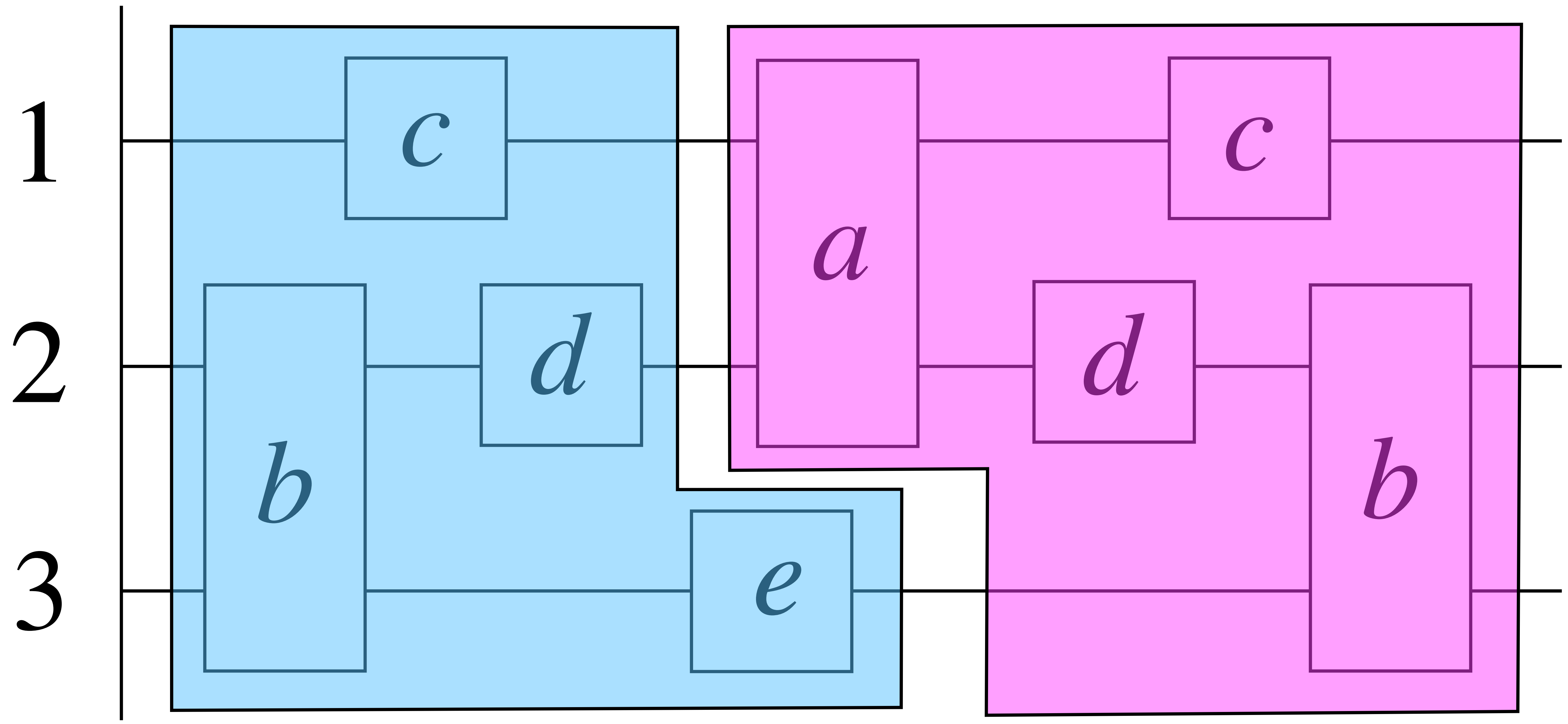
Concatenation, Independence, Commutation, Monoid



- $x \; I \; y$ iff $\mathrm{loc}(x) \cap \mathrm{loc}(y) = \emptyset$
- $a \; I \; e$
- $a \cdot e = e \cdot a$

- *$Tr(\Sigma)$ with trace concatenation is a monoid*
- *Free partially commutative monoid*

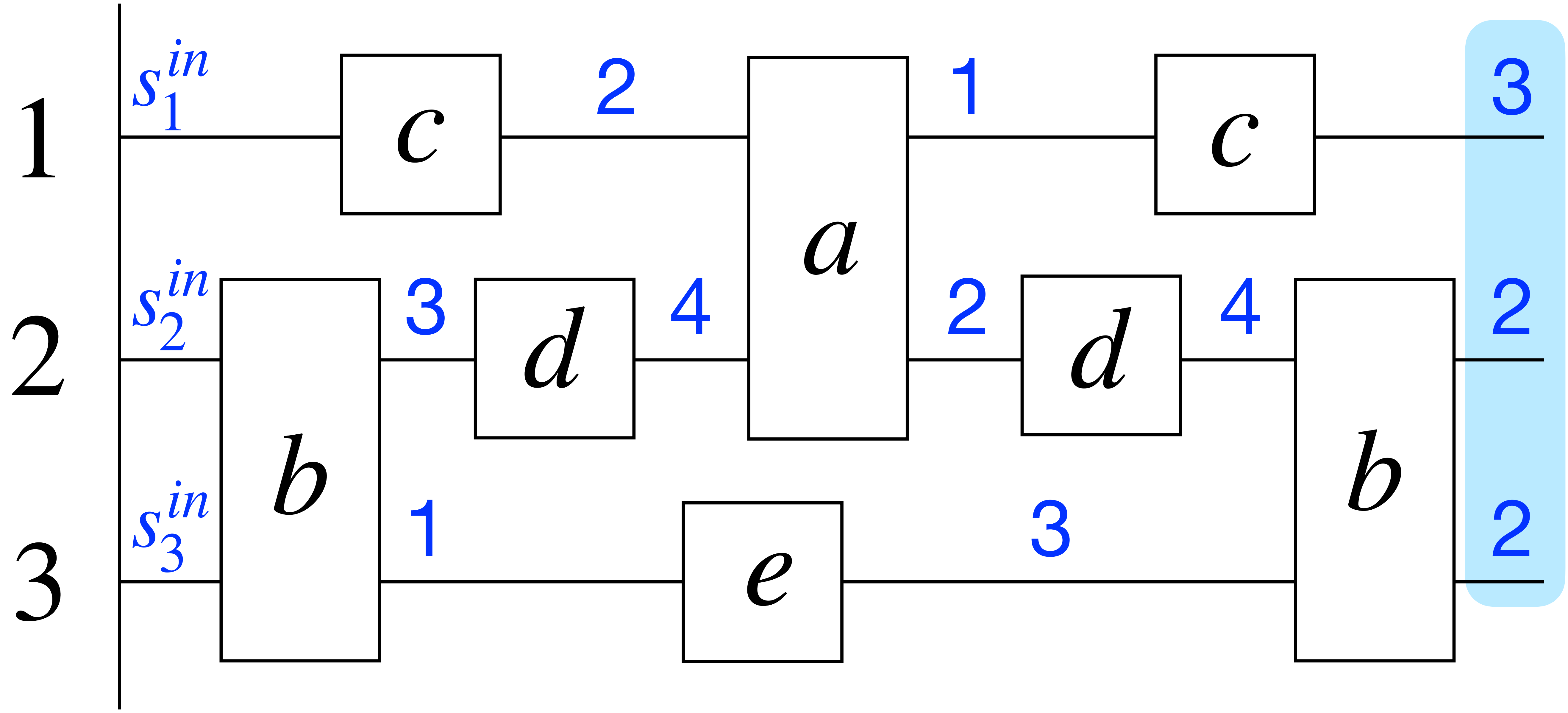
Concatenation, Independence, Commutation, Monoid



