Propositional Dynamic Logic and Asynchronous Cascade Decompositions for Regular Trace Languages

CONCUR, Sept. 2022, Warsaw

Paul Gastin
LMF, ENS Paris-Saclay, IRL ReLaX

Joint Work with Bharat Adsul (IIT Bombay), Saptarshi Sarkar (IIT Bombay) and Pascal Weil (LaBRI, Univ. Bordeaux)

Outline

- Model of Concurrency: Mazurkiewicz Traces
- A propositional dynamic logic for traces
- Asynchronous automata (Zielonka) and asynchronous labeling functions
- Cascade product / decomposition (Krohn-Rhodes)
- Conclusion

Mazurkiewicz Traces

Architecture

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

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

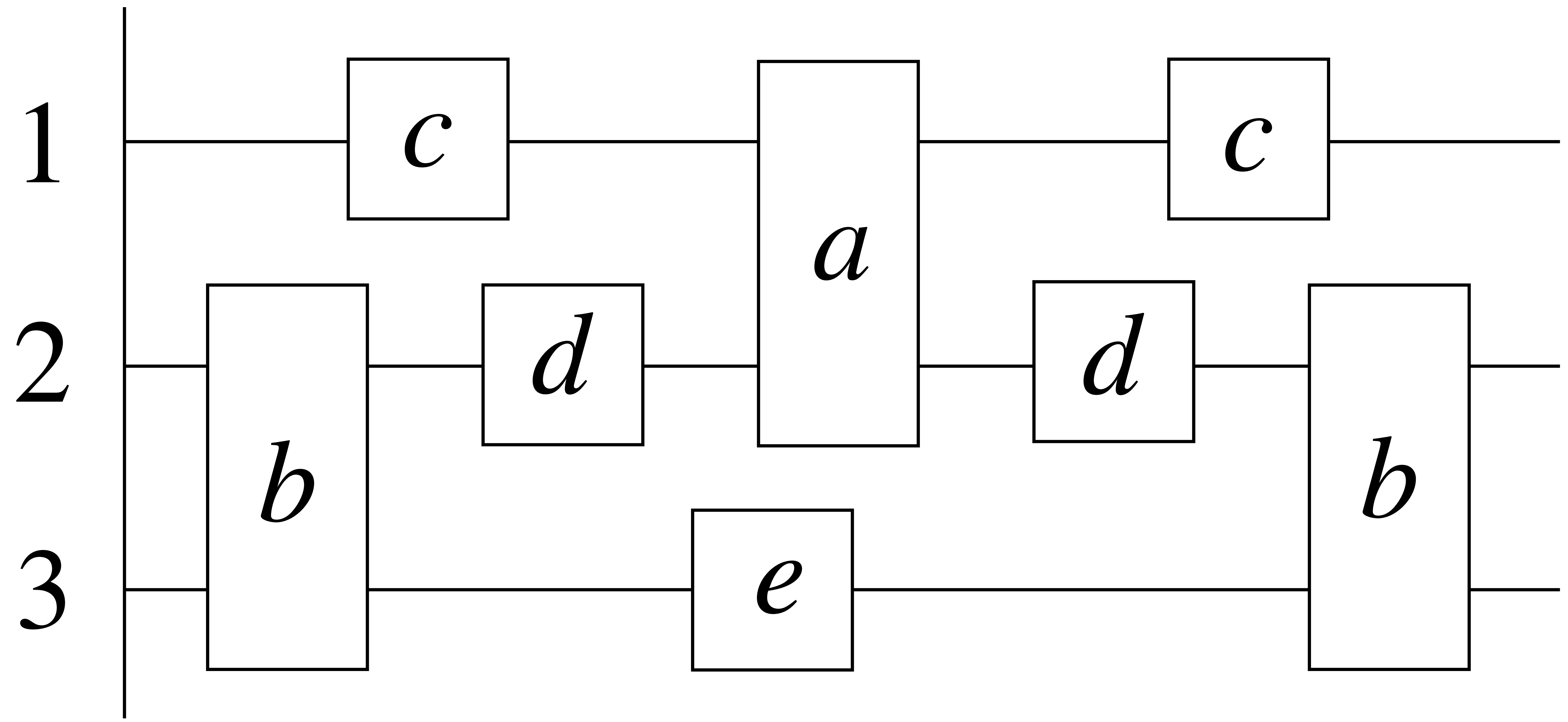
$\mathrm{loc}(a) = \{1,2\}$

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

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

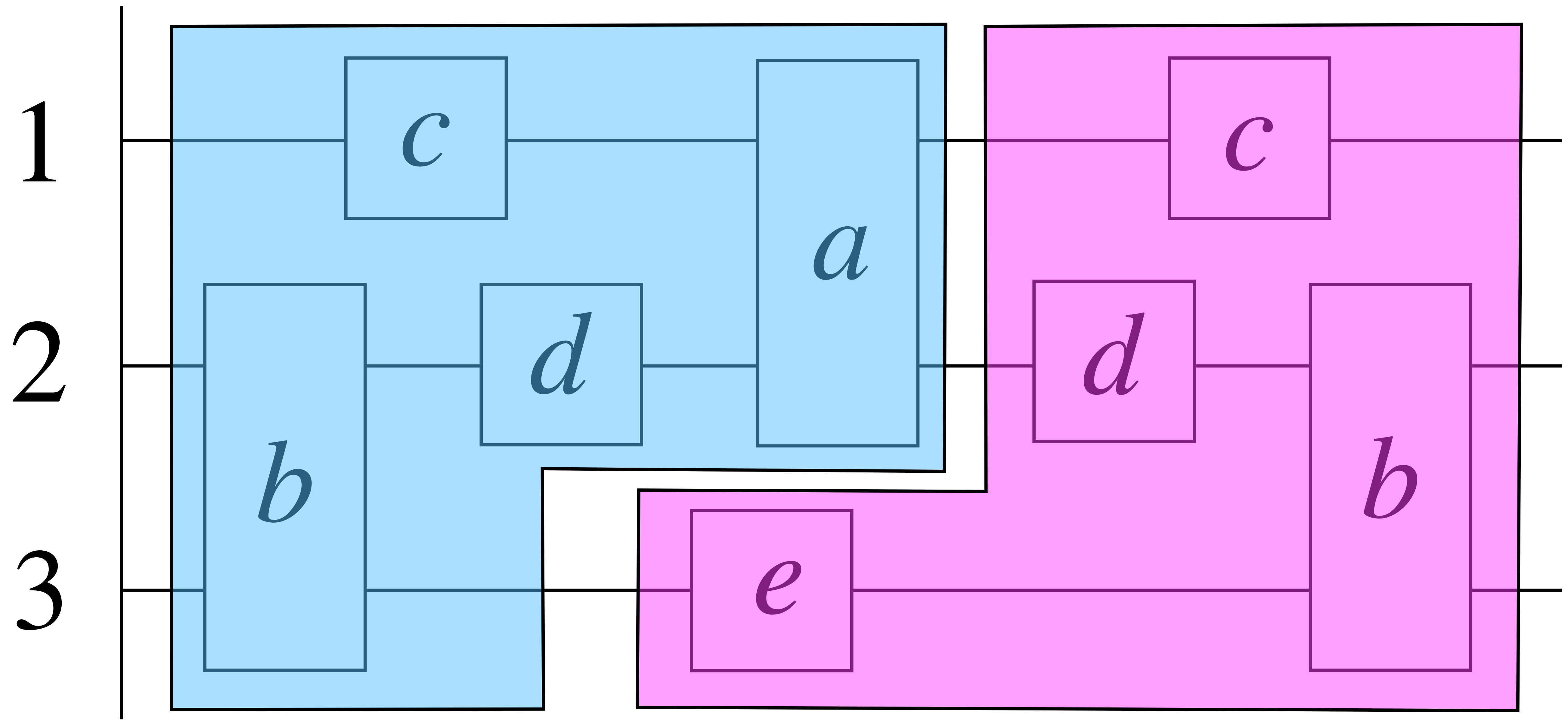
$\mathrm{loc}(d) = \{2\}$

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

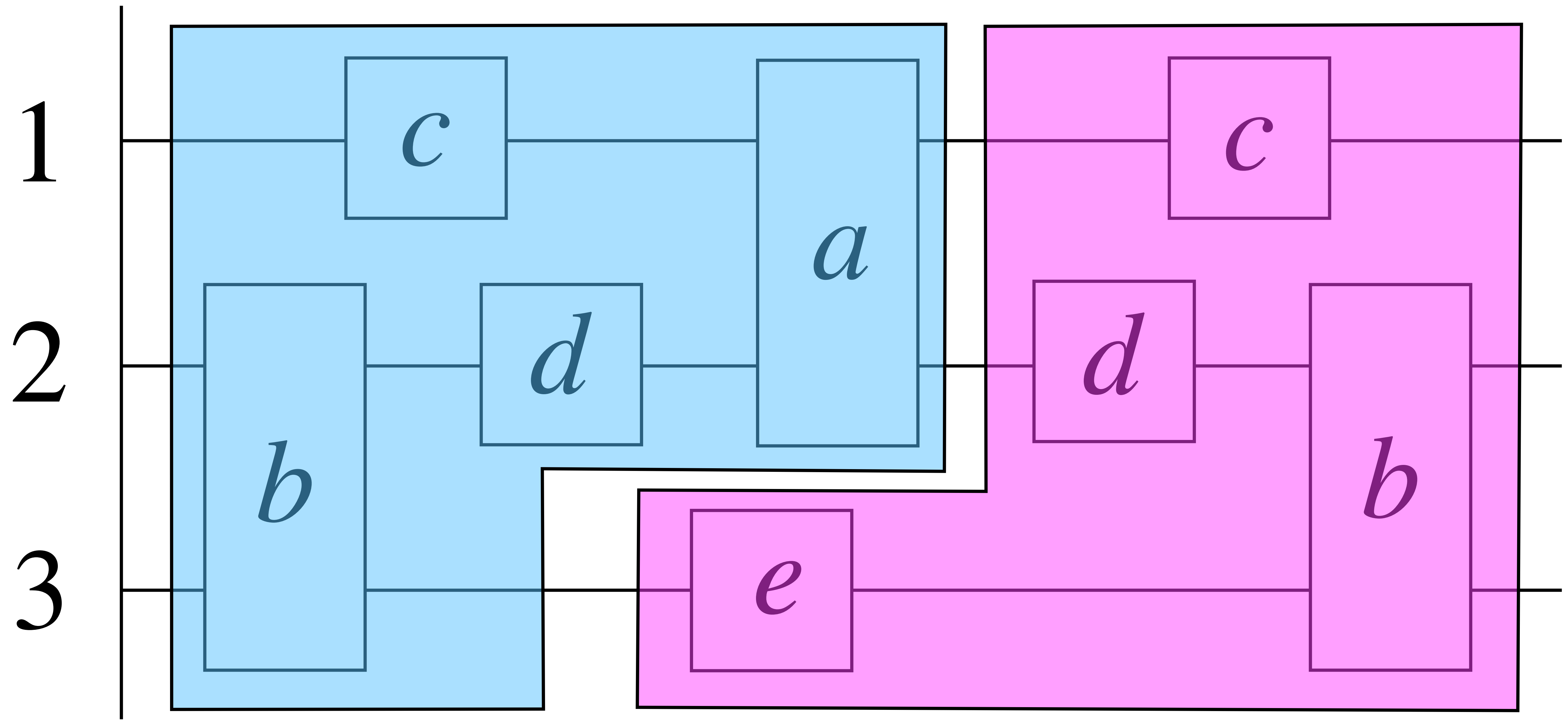


- set of traces denoted $Tr(\Sigma, \mathscr{P}, \mathrm{loc})$ or simply $Tr(\Sigma)$
- Trace Language: $L \subseteq Tr(\Sigma)$

Concatenation, Independence, Commutation, Monoid

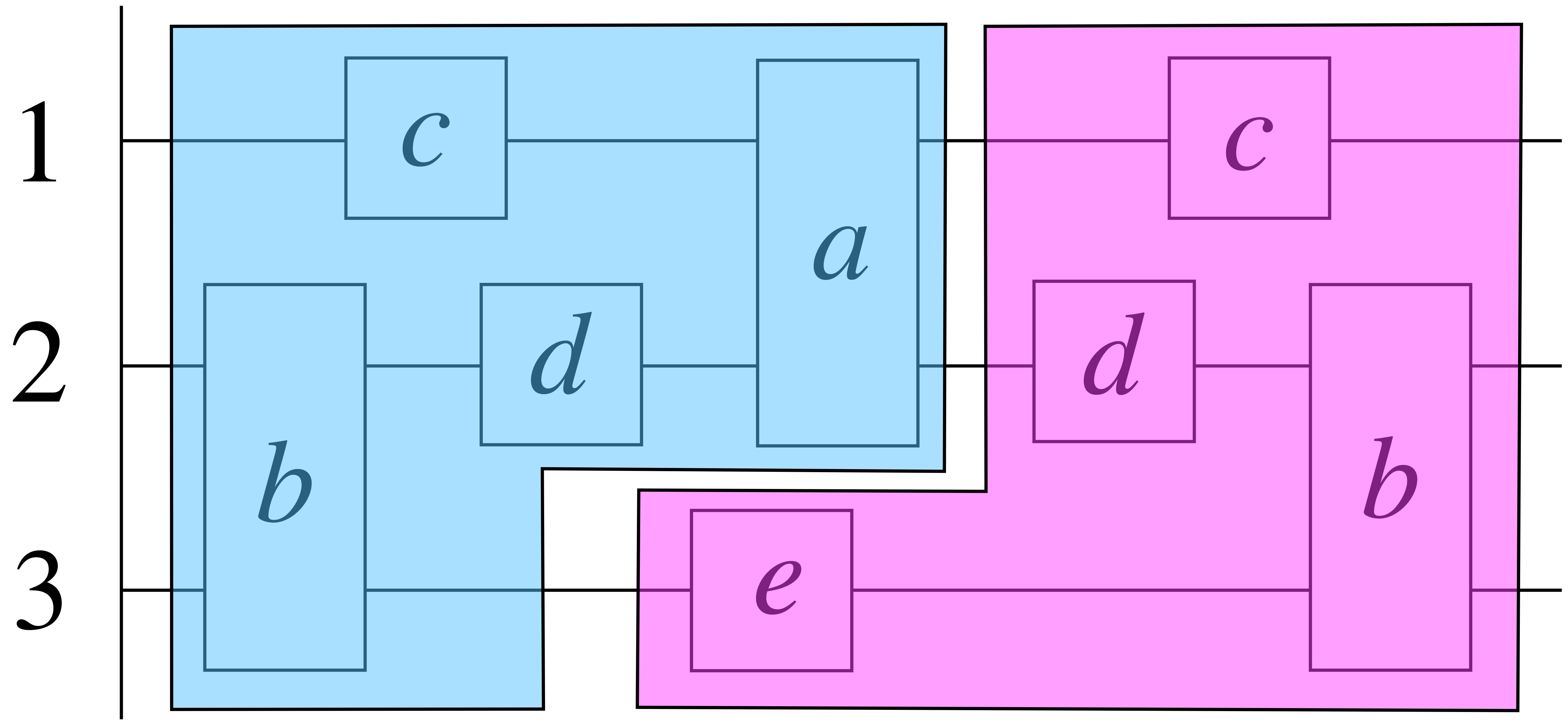


Concatenation, Independence, Commutation, Monoid



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

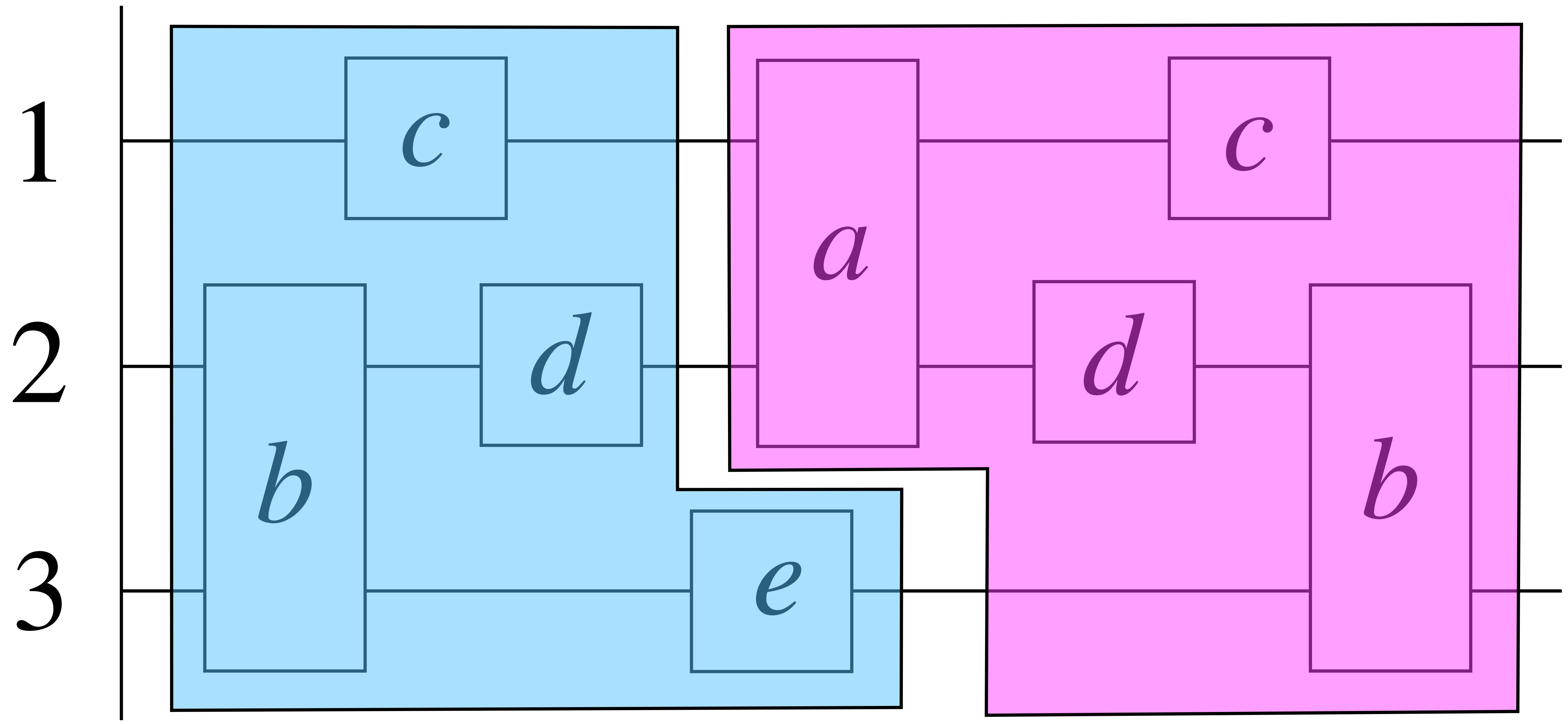
Concatenation, Independence, Commutation, Monoid