- $x \; I \; y$ iff $\mathrm{loc}(x) \cap \mathrm{loc}(y) = \varnothing$
- $a \; I \; e$
- $a \cdot e = e \cdot a$

- *$Tr(\Sigma)$ with trace concatenation is a monoid*
- *Free partially commutative monoid*

Asynchronous automata (Zielonka)



$\mathcal{A} = \left(\{S_i\}_{i\in\mathcal{P}}, \{\delta_a\}_{a\in\Sigma}, s^{in}, F\right)$

- S_i local states for process i
- δ_a transition function for action a
- $s^{in} = (s_1^{in}, s_2^{in}, s_3^{in})$ global initial state
- F global accepting states

$\delta_c : S_1 \to S_1$

$\delta_b : S_2 \times S_3 \to S_2 \times S_3$

$\delta_d : S_2 \to S_2$

$\delta_a : S_1 \times S_2 \to S_1 \times S_2$

$\delta_e : S_3 \to S_3$

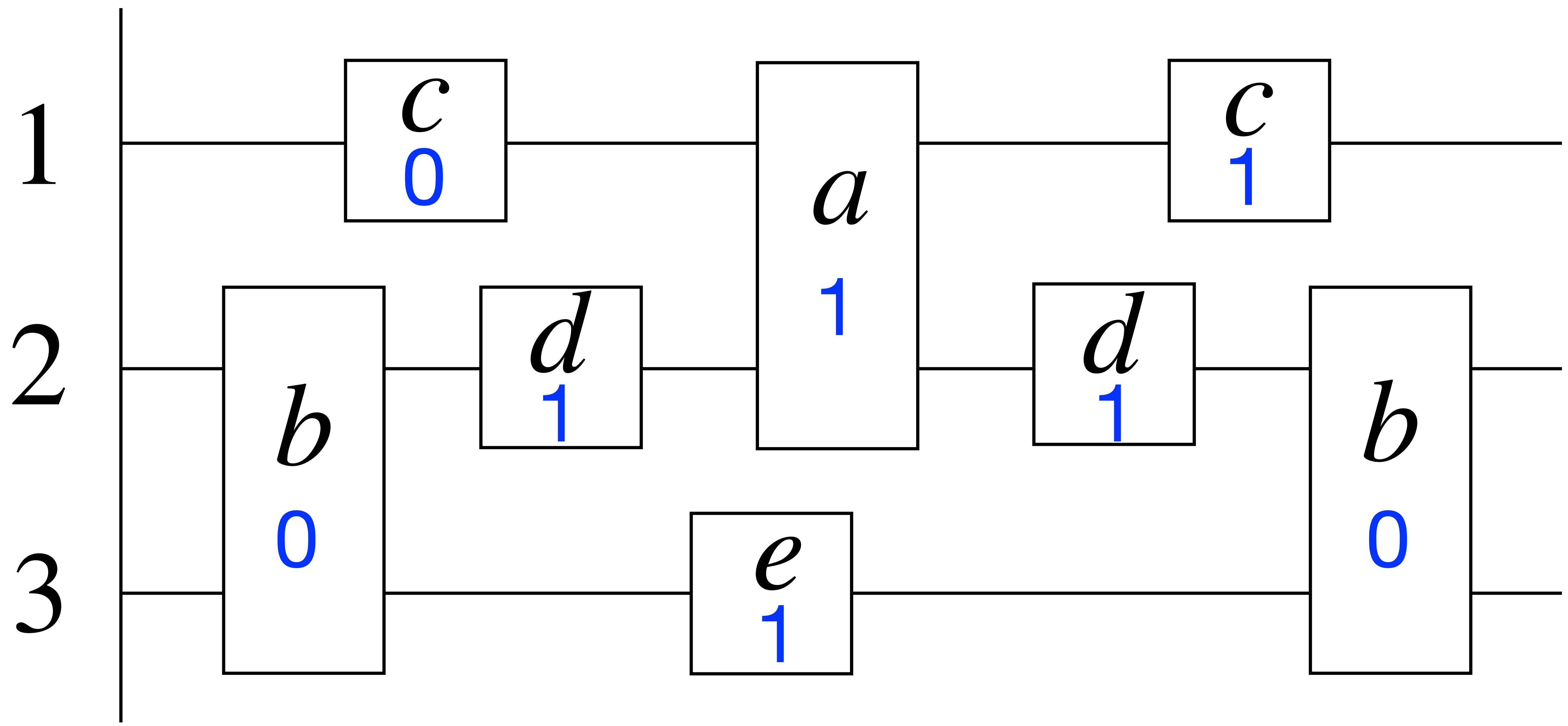
Theorem (Zielonka, 1987)

Asynchronous Automata = Regular Trace Languages

Outline

- Labelling functions, sequential transducers and cascade product
- Krohn-Rhodes theorem for aperiodic/regular word languages
- Model of concurrency: Mazurkiewicz traces and asynchronous Zielonka automata
- Asynchronous labelling functions, transducers and cascade product
- Propositional dynamic logic for traces
- Conclusion