- $x \; I \; y$ 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*

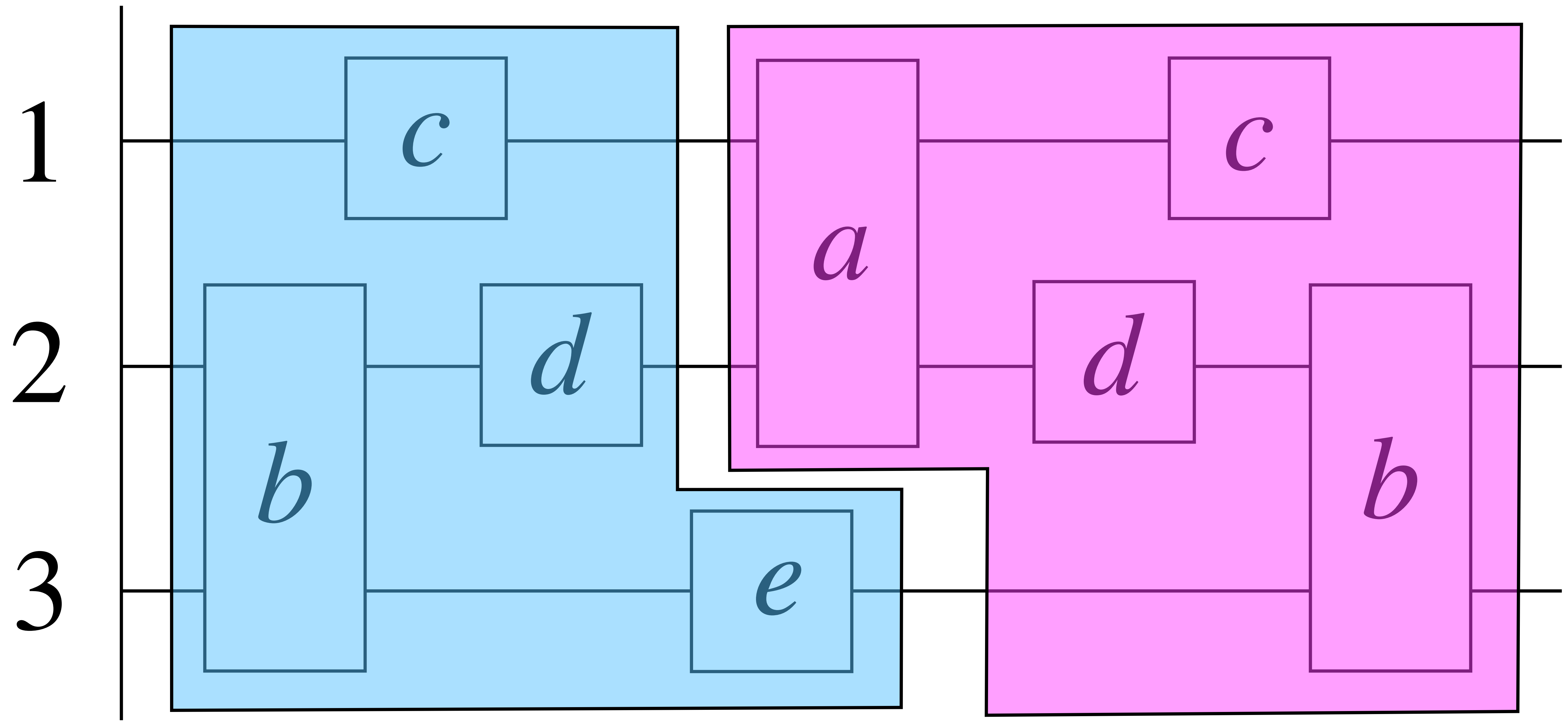
Concatenation, Independence, Commutation, Monoid



- $x \; I \; y$ 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*

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*
- $L \subseteq Tr(\Sigma)$ *is regular if* $L = \eta^{-1}(\eta(L))$ *for some morphism* $\eta \colon Tr(\Sigma) \to M$ *(finite monoid)*

	Expressions	Computation	Logic	Algebra
Regular Trace Languages	*Regular expressions with restricted star operation*	*Diamond automata*	*Monadic second order logic*	*Morphisms to finite Monoids*

Ochmański   Muscholl   Diekert   Gastin   Restivo   Salemi   Thomas   et. al.

	Expressions	Computation	Logic	Algebra
Regular Trace Languages	*Regular expressions with restricted star operation*	*Diamond automata*	*Monadic second order logic*	*Morphisms to finite Monoids*
$\supsetneq$				
Aperiodic Trace Languages	*Star-free expressions*	*Counter-free diamond automata*	*First order logic* *Temporal logic*	*Aperiodic Monoids*

Ochmański Muscholl Diekert Gastin Restivo Salemi Thomas et. al.

Outline

- ✅ Model of Concurrency: Mazurkiewicz Traces
- A propositional dynamic logic for traces
- Asynchronous automata (Zielonka) and asynchronous labeling functions
- Cascade product / decomposition (Krohn-Rhodes)
- Conclusion

Propositional Dynamic Logic

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 $\pi ::= \varphi? \mid x := e \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

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 $(\varphi? \cdot \pi_1) + (\neg\varphi? \cdot \pi_2)$

Propositional Dynamic Logic

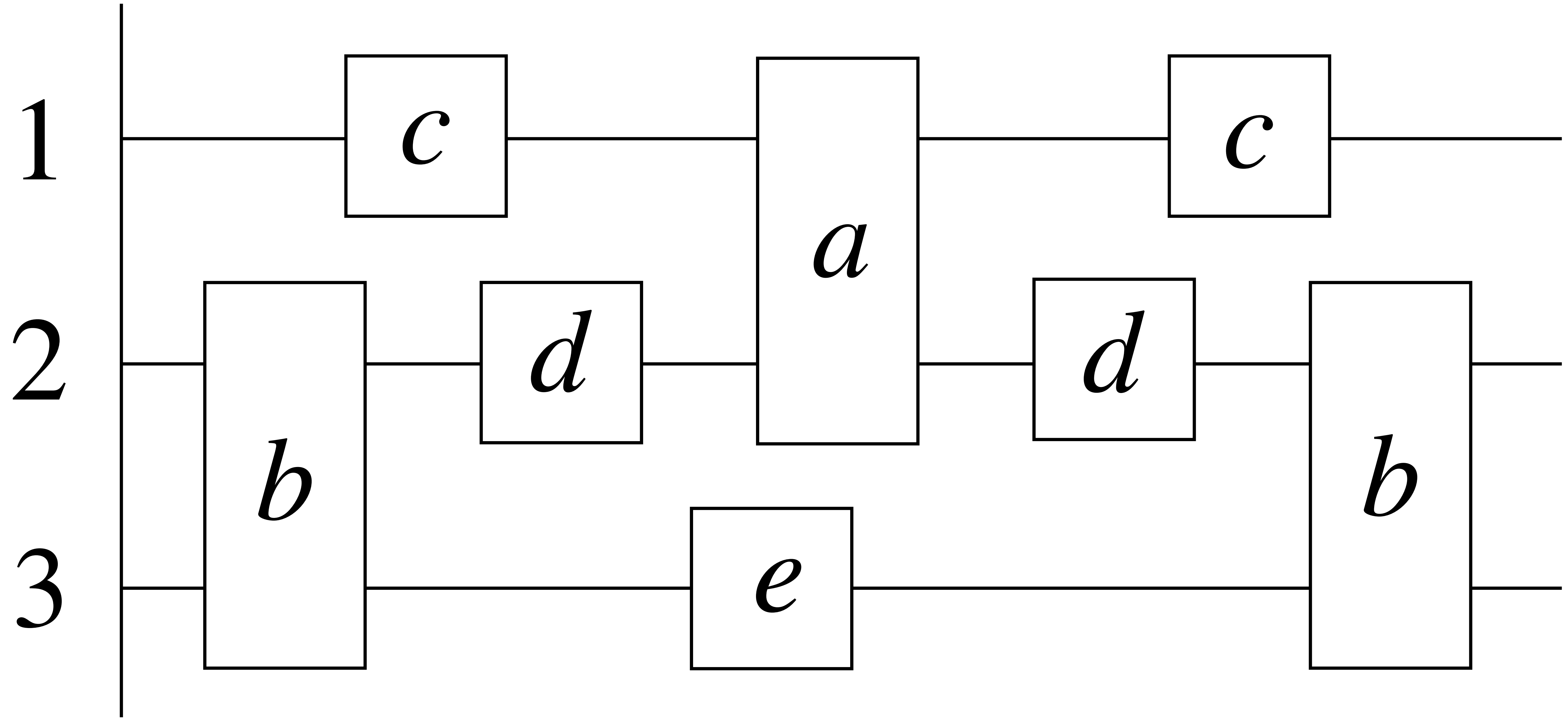
- First introduced to reason about programs (Fischer, Ladner 1979)
  - State formulas $\varphi ::= p \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi$
  - Program expressions $\pi ::= \varphi? \mid x := e \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$
    - If φ then π_1 else π_2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

Propositional Dynamic Logic

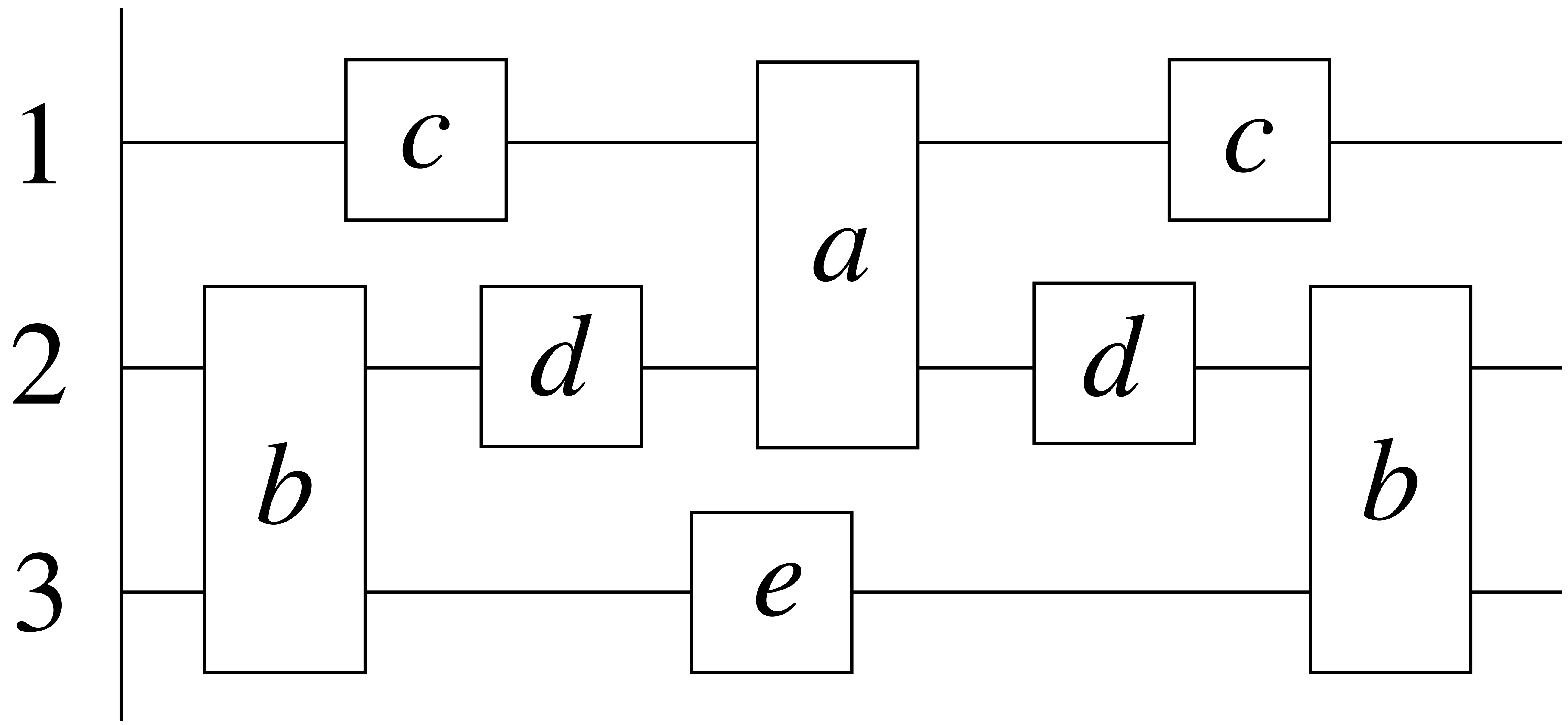
- First introduced to reason about programs (Fischer, Ladner 1979)
  - State formulas $\varphi ::= p \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi$
  - Program expressions $\pi ::= \varphi? \mid x := e \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$
    - If φ then π_1 else π_2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

Regular word languages = MSO definable = LDL definable

Past PDL for Traces

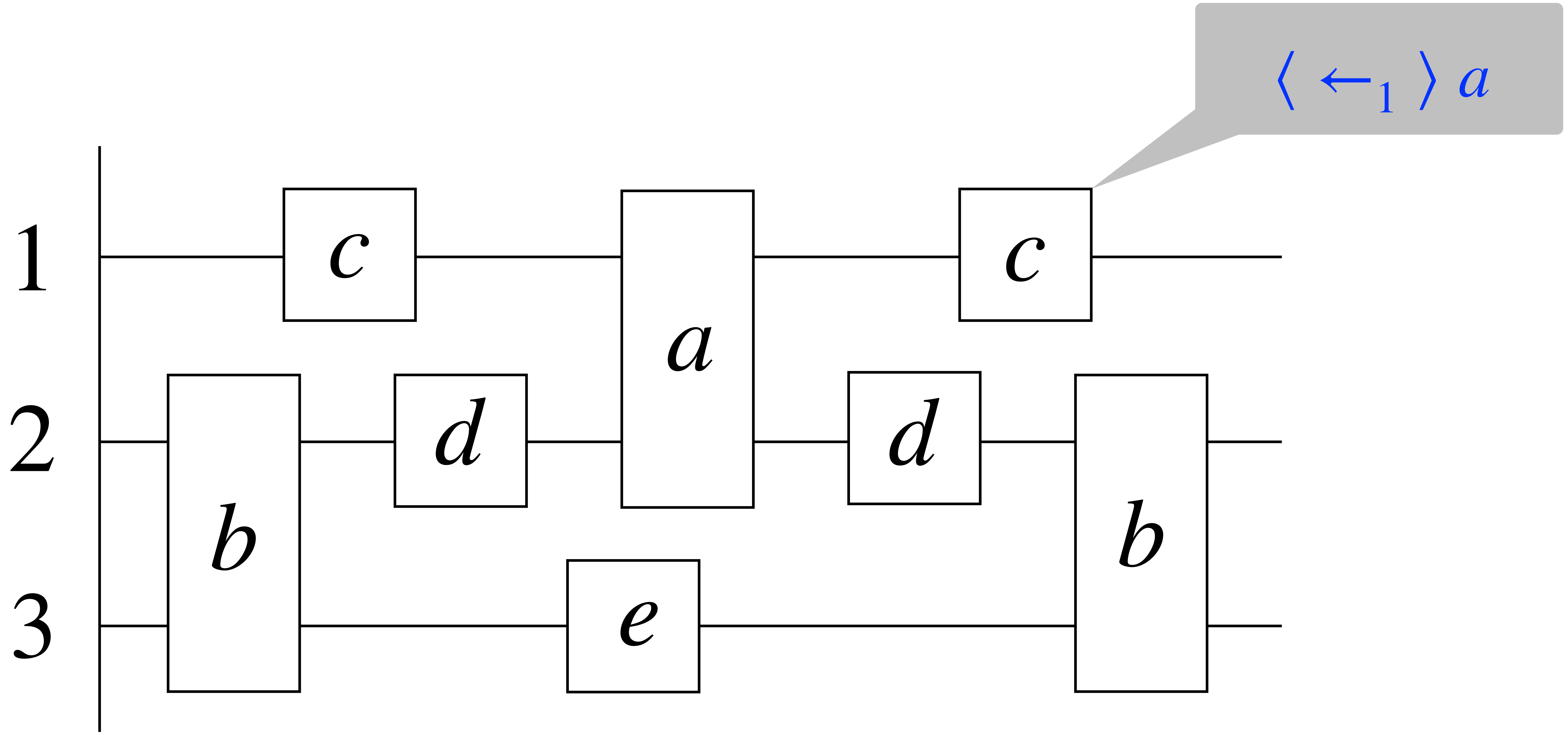


Past PDL for Traces



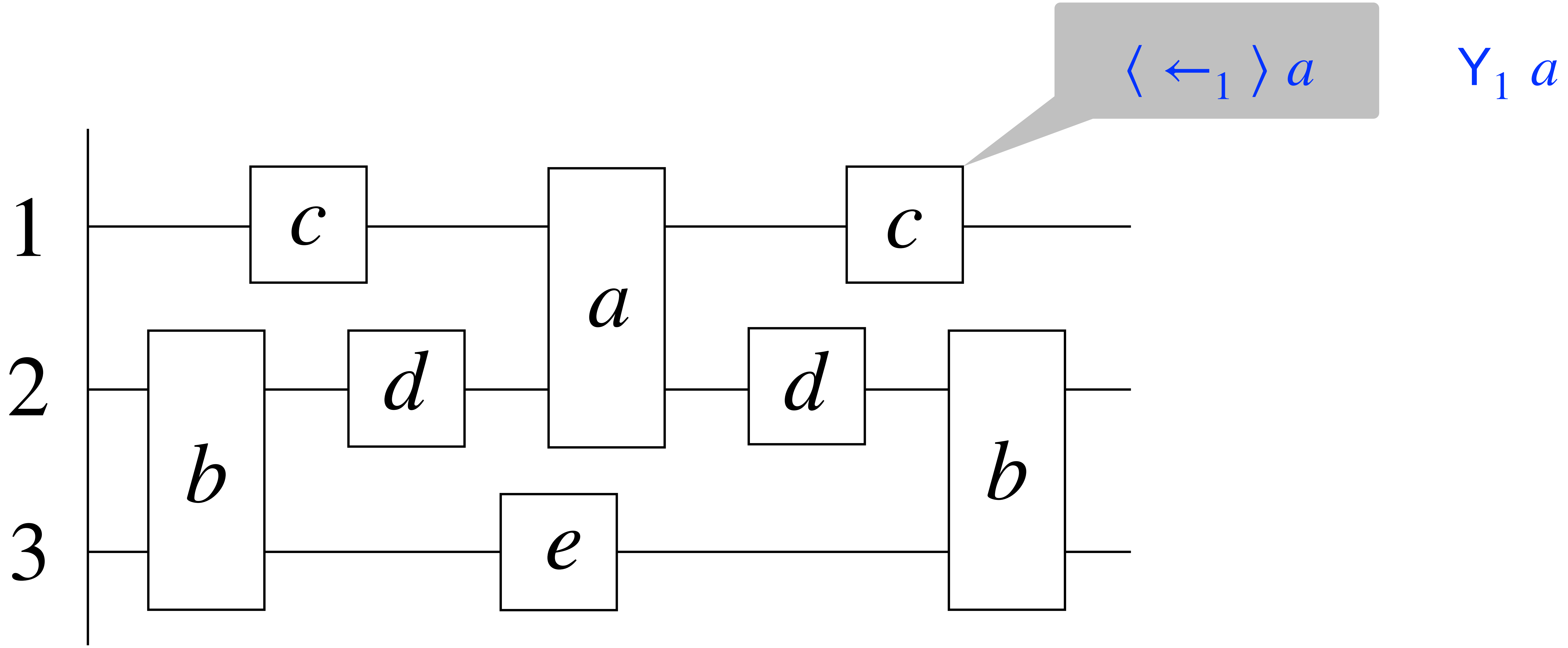
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces



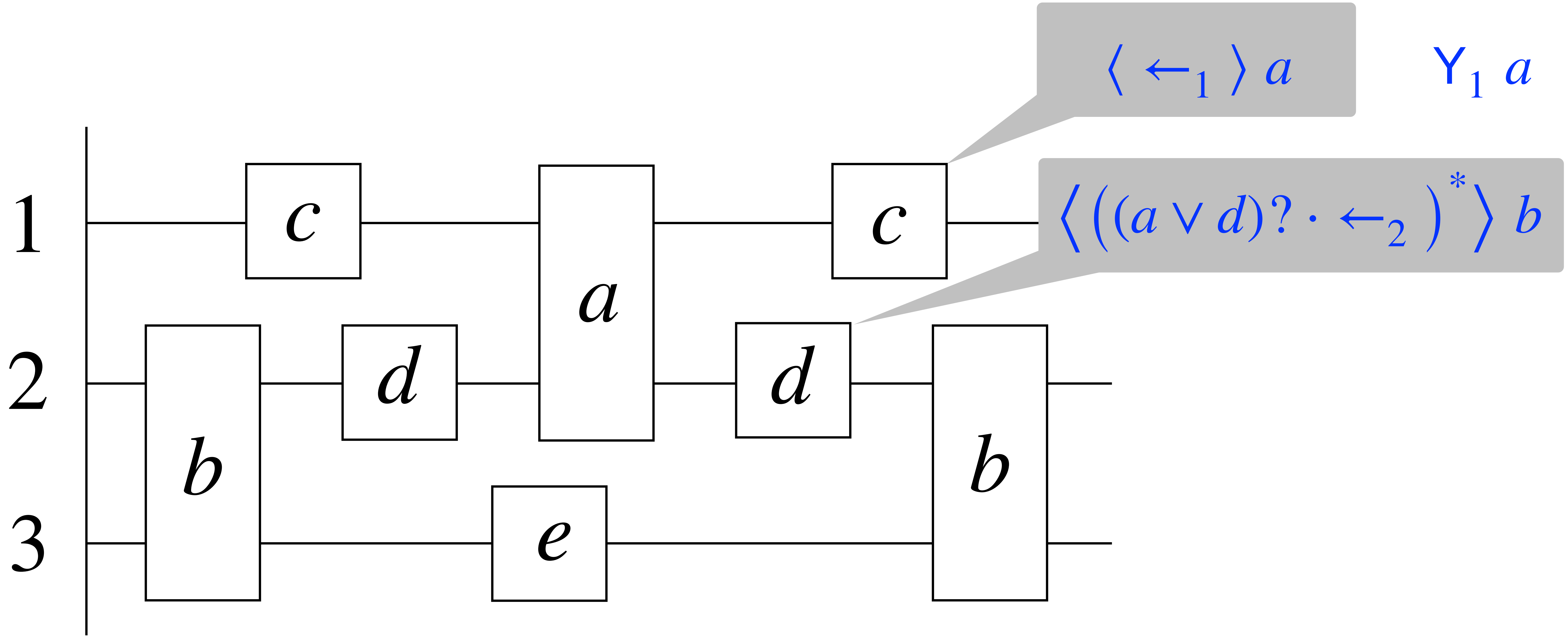
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces



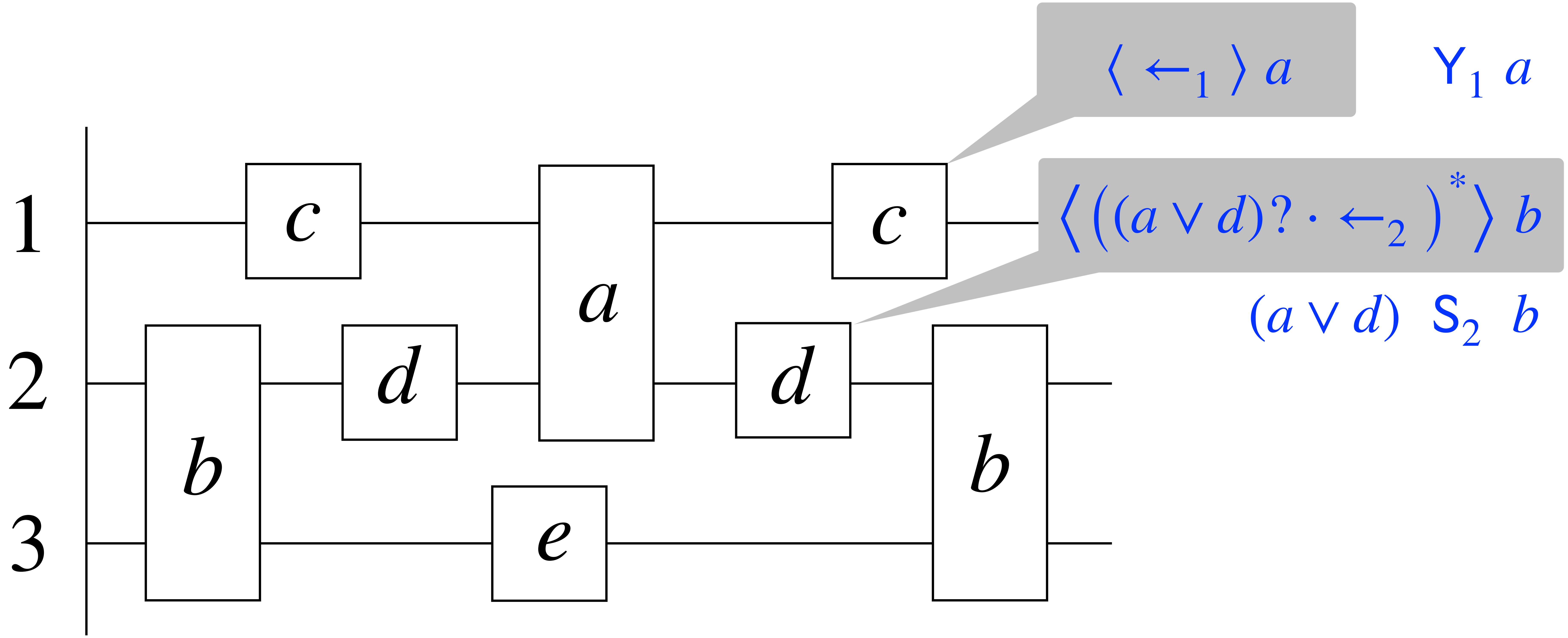
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg \varphi \mid \langle \pi \rangle \varphi$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces



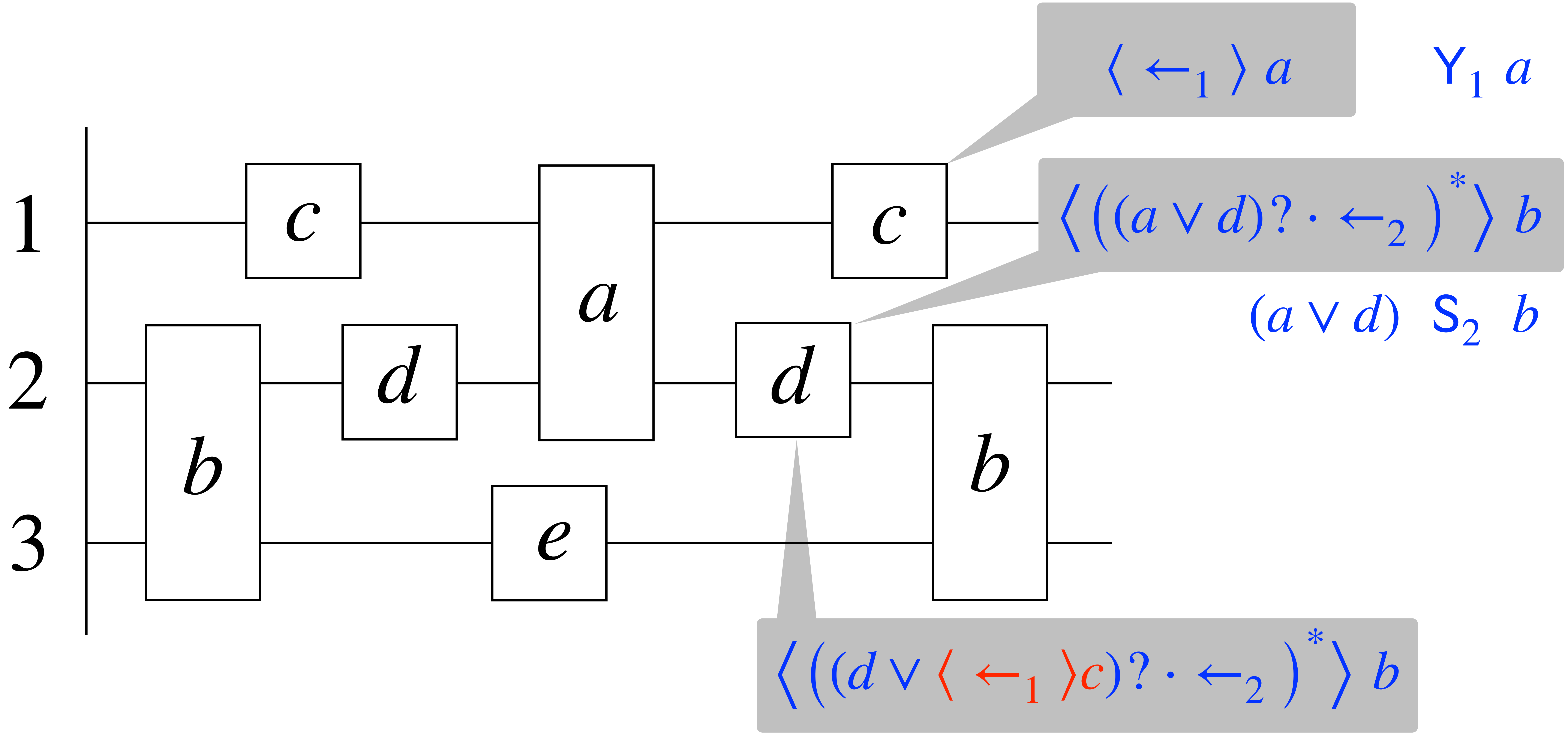
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces



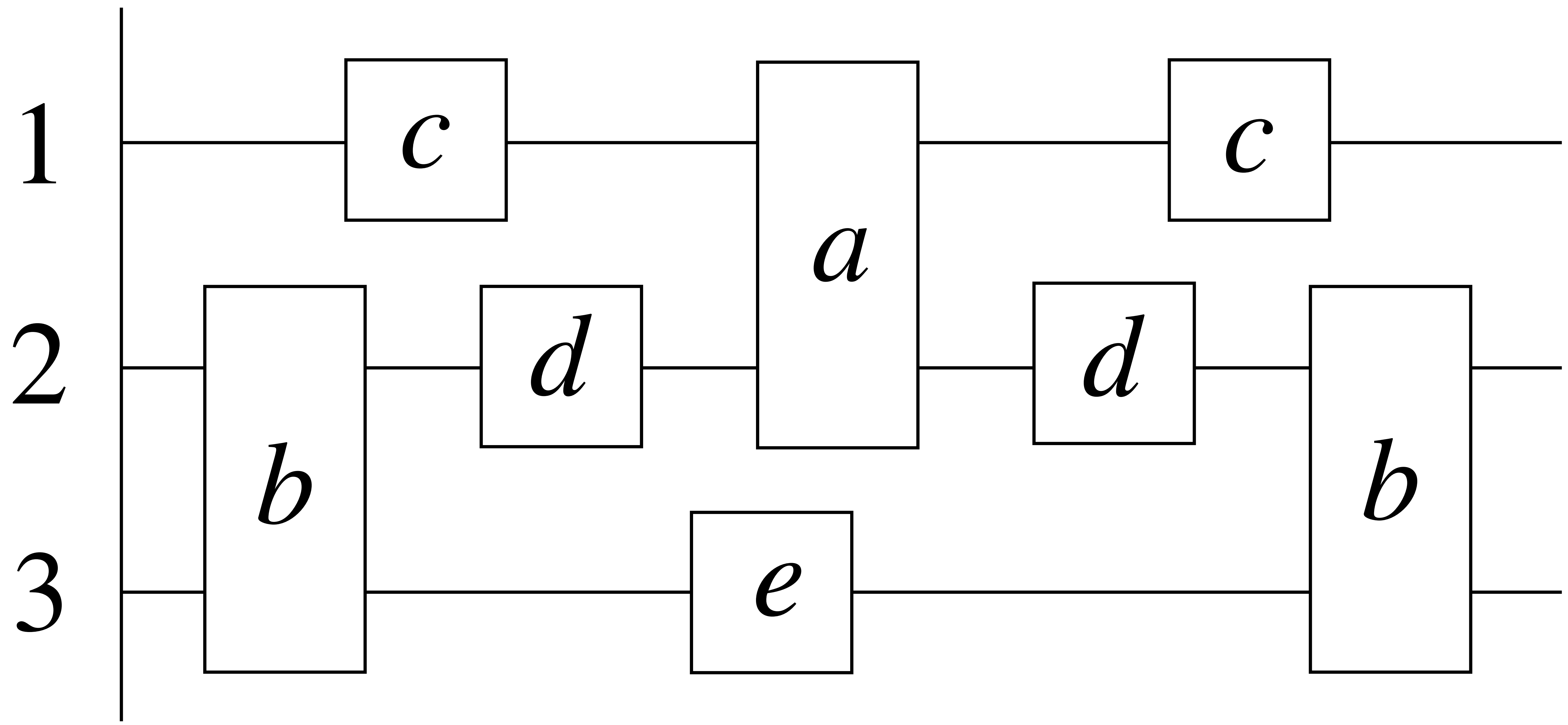
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle \pi \rangle \varphi$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces



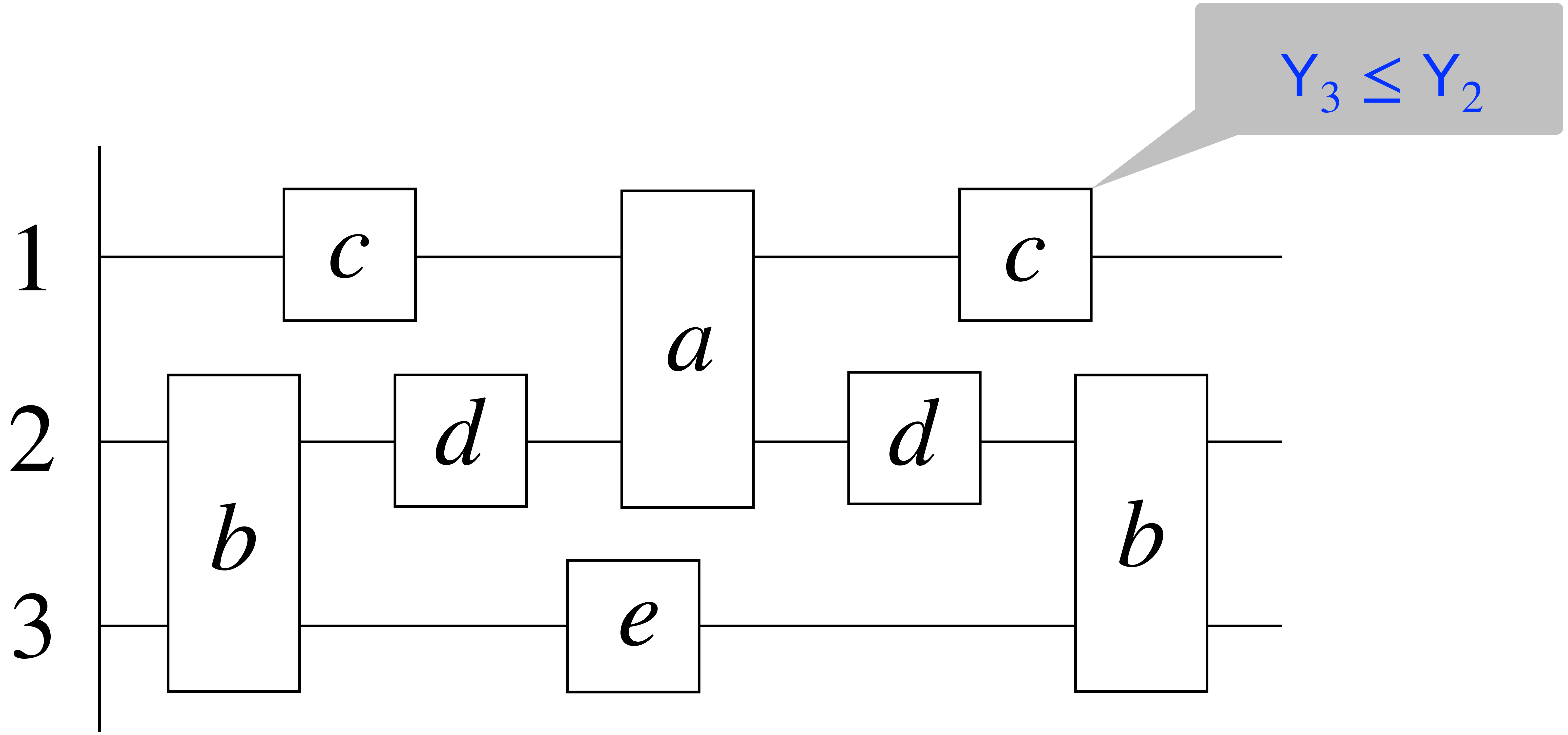
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg \varphi \mid \langle \pi \rangle \varphi$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces



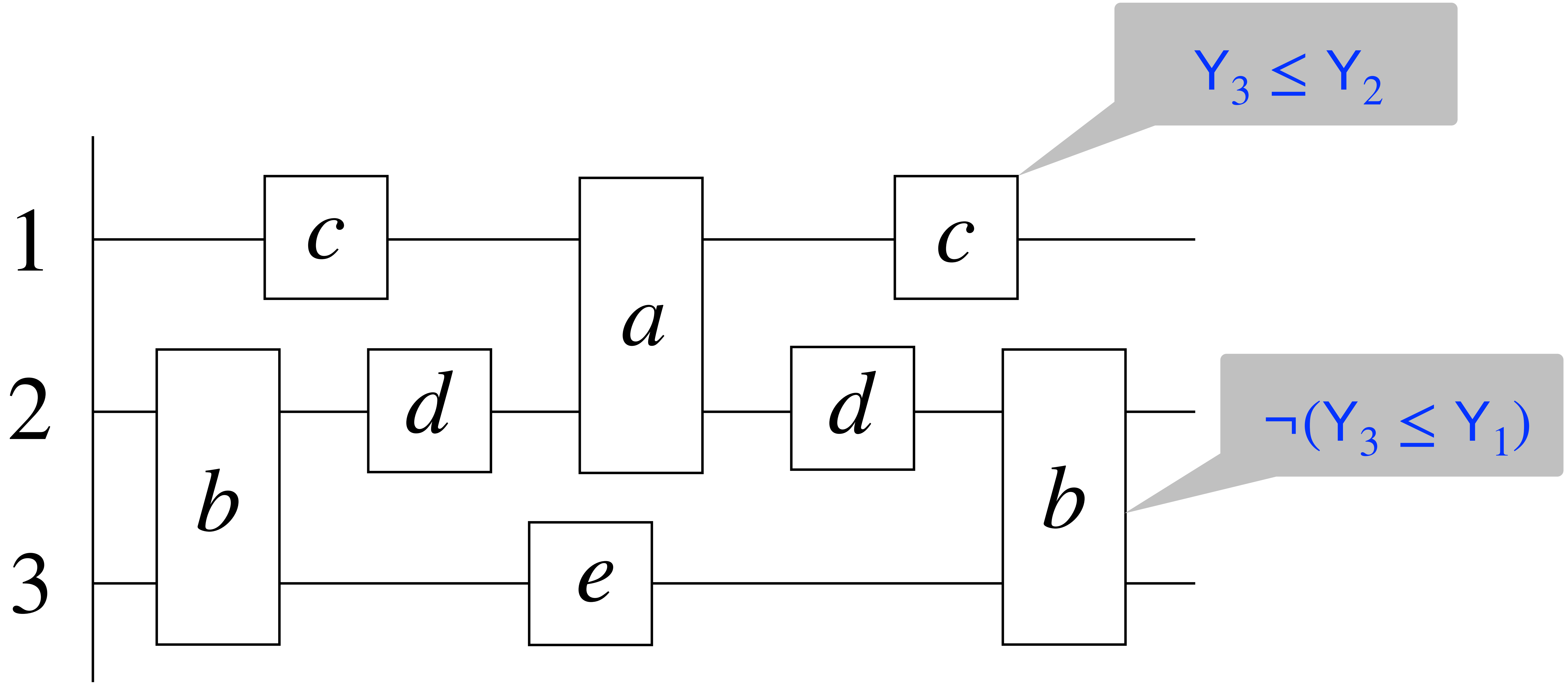
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi \mid \mathsf{Y}_i \leq \mathsf{Y}_j$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces



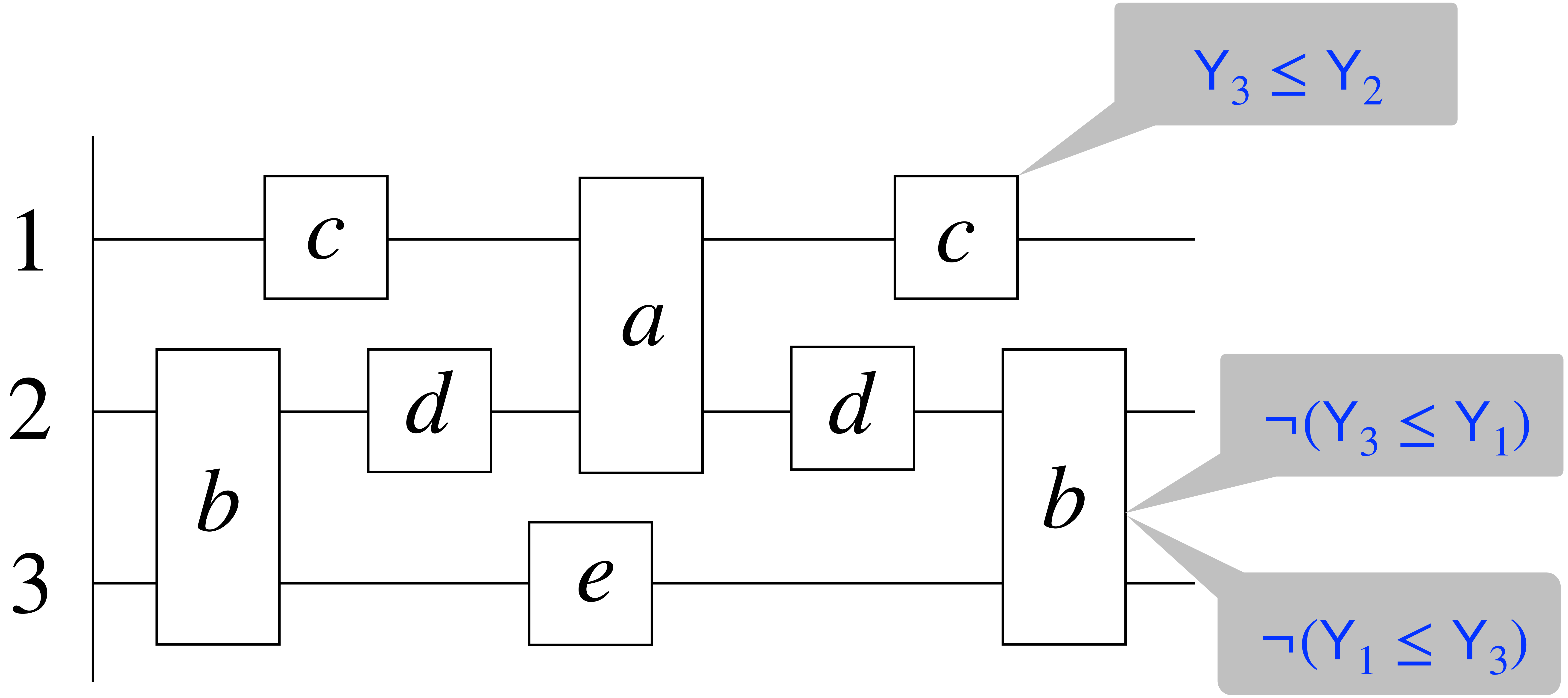
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi \mid \mathsf{Y}_i \leq \mathsf{Y}_j$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces



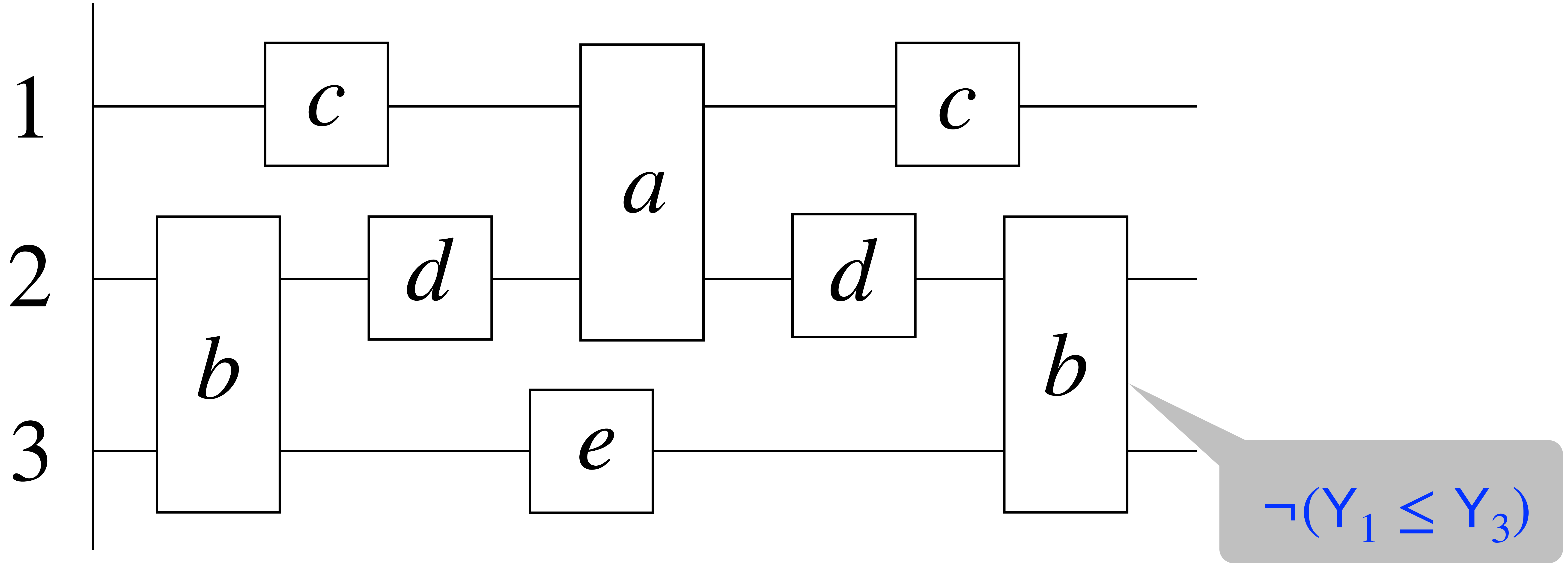
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi \mid Y_i \leq Y_j$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces



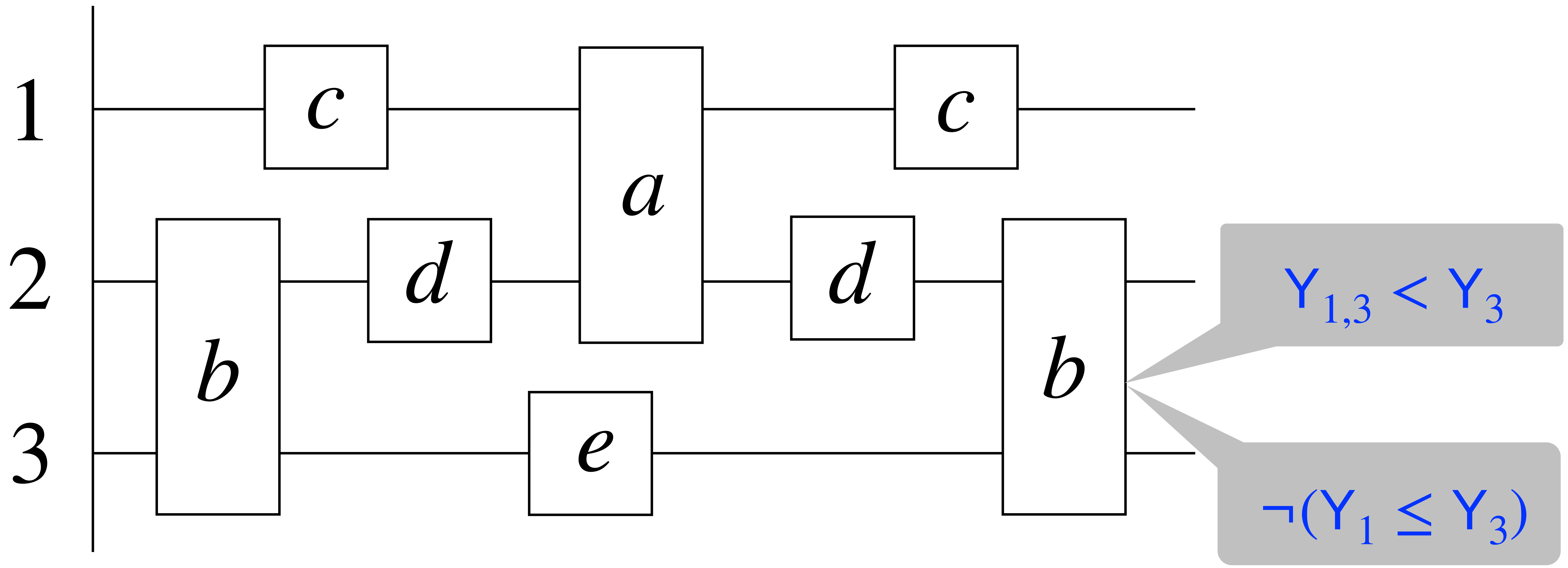
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi \mid Y_i \leq Y_j$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces



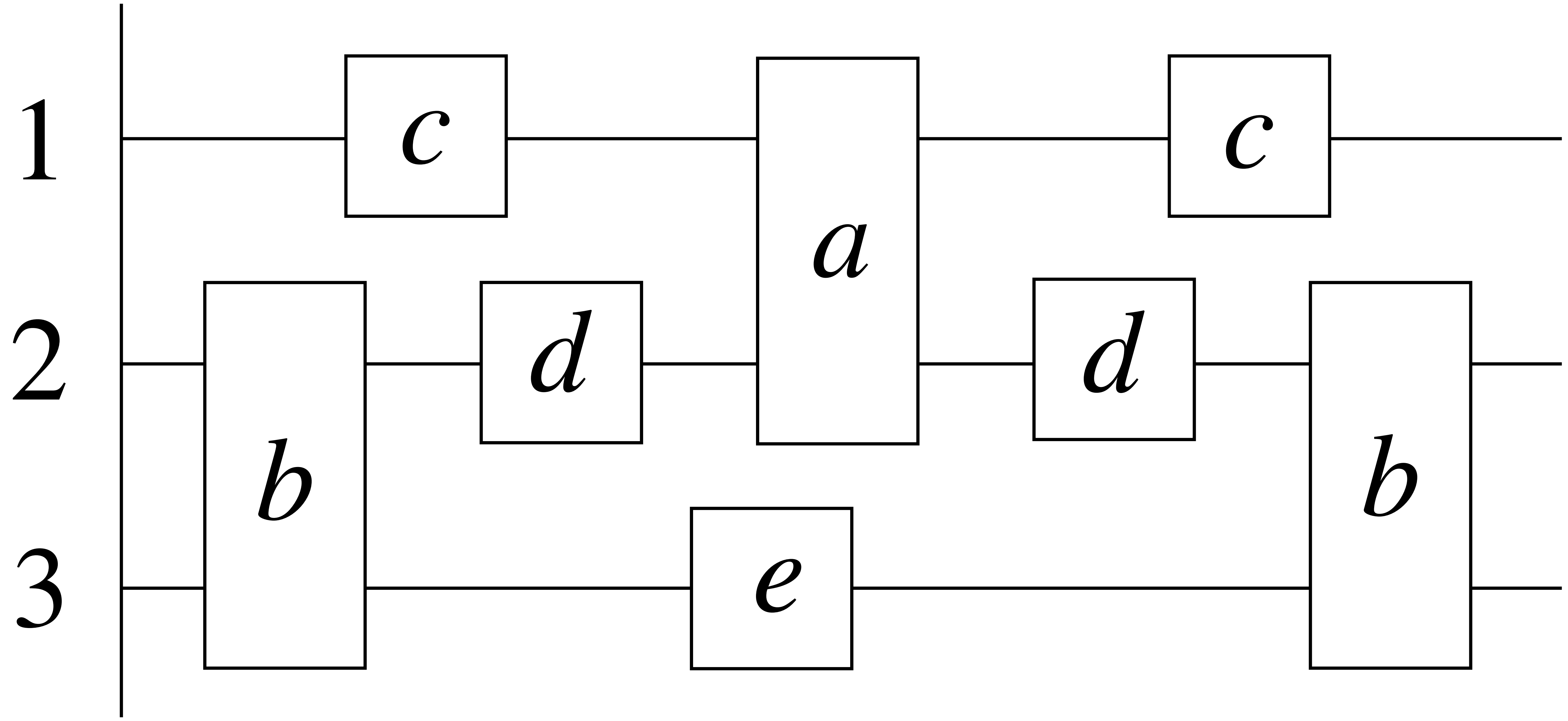
- State/Event formulas $\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi \mid \mathsf{Y}_i \leq \mathsf{Y}_j \mid \mathsf{Y}_{i,k} \leq \mathsf{Y}_j$
- Program/Path expressions $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces



- 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 $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

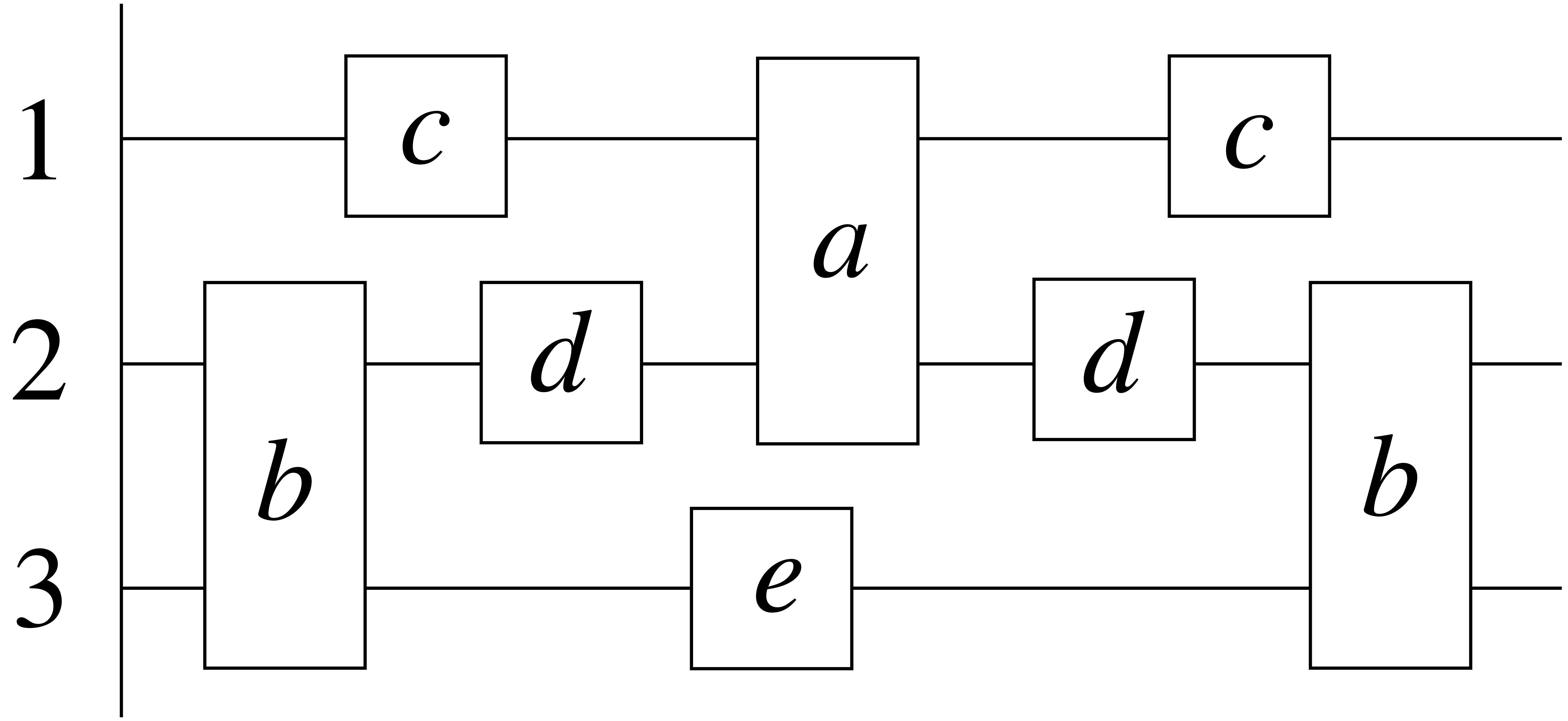
Past PDL for Traces



- 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 $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces

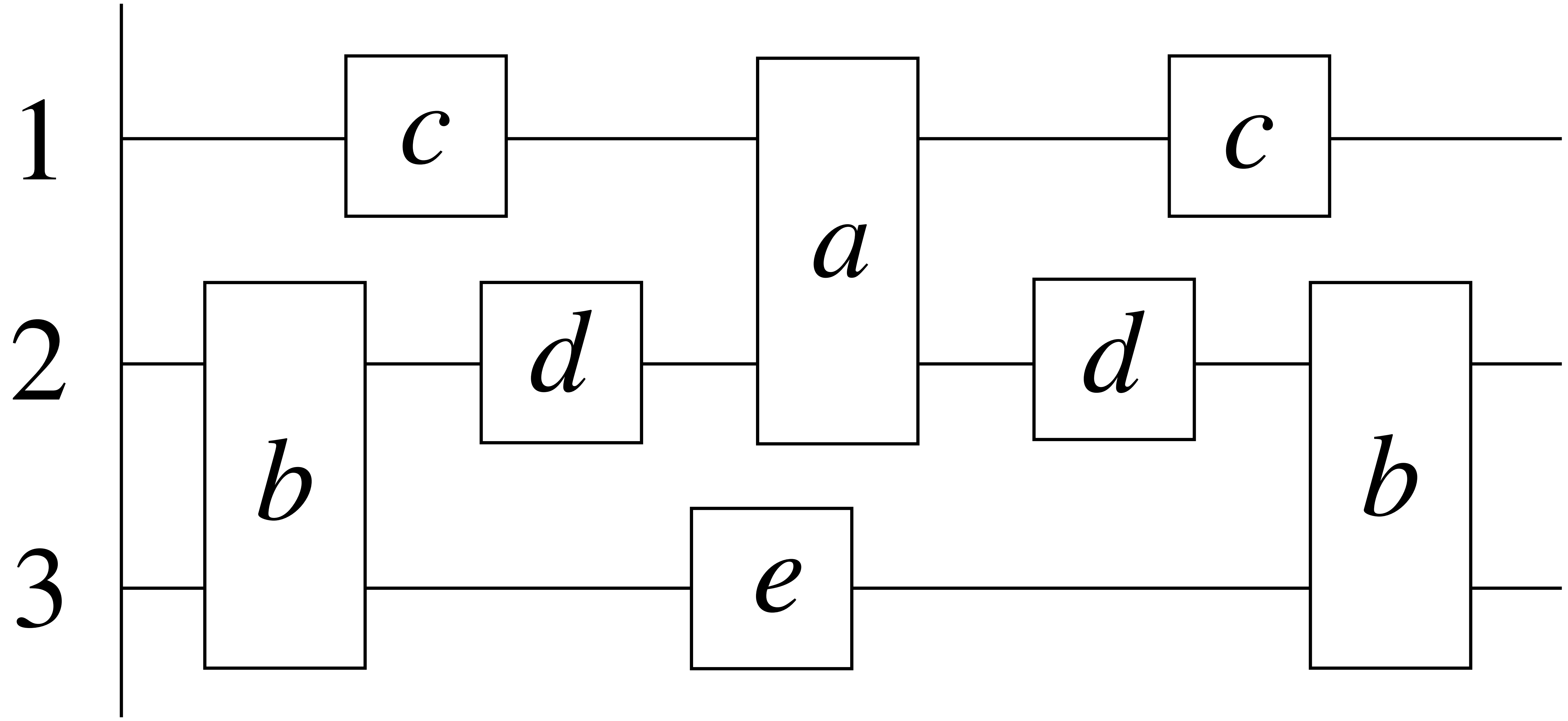
- $T \models \mathsf{EM}_1\, c$



- 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 $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces

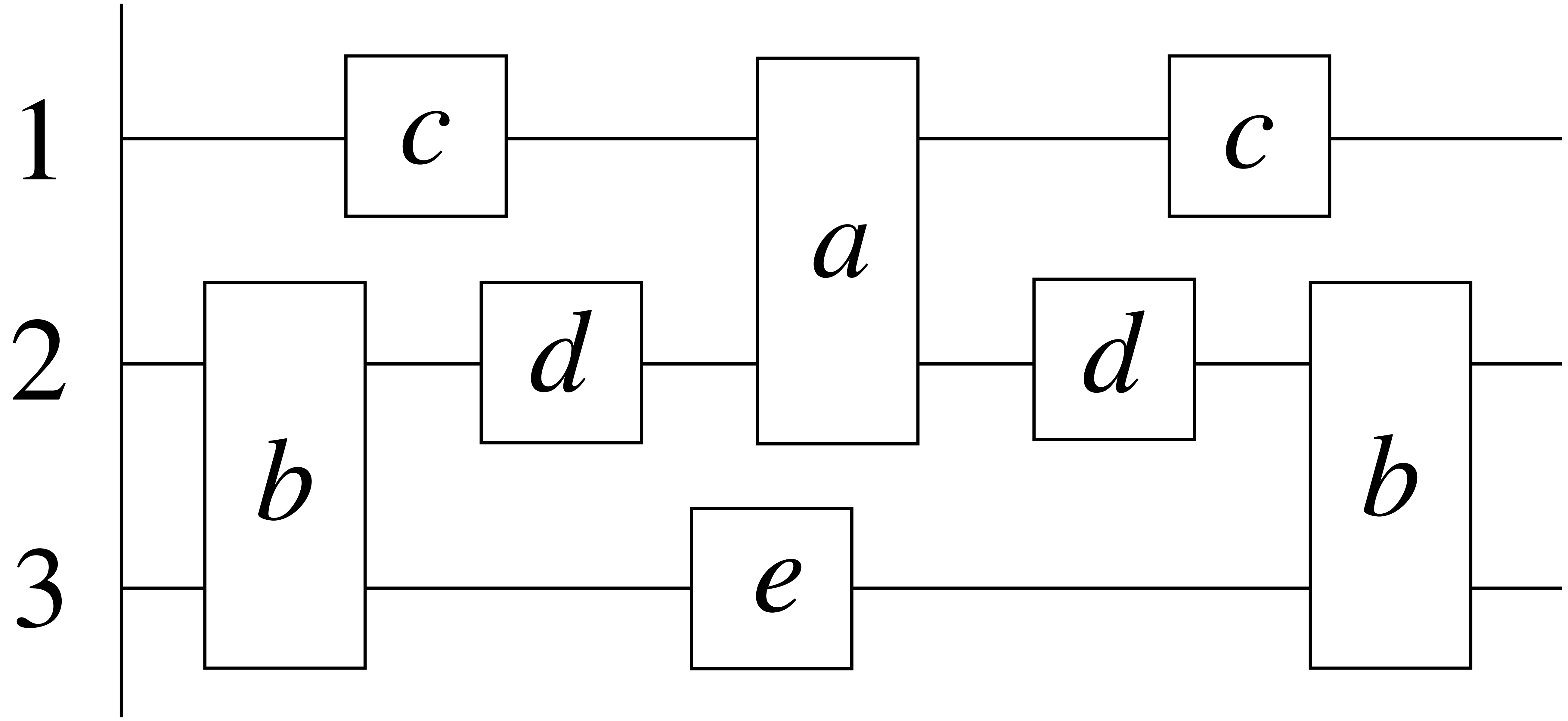
- $T \vDash \mathsf{EM}_1\ c$
- $T \vDash \mathsf{EM}_2\ \langle \leftarrow_3 \rangle\ e$



- 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 $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces

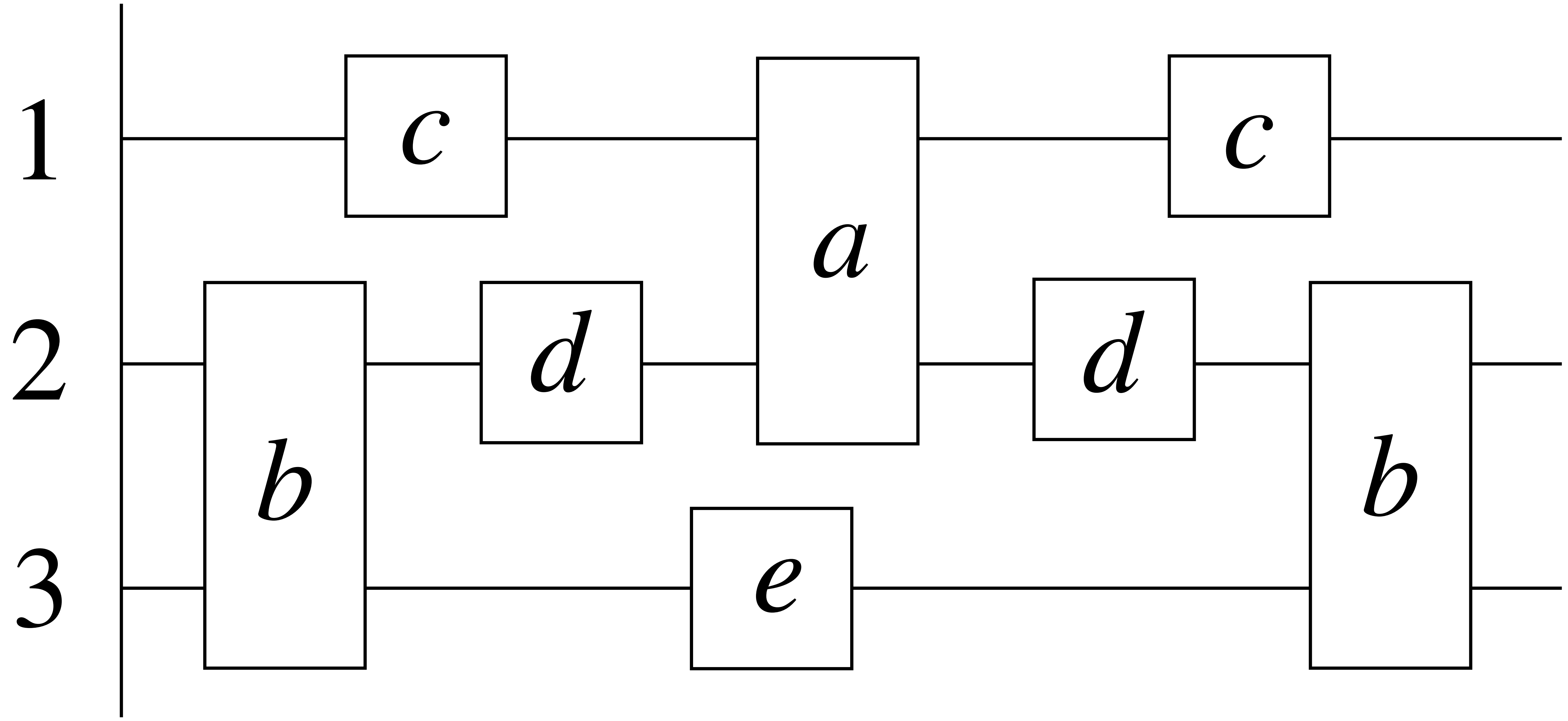
- $T \vDash \mathsf{EM}_1\ c$
- $T \vDash \mathsf{EM}_2\ \langle \leftarrow_3 \rangle\ e$
- $T \vDash \neg(\mathsf{L}_1 \leq \mathsf{L}_3)$