Labelling function



$\theta : Tr(\Sigma) \to Tr(\Sigma \times \Gamma)$

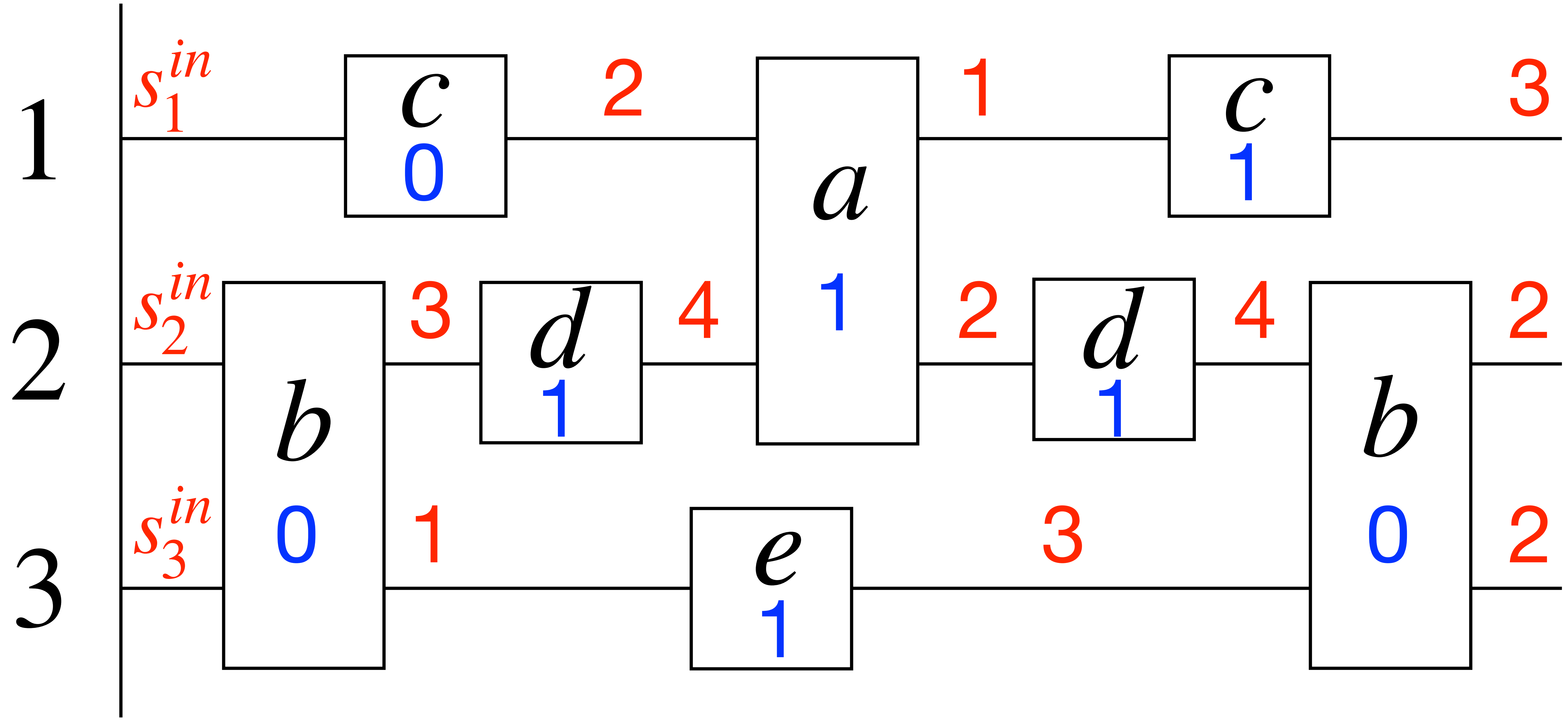
$\Gamma = \{0,1\}$

$Y_3 \leq Y_2$: In the strict past,

the last event on process 3 is below

the last event on process 2

Asynchronous Labelling function



$\theta: Tr(\Sigma) \to Tr(\Sigma \times \Gamma)$

$\Gamma = \{0,1\}$ and $Y_3 \leq Y_2$

In the strict past,
the last event on process 3 is below
the last event on process 2

Asynchronous (letter-to-letter) transducer $\mathcal{T} = \left(\mathcal{A}, \{\mu_a\}_{a\in\Sigma}\right)$

- $\mathcal{A} = \left(\{S_i\}_{i\in\mathcal{P}}, \{\delta_a\}_{a\in\Sigma}, s^{in}\right)$
- $\mu_a: S_a \to \Gamma$

$\mu_c: S_1 \to \Gamma$

$\mu_b: S_2 \times S_3 \to \Gamma$

$\mu_d: S_2 \to \Gamma$

$\mu_a: S_1 \times S_2 \to \Gamma$

$\mu_e: S_3 \to \Gamma$

Composition - Cascade product

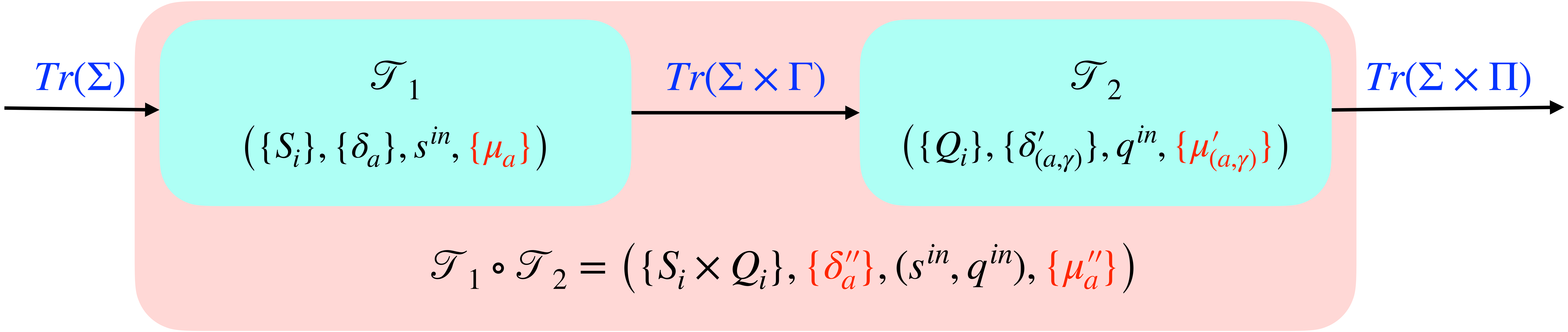
- Composition of labelling functions

$$Tr(\Sigma) \xrightarrow{\theta_1} Tr(\Sigma \times \Gamma) \xrightarrow{\theta_2} Tr(\Sigma \times \Pi)$$

$$\boxed{\delta''_a(s,q) = \left(\delta_a(s), \delta'_{(a,\mu_a(s))}(q)\right)}$$

$$\boxed{\mu''_a(s,q) = \mu'_{(a,\mu_a(s))}(q)}$$

- Cascade product of asynchronous (letter-to-letter) transducers



Cascade Decomposition

Main Theorem 1

Any asynchronous labelling function can be realised by a cascade product of local asynchronous transducers:

$$\mathscr{T}_1 \circ \mathscr{T}_2 \circ \cdots \circ \mathscr{T}_n$$

$\mathscr{T} = \left(\{S_i\}_{i\in\mathscr{P}}, \{\delta_a\}, s^{in}, \{\mu_a\}\right)$ is k-local if only k is non-trivial $(i = k \vee |S_i| = 1)$

Corollary: Zielonka's theorem

Asynchronous Automata = Regular Trace Languages

Cascade Decomposition

Main Theorem 1

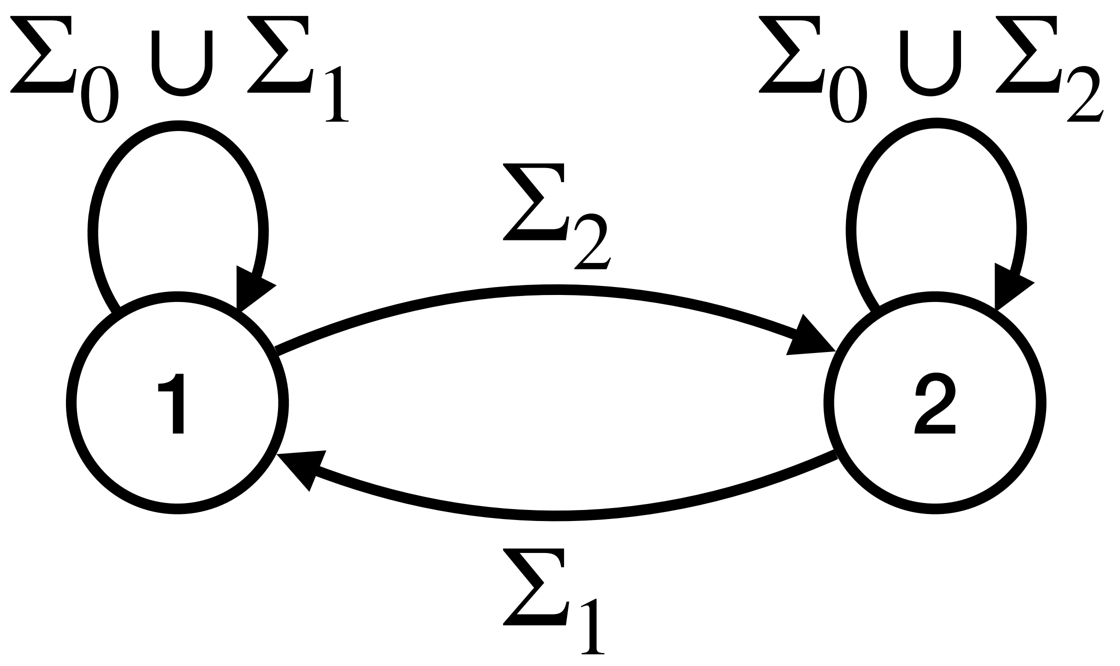
Any asynchronous labelling function can be realised by a cascade product of local asynchronous transducers:

$$\mathcal{T}_1 \circ \mathcal{T}_2 \circ \cdots \circ \mathcal{T}_n$$

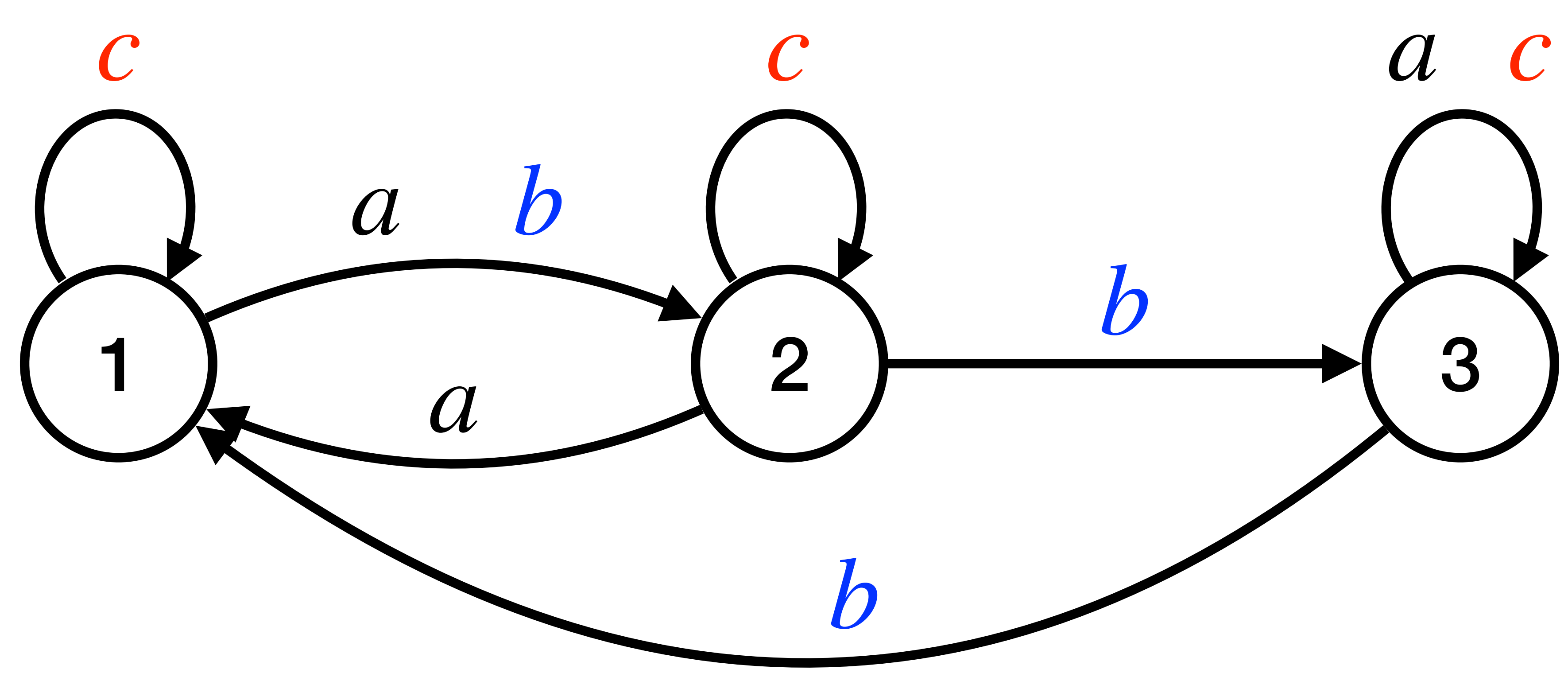
Bonus: Using Krohn-Rhodes theorem

Each local asynchronous transducer $\mathcal{T}$ can be chosen to be (on its non-trivial component)

Reset - $\mathcal{U}_2$



Permutation



Cascade Decomposition

Main Theorem 1

Any asynchronous labelling function can be realised by a cascade product of local asynchronous transducers:

$$\mathcal{T}_1 \circ \mathcal{T}_2 \circ \cdots \circ \mathcal{T}_n$$

Proof sketch:

- Design a local and past propositional dynamic logic (locPastPDL)
  - State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi$
  - Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$
- Prove that event formulas are expressively complete wrt regular past predicates (difficult)
- For each event formula φ, construct by structural induction a cascade product of local asynchronous transducers computing its labelling function θ_φ (easier)