- 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 $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

Past PDL for Traces

- $T \models \mathsf{EM}_1\ c$
- $T \models \mathsf{EM}_2\ \langle \leftarrow_3 \rangle\ e$
- $T \models \neg(\mathsf{L}_1 \leq \mathsf{L}_3)$
- $T \models \mathsf{L}_{1,2} < \mathsf{L}_3$



- 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 $\pi ::= \varphi? \mid \leftarrow_i \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$

(Local) Past-PDL for Traces

Theorem 1

PastPDL = Regular Trace Languages

(Local) Past-PDL for Traces

Theorem 1

locPastPDL = PastPDL = Regular Trace Languages

(Local) Past-PDL for Traces

Theorem 1

locPastPDL = PastPDL = Regular Trace Languages

locPastPDL

$\subseteq$ fragment

PastPDL

(Local) Past-PDL for Traces

Theorem 1

locPastPDL = PastPDL = Regular Trace Languages

locPastPDL

$\subseteq$ fragment

PastPDL

$\subseteq$ easy

MSO

(Local) Past-PDL for Traces

Theorem 1

locPastPDL = PastPDL = Regular Trace Languages

locPastPDL

$\subseteq$ fragment

PastPDL

$\subseteq$ easy

MSO

$\subseteq$ known

Morphisms

(Local) Past-PDL for Traces

Theorem 1

locPastPDL = PastPDL = Regular Trace Languages

locPastPDL

I∩ fragment

PastPDL

I∩ easy

MSO

I∩ known

Morphisms

I∩ complex & subtle

locPastPDL

(Local) Past-PDL for Traces

Theorem 1

locPastPDL = PastPDL = Regular Trace Languages

locPastPDL

$\subseteq$ fragment

PastPDL

$\subseteq$ easy

MSO

$\subseteq$ known

Morphisms

$\subseteq$ complex & subtle

locPastPDL

- Let $\eta : Tr(\Sigma) \to M$ be a morphism to a finite monoid
- For each $m \in M$, we construct a locPastPDL sentence $\Phi^{(m)}$ which defines $\eta^{-1}(m)$

(Local) Past-PDL for Traces

Theorem 1

locPastPDL = PastPDL = Regular Trace Languages

locPastPDL

$\subseteq$ fragment

PastPDL

$\subseteq$ easy

MSO

$\subseteq$ known

Morphisms

$\subseteq$ complex & subtle

locPastPDL

- Let $\eta : Tr(\Sigma) \to M$ be a morphism to a finite monoid
- For each $m \in M$, we construct a locPastPDL sentence $\Phi^{(m)}$ which defines $\eta^{-1}(m)$
- First, construct a locPastPDL event formula $\varphi^{(m)}$ such that, if T is a prime trace (i.e., having a single maximal event), $\eta(T) = m$ if and only if $T, \max(T) \models \varphi^{(m)}$

  (induction on the number of processes, depends crucially on $Y_i \leq Y_j$, $Y_{i,k} \leq Y_j$)

(Local) Past-PDL for Traces

Theorem 1

PastPDL = Regular Trace Languages

locPastPDL

I∩ fragment

PastPDL

I∩ easy

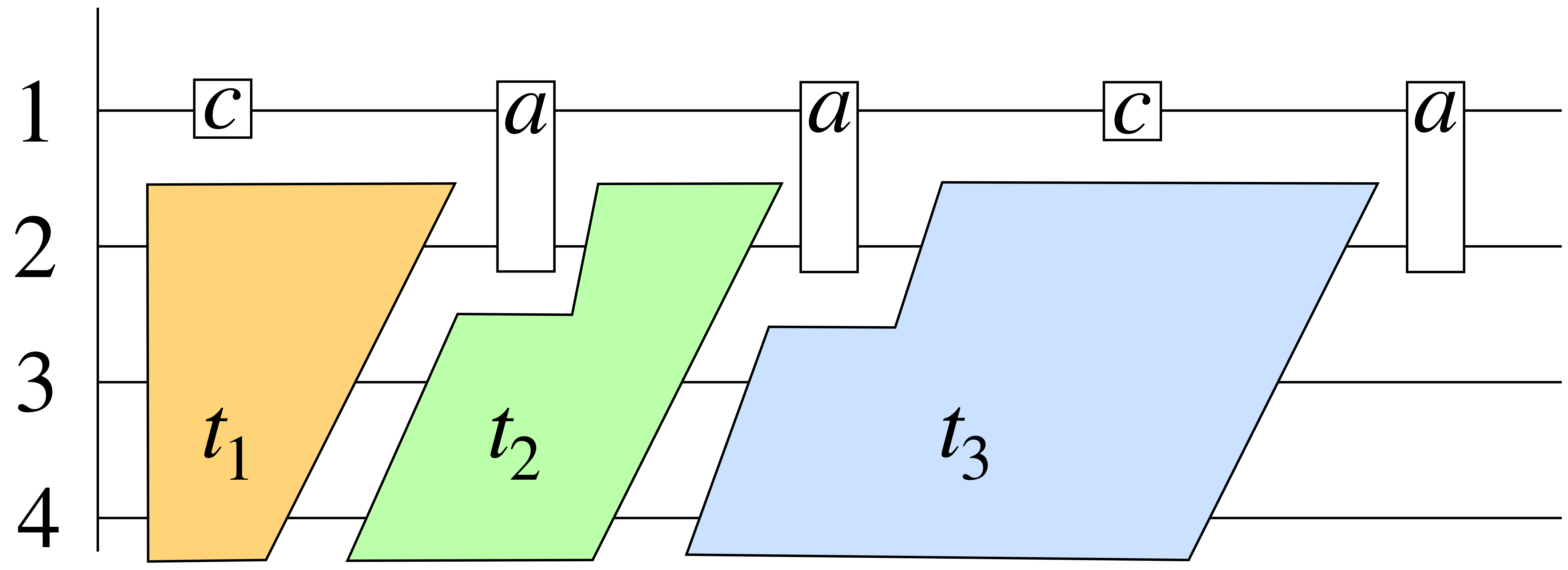
MSO

I∩ known

Morphisms

I∩ complex & subtle

locPastPDL



(Local) Past-PDL for Traces

Theorem 1

locPastPDL = PastPDL = Regular Trace Languages

locPastPDL

$\subseteq$ fragment

PastPDL

$\subseteq$ easy

MSO

$\subseteq$ known

Morphisms

$\subseteq$ complex & subtle

locPastPDL

- Let $\eta : Tr(\Sigma) \to M$ be a morphism to a finite monoid
- For each $m \in M$, we construct a locPastPDL sentence $\Phi^{(m)}$ which defines $\eta^{-1}(m)$
- First, construct a locPastPDL event formula $\varphi^{(m)}$ such that, if T is a prime trace (i.e., having a single maximal event), $\eta(T) = m$ if and only if $T, \max(T) \models \varphi^{(m)}$ (induction on the number of processes, depends crucially on $Y_i \leq Y_j$, $Y_{i,k} \leq Y_j$)
- For the general case, decompose an arbitrary (non prime) trace into a product of prime traces.

(Local) Past-PDL for Traces

Theorem 1

PastPDL = Regular Trace Languages

locPastPDL

I∩ fragment

PastPDL

I∩ easy

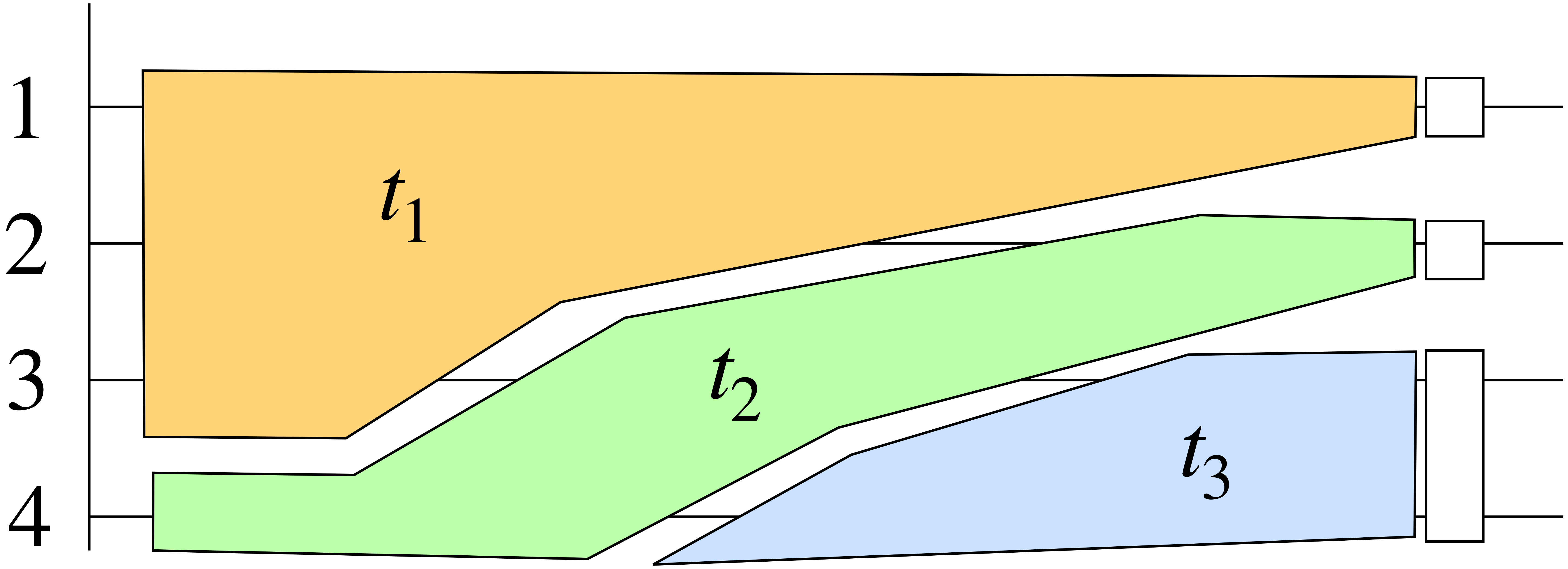
MSO

I∩ known

Morphisms

I∩ complex & subtle

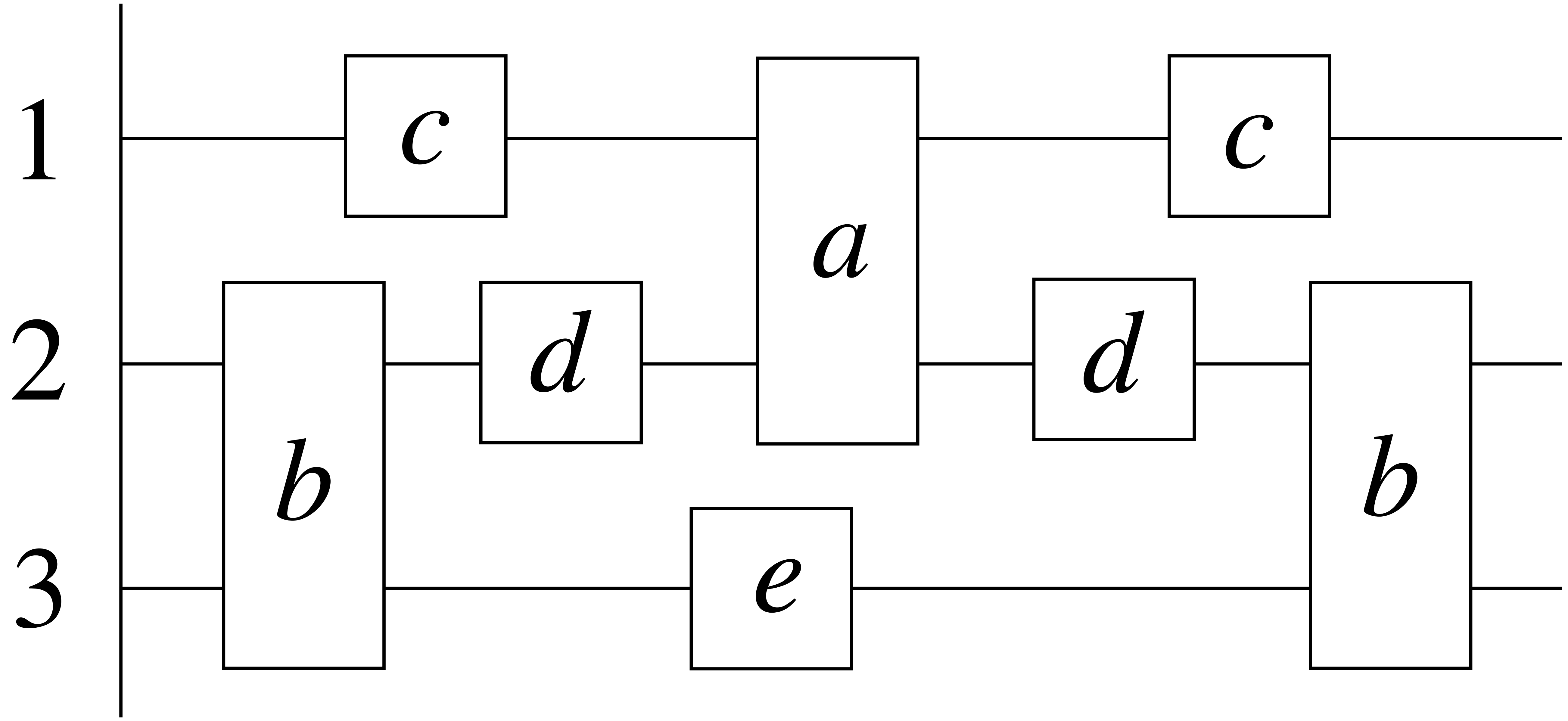
locPastPDL



Outline

- ✓ Model of Concurrency: Mazurkiewicz Traces
- ✓ A propositional dynamic logic for traces
- Asynchronous automata (Zielonka) and asynchronous labeling functions
- Cascade product / decomposition (Krohn-Rhodes)
- Conclusion

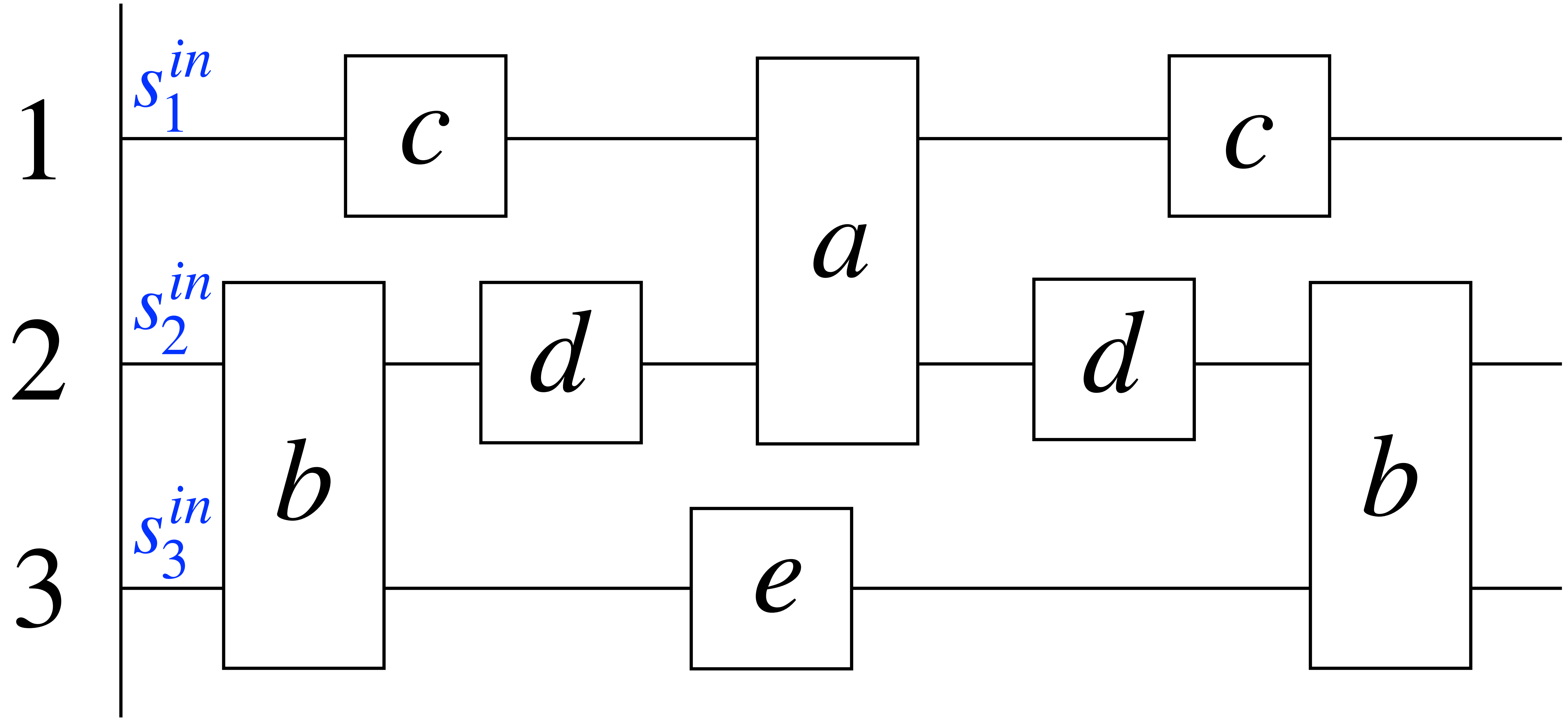
Asynchronous automata (Zielonka)