Outline

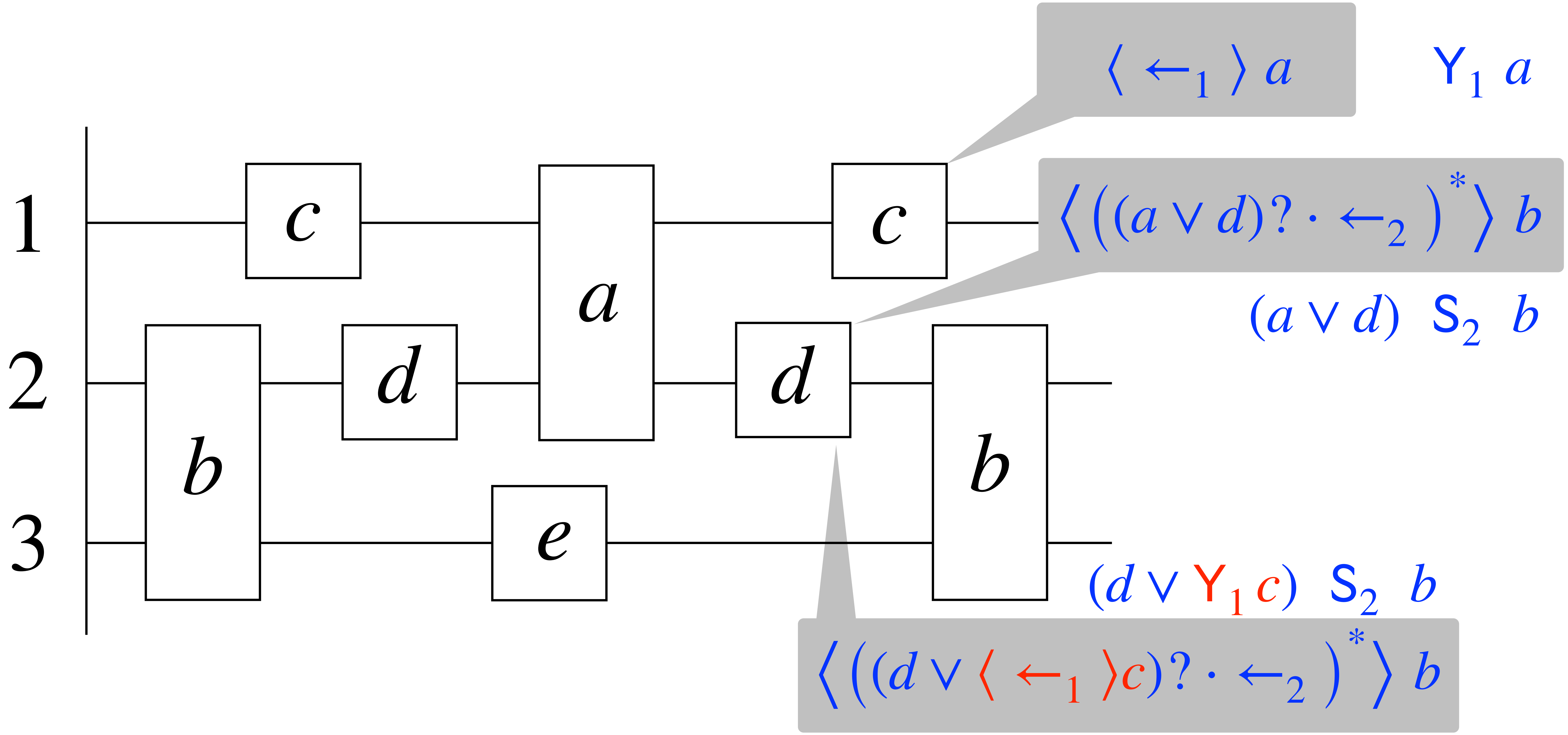
- Labelling functions, sequential transducers and cascade product
- Krohn-Rhodes theorem for aperiodic/regular word languages
- Model of concurrency: Mazurkiewicz traces and asynchronous Zielonka automata
- Asynchronous labelling functions, transducers and cascade product
- Propositional dynamic logic for traces
- Conclusion

Propositional Dynamic Logic

- First introduced to reason about programs (Fischer, Ladner 1979)
  - State formulas $\varphi ::= p \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi$
  - Program expressions $\pi ::= \varphi? \mid x := e \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$
    - If φ then π_1 else π_2 $(\varphi? \cdot \pi_1) + (\neg\varphi? \cdot \pi_2)$
    - While φ do π_1 od; π_2 $(\varphi? \cdot \pi_1)^* \cdot \neg\varphi? \cdot \pi_2$
- Interpretation over words: Linear Dynamic Logic (Giacomo, Vardi 2013)

Regular word languages = MSO definable = LDL definable

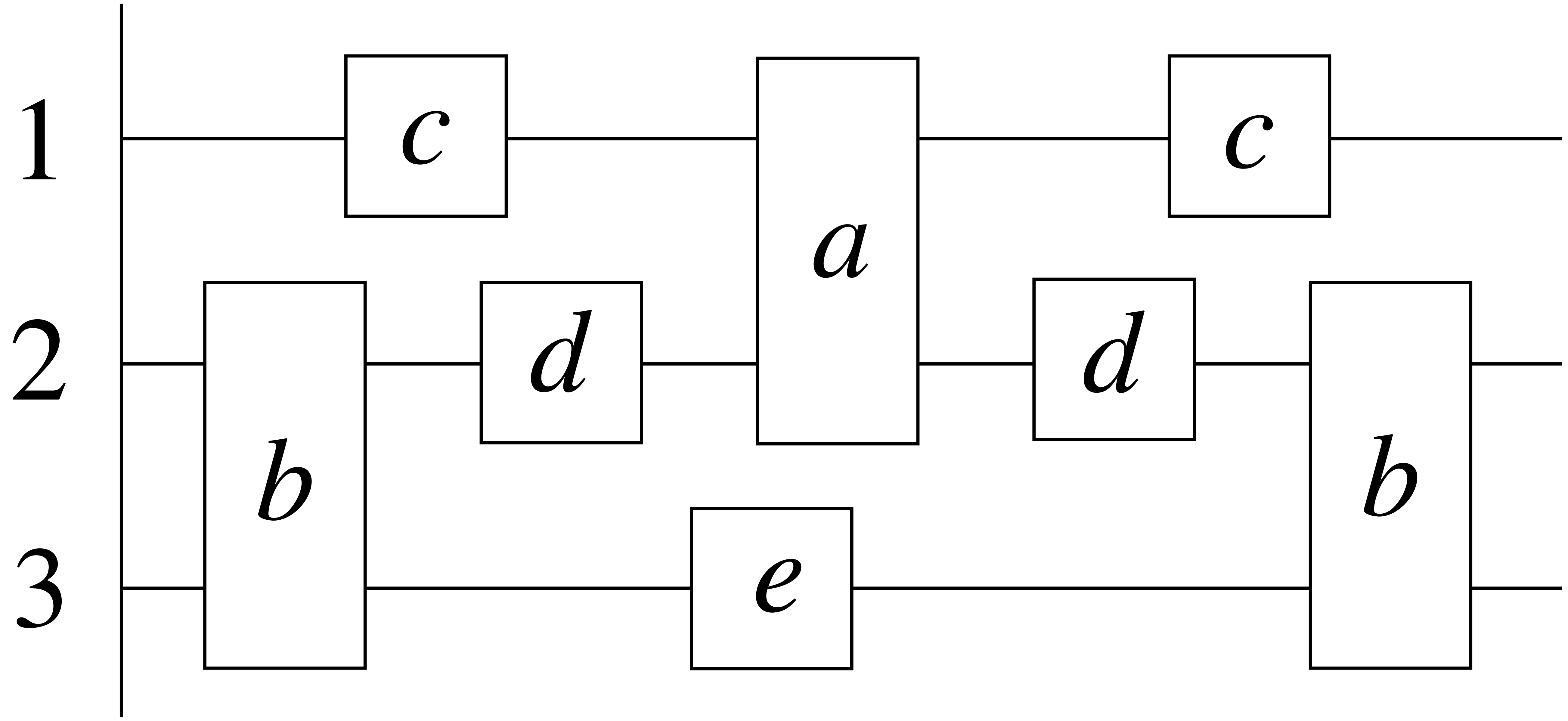
Past PDL for Traces



- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg \varphi \mid \langle \pi \rangle \varphi$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces

$$T \models \mathsf{EM}_1\, c \wedge \mathsf{EM}_3 \left(b \wedge \langle \leftarrow_2 \cdot \left((a \vee d)? \cdot \leftarrow_2 \right)^* \rangle b \right)$$



- Sentences / Trace formulas $\quad \Phi ::= \mathsf{EM}_i\ \varphi \mid \Phi \vee \Phi \mid \neg \Phi$
- State/Event formulas $\quad \varphi ::= a \mid \varphi \vee \varphi \mid \neg \varphi \mid \langle \pi \rangle \varphi$
- Program/Path expressions $\quad \pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

(Local) Past-PDL for Traces

Main Theorem 2

- Sentences locPastPDL = PastPDL = Regular Trace Languages
- Event formulas locPastPDL = PastPDL = Regular Past Predicates

PastPDL

$\cap|$ easy

MSO

$\cap|$ known

Morphisms

$\cap|$ difficult

locPastPDL

- Let $\eta\colon Tr(\Sigma) \to M$ be a morphism to a finite monoid
- For each $m \in M$, we construct a locPastPDL event formula $\varphi^{(m)}$ such that, if T is a prime trace (i.e., having a single maximal event),

$$\eta(T) = m \text{ if and only if } T, \max(T) \models \varphi^{(m)}$$

Induction on the number of processes

(Local) Past-PDL for Traces

Main Theorem 2

- Sentences locPastPDL = PastPDL = Regular Trace Languages
- Event formulas locPastPDL = PastPDL = Regular Past Predicates

PastPDL

$\subseteq$ easy

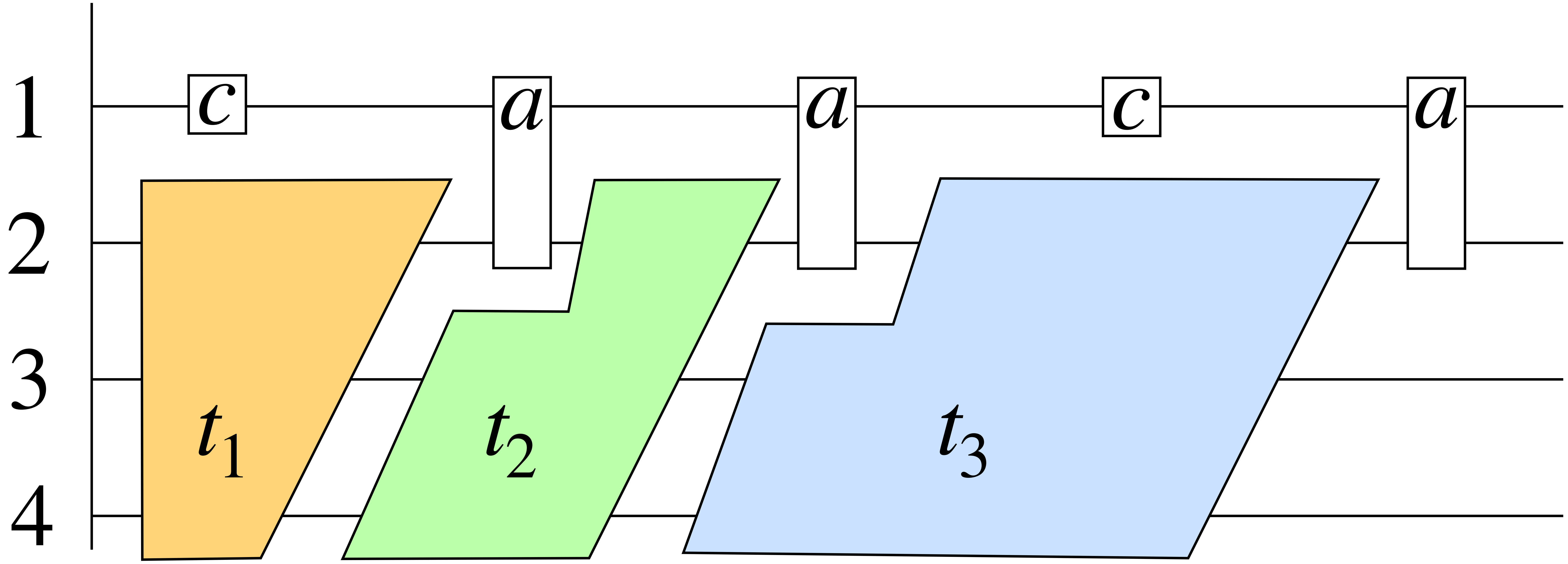
MSO

$\subseteq$ known

Morphisms

$\subseteq$ difficult

locPastPDL



(Local) Past-PDL for Traces

Main Theorem 2

- Sentences locPastPDL = PastPDL = Regular Trace Languages
- Event formulas locPastPDL = PastPDL = Regular Past Predicates

PastPDL

$\subseteq$ easy

MSO

$\subseteq$ known

Morphisms

$\subseteq$ difficult

locPastPDL

- Let $\eta\colon Tr(\Sigma) \to M$ be a morphism to a finite monoid
- For each $m \in M$, we construct a locPastPDL event formula $\varphi^{(m)}$ such that,

  if T is a prime trace (i.e., having a single maximal event),

  $$\eta(T) = m \text{ if and only if } T, \max(T) \models \varphi^{(m)}$$

  Induction on the number of processes
- For each $m \in M$, we construct a sentence $\Phi^{(m)}$ which defines $\eta^{-1}(m)$

  decompose an arbitrary (non prime) trace into a product of prime traces.