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

- 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

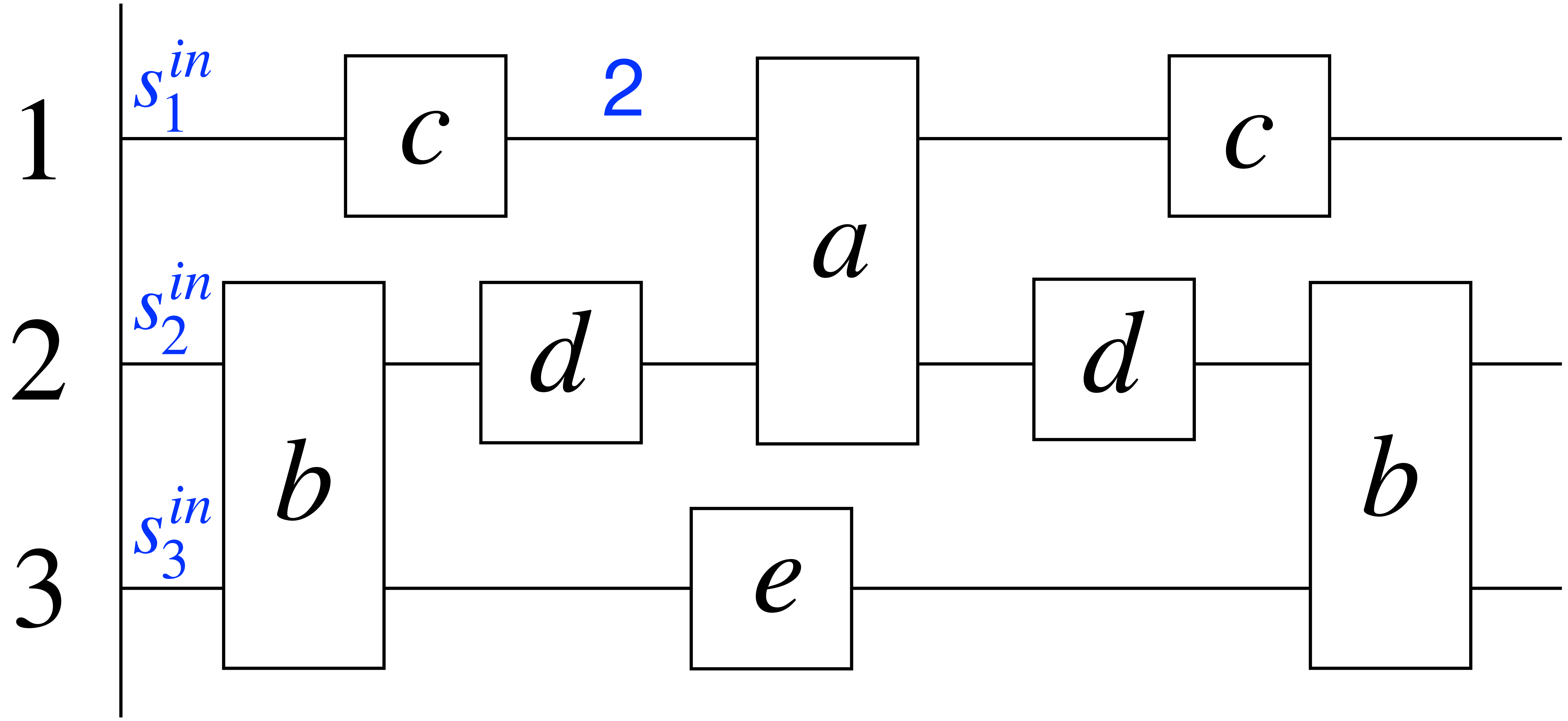
Asynchronous automata (Zielonka)



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

- 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

Asynchronous automata (Zielonka)

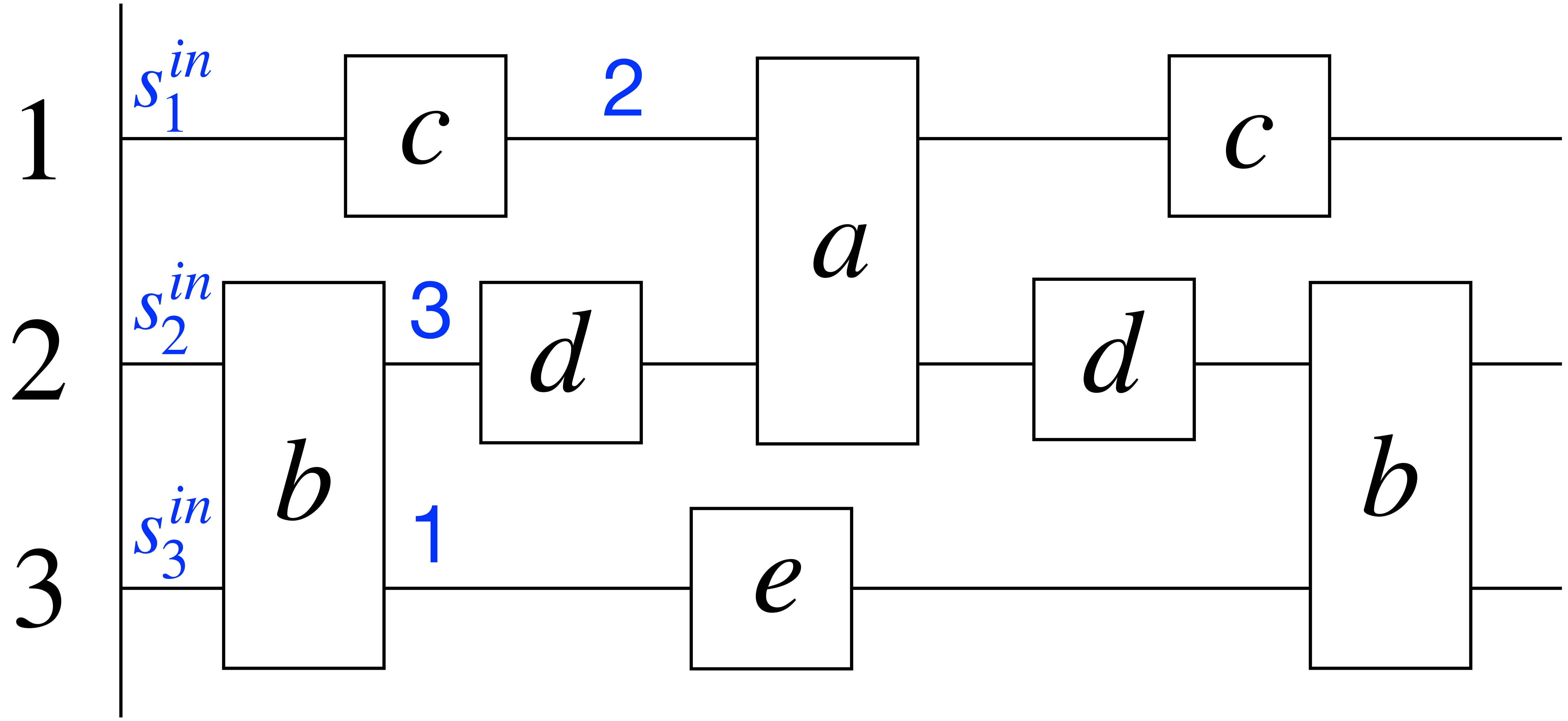


$$\mathscr{A} = \left(\{S_i\}_{i\in\mathscr{P}}, \{\delta_a\}_{a\in\Sigma}, s^{in}, F\right)$$

- 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

$$\delta_c: S_1 \to S_1$$

Asynchronous automata (Zielonka)



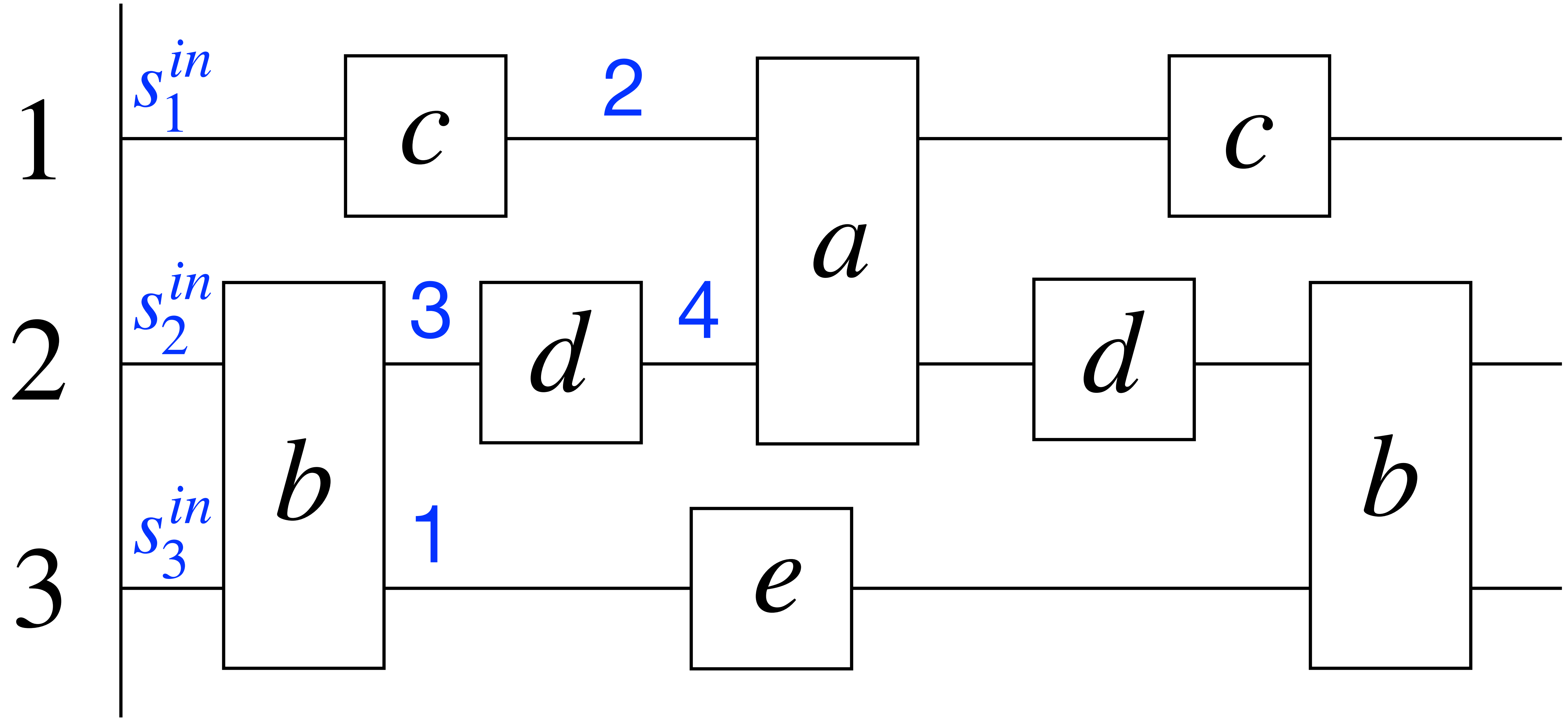
$\mathscr{A} = \left(\{S_i\}_{i\in\mathscr{P}}, \{\delta_a\}_{a\in\Sigma}, s^{in}, F\right)$

- 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

$\delta_c : S_1 \to S_1$

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

Asynchronous automata (Zielonka)



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

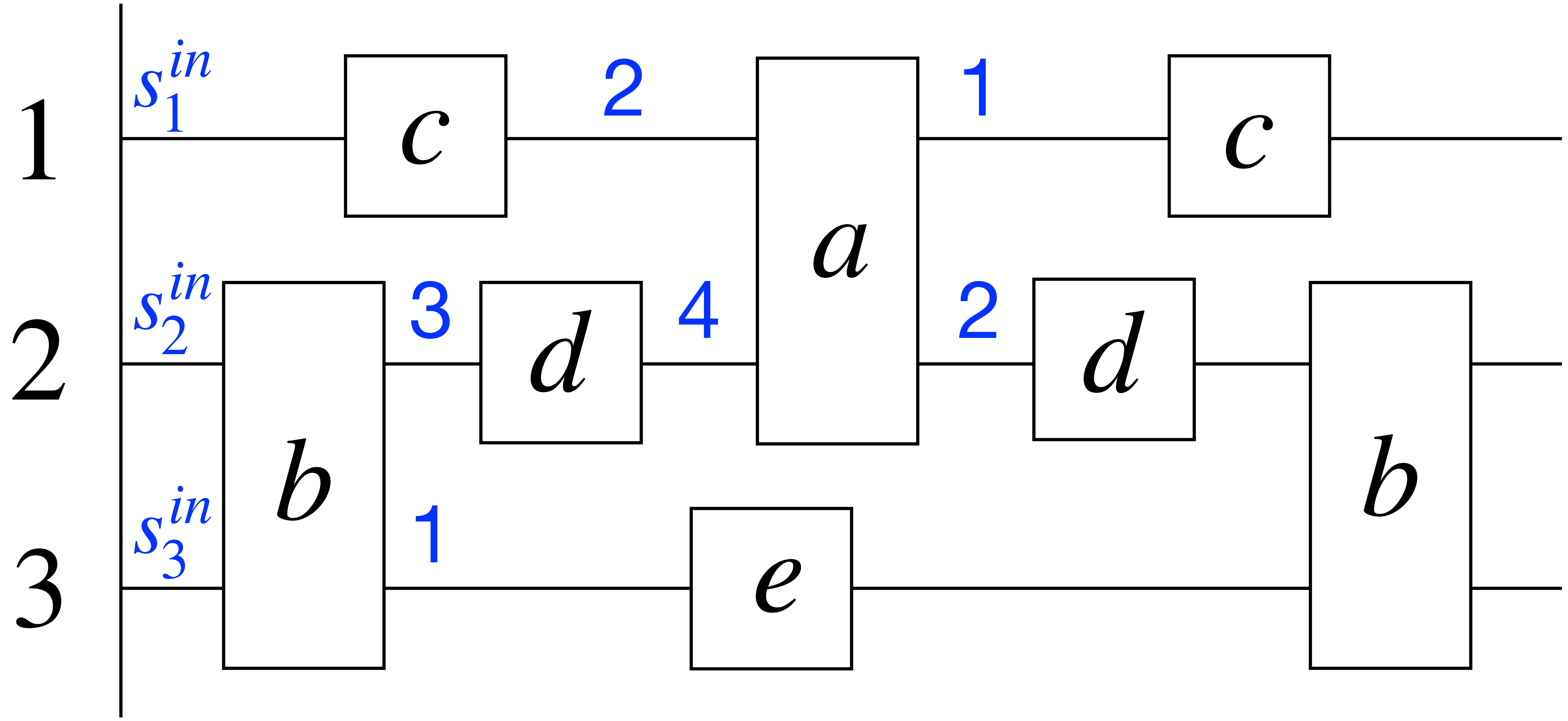
- S_i local states for process i
- δ_a transition function for action a
- $s^{in} = (s_1^{in}, s_2^{in}, s_3^{in})$ global initial state
- F global accepting states

$\delta_c \colon S_1 \to S_1$

$\delta_b \colon S_2 \times S_3 \to S_2 \times S_3$

$\delta_d \colon S_2 \to S_2$

Asynchronous automata (Zielonka)