(Local) Past-PDL for Traces

Main Theorem 2

- Sentences locPastPDL = PastPDL = Regular Trace Languages
- Event formulas locPastPDL = PastPDL = Regular Past Predicates

PastPDL

I∩ easy

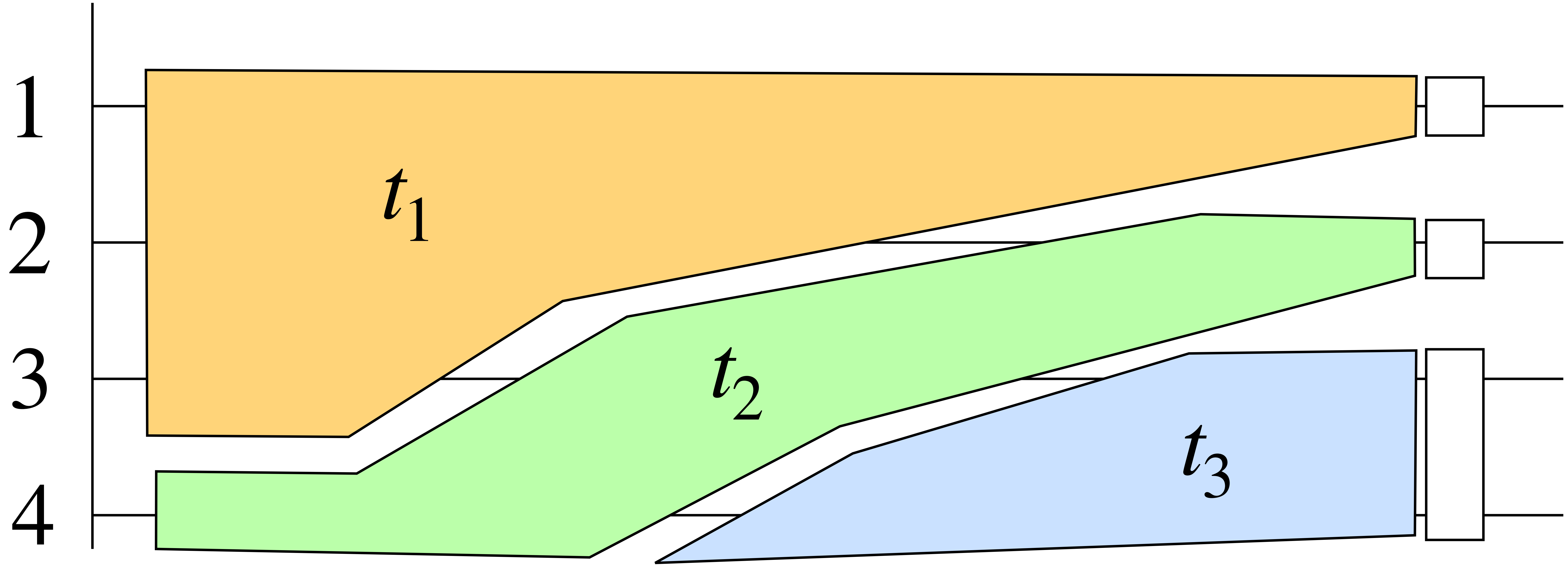
MSO

I∩ known

Morphisms

I∩ difficult

locPastPDL



(Local) Past-PDL for Traces

Main Theorem 2

- Sentences locPastPDL = PastPDL = Regular Trace Languages
- Event formulas locPastPDL = PastPDL = Regular Past Predicates

PastPDL

$\subseteq$ easy

MSO

$\subseteq$ known

Morphisms

$\subseteq$ difficult

locPastPDL

- Let $\eta \colon Tr(\Sigma) \to M$ be a morphism to a finite monoid
- For each $m \in M$, we construct a locPastPDL event fo[...]at, if T is a prime trace (i.e., having a single maxi[...]

$$\eta(T) = m \text{ if and } [...] \varphi^{(m)}$$

Induction on the number of pro[...]

- For each $m \in M$, we constr[...] $\Phi^{(m)}$ which defines $\eta^{-1}(m)$

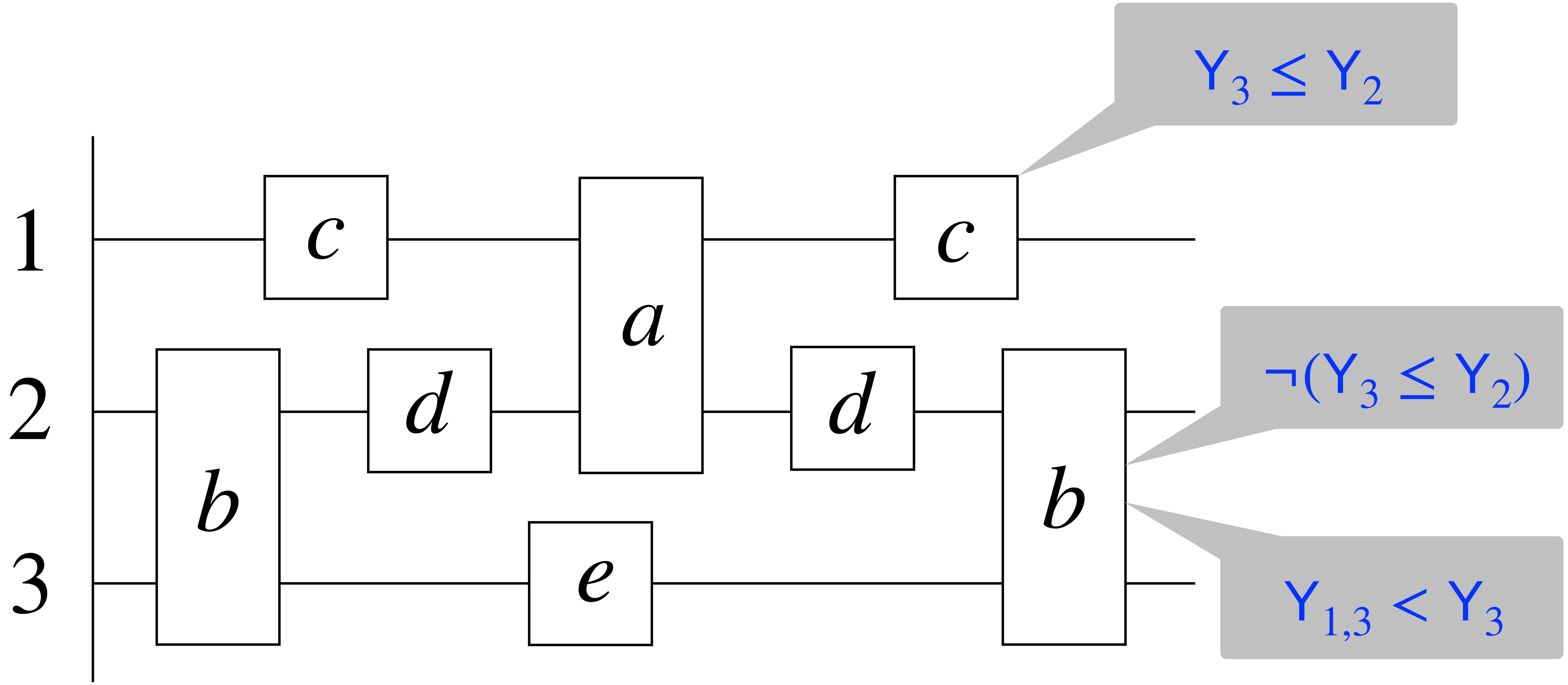
decompose an arbitrary (non p[...] trace into a product of prime traces.

Decompositions depend crucially on

$Y_i \leq Y_j$, $Y_{i,k} \leq Y_j$ and $L_i \leq L_j$, $L_{i,k} \leq L_j$

Extended locPastPDL for Traces

- $T \vDash \neg(\mathsf{L}_1 \leq \mathsf{L}_3)$
- $T \vDash \mathsf{L}_{1,2} < \mathsf{L}_3$



- Trace formulas / Sentences $\Phi ::= \mathsf{EM}_i\ \varphi \mid \Phi \vee \Phi \mid \neg\Phi \mid \mathsf{L}_i \leq \mathsf{L}_j \mid \mathsf{L}_{i,k} \leq \mathsf{L}_j$
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi \mid \mathsf{Y}_i \leq \mathsf{Y}_j \mid \mathsf{Y}_{i,k} \leq \mathsf{Y}_j$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Extended locPastPDL for Traces

Theorem (Adsul, Gastin, Sarkar, Weil — CONCUR'22)

- Extended locPastPDL is expressively complete for regular trace languages
- Any regular trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local asynchronous transducers:

$$\mathscr{G} \circ \mathscr{T}_1 \circ \mathscr{T}_2 \circ \cdots \circ \mathscr{T}_n$$

Theorem (Mukund-Sohoni 1997)

There is an asynchronous letter-to-letter transducer $\mathscr{G}$ which computes the truth values of the constants from

$$\mathscr{Y} = \{\mathsf{Y}_i \leq \mathsf{Y}_j \,, \mathsf{Y}_{i,k} \leq \mathsf{Y}_j \mid i, j, k \in \mathscr{P}\}.$$

~~Extended~~ locPastPDL for Traces

Theorem (Adsul, Kulkarni, Gastin, Weil — SUBMITTED'24)

- locPastPDL = Extended locPastPDL
- locPastPDL is expressively complete for regular trace languages
- Any regular trace language is accepted by a cascade product of local asynchronous transducers:

$$\mathcal{T}_1 \circ \mathcal{T}_2 \circ \cdots \circ \mathcal{T}_n$$

- The gossip problem can be solved with a cascade product of local asynchronous transducers.

Aperiodic = FO-definable

Theorem [Adsul, Gastin, Sarkar, Weil — Concur'20, LMCS'22]

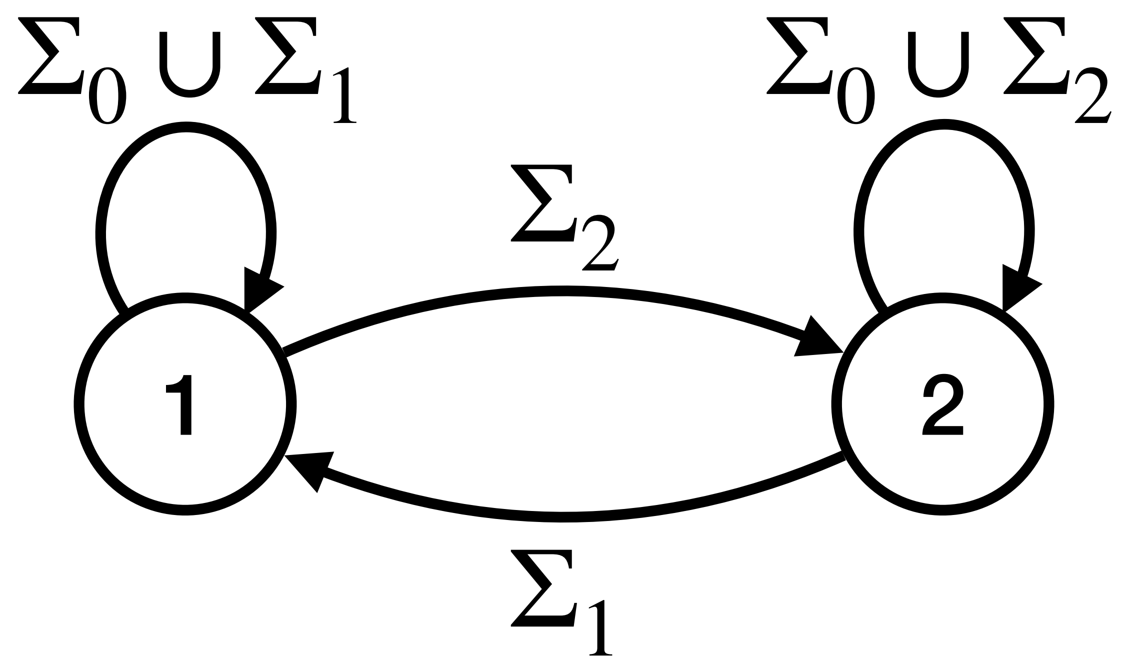
Any aperiodic (FO) trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local reset transducers:

$$\mathscr{G} \circ \mathscr{U}_2 \circ \mathscr{U}_2 \circ \cdots \circ \mathscr{U}_2$$

Direct proof (Not using Krohn-Rhodes theorem)

based on a past temporal logic $\mathrm{LTL}\left(\mathrm{Y}_i \leq \mathrm{Y}_j, \mathrm{S}_i\right)$ proved expressively complete for FO

Reset - $\mathscr{U}_2$



Outline

- Labelling functions, sequential transducers and cascade product
- Krohn-Rhodes theorem for aperiodic/regular word languages
- Model of concurrency: Mazurkiewicz traces and asynchronous Zielonka automata
- Asynchronous labelling functions, transducers and cascade product
- Propositional dynamic logic for traces
- Conclusion

Conclusion

Main results

- (Local) (past) Propositional dynamic logic expressively complete for regular trace languages
  - Specification language: natural, easy, expressive, good complexity
- Cascade decomposition using simple & local asynchronous automata/transducers
  - Allows inductive reasoning on automata
- $\mathcal{U}_2 \circ \cdots \circ \mathcal{U}_2 = \mathrm{locTL}(\mathsf{SS}_i) \subseteq$ Aperiodic = FO = $\mathrm{locTL}(\mathsf{SS}_i, \mathsf{Y}_i \leq \mathsf{Y}_j) = \mathcal{G} \circ \mathcal{U}_2 \circ \cdots \circ \mathcal{U}_2$
  - Equality for acyclic architectures (communication graph).
  - Inclusion strict in general: gossip is (past) first-order definable, but cannot be computed with an aperiodic asynchronous transducer, hence also with a cascade of $\mathcal{U}_2$.

Future work

- Generalisation to other structures, eg, Message sequence charts & Message passing automata?

Thank you for your attention!