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

- 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

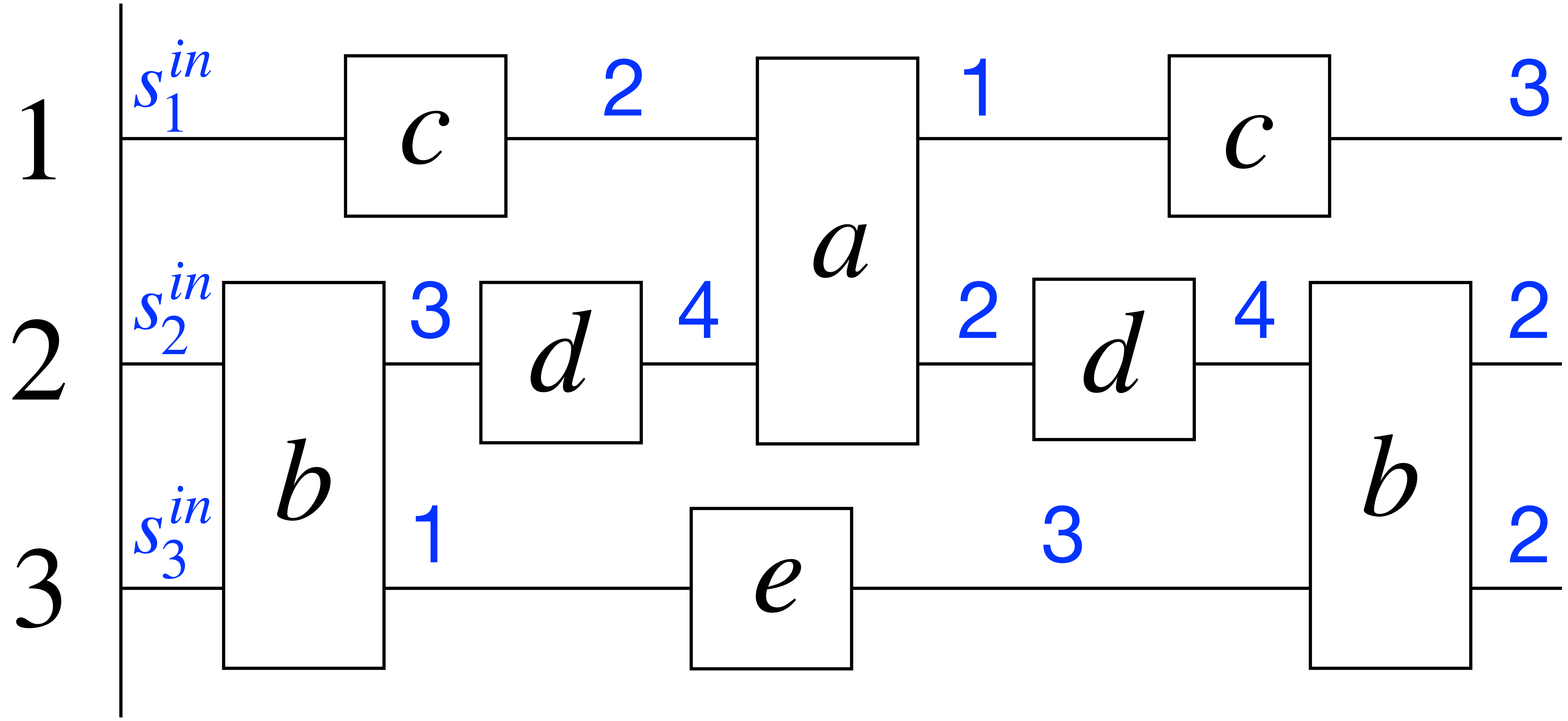
$$\delta_c : S_1 \to S_1$$

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

$$\delta_d : S_2 \to S_2$$

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

Asynchronous automata (Zielonka)



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

- 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

$\delta_c : S_1 \to S_1$

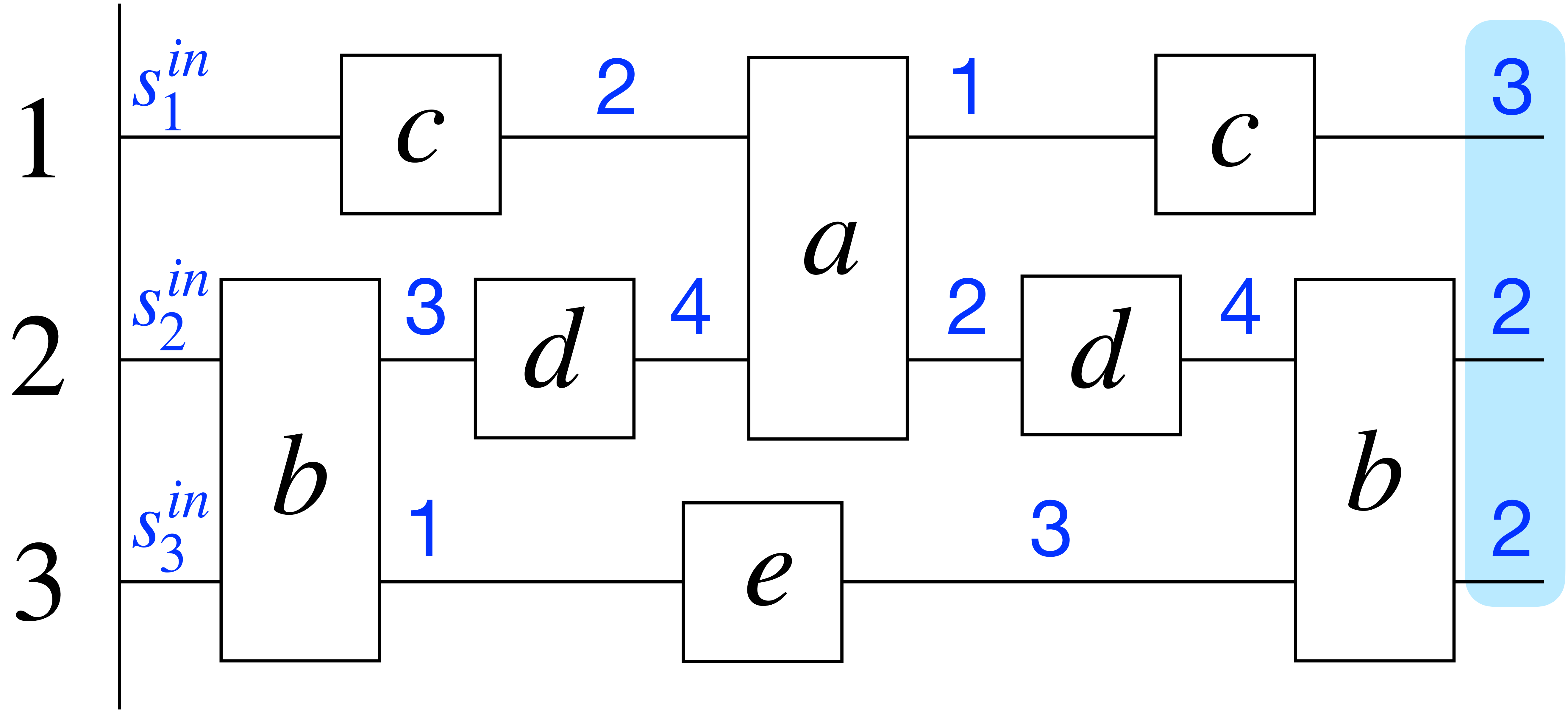
$\delta_b : S_2 \times S_3 \to S_2 \times S_3$

$\delta_d : S_2 \to S_2$

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

$\delta_e : S_3 \to S_3$

Asynchronous automata (Zielonka)



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

- 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

$\delta_c : S_1 \to S_1$

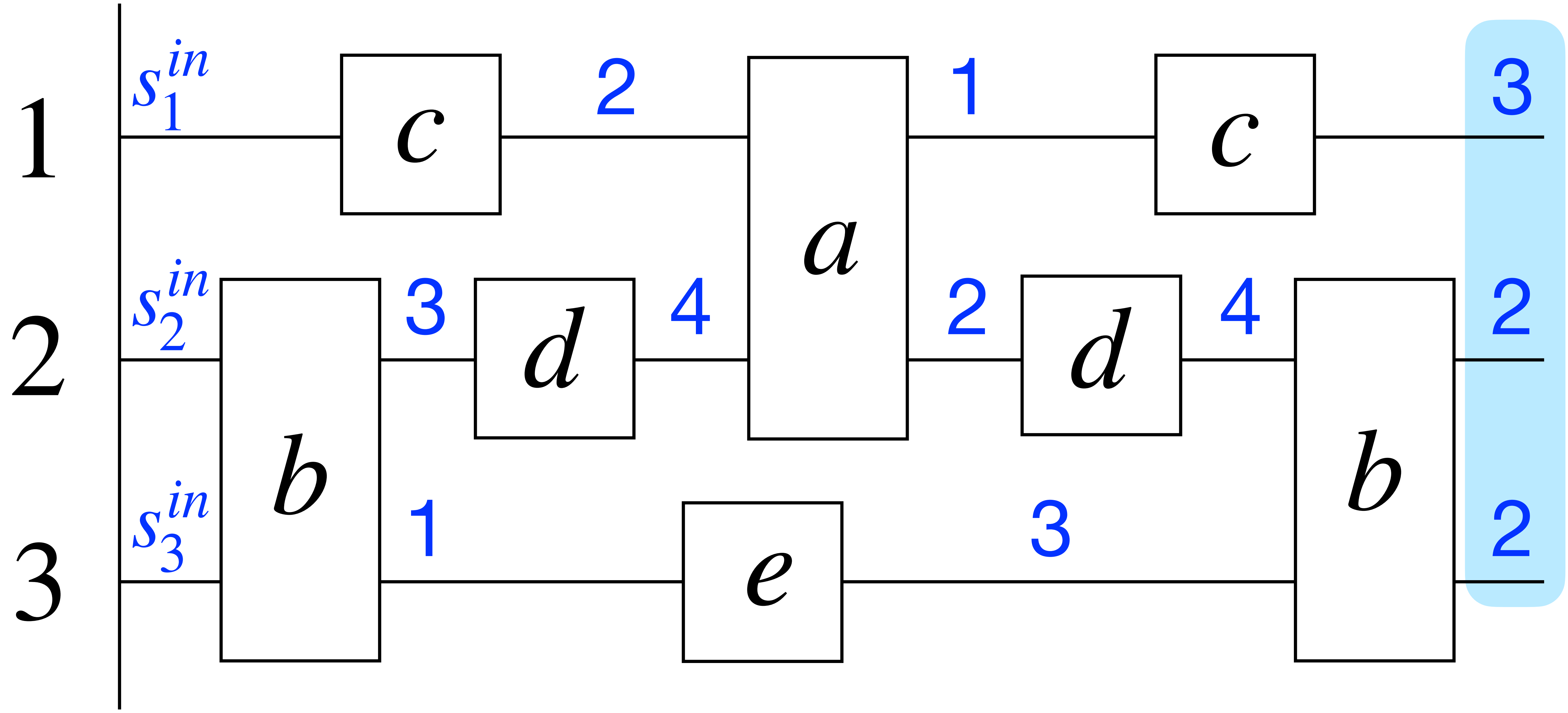
$\delta_b : S_2 \times S_3 \to S_2 \times S_3$

$\delta_d : S_2 \to S_2$

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

$\delta_e : S_3 \to S_3$

Asynchronous automata (Zielonka)



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

- 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

$\delta_c : S_1 \to S_1$

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

$\delta_d : S_2 \to S_2$

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

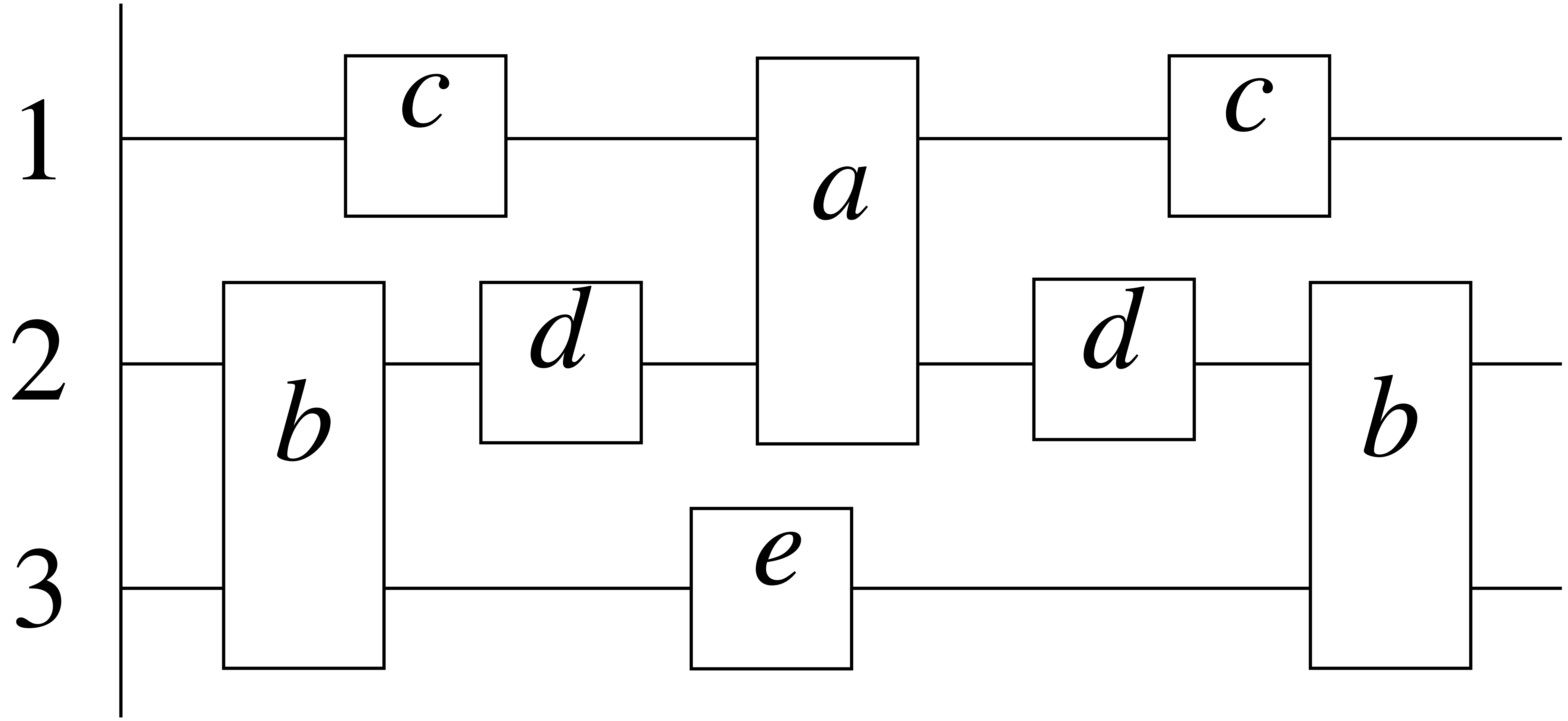
$\delta_e : S_3 \to S_3$

Theorem (Zielonka, 1987)

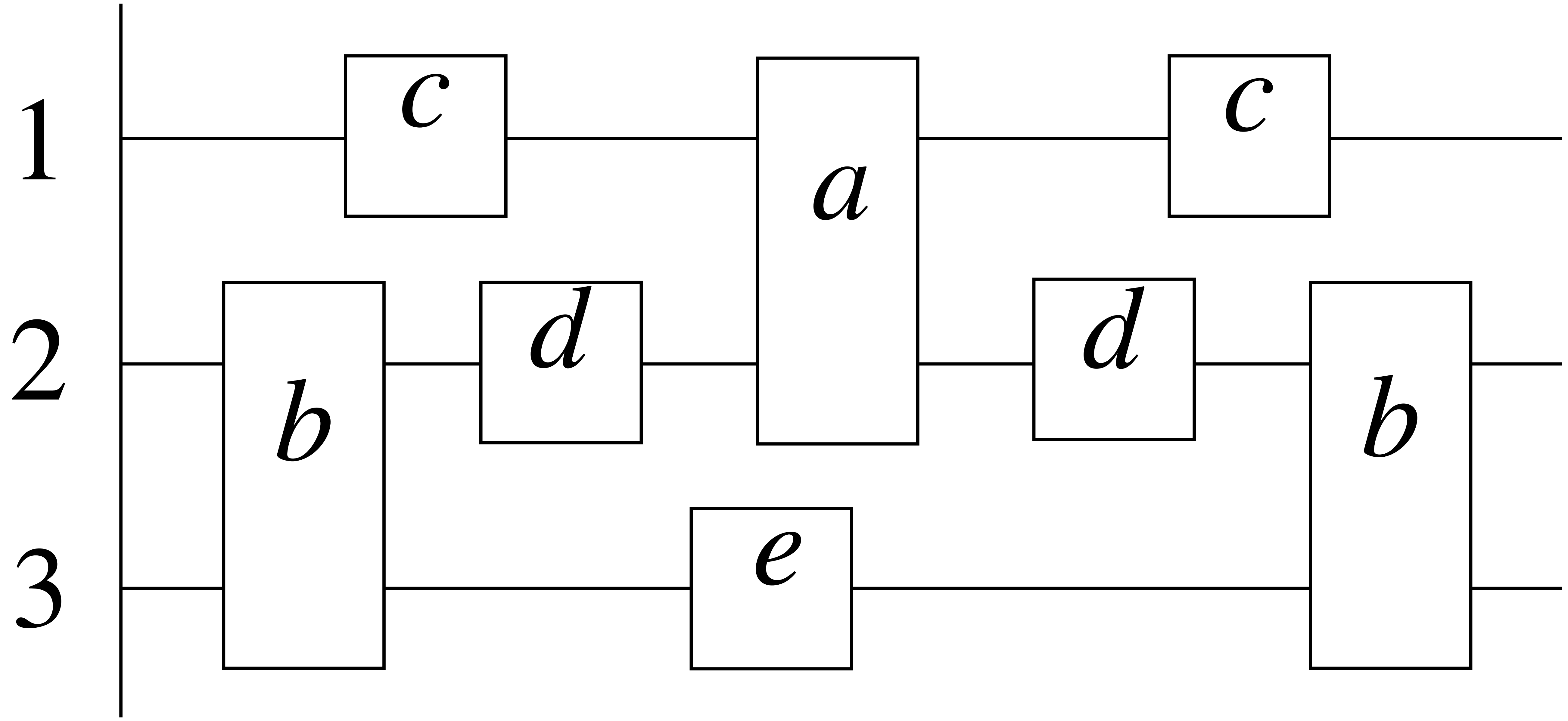
Asynchronous Automata = Regular Trace Languages

Labeling function

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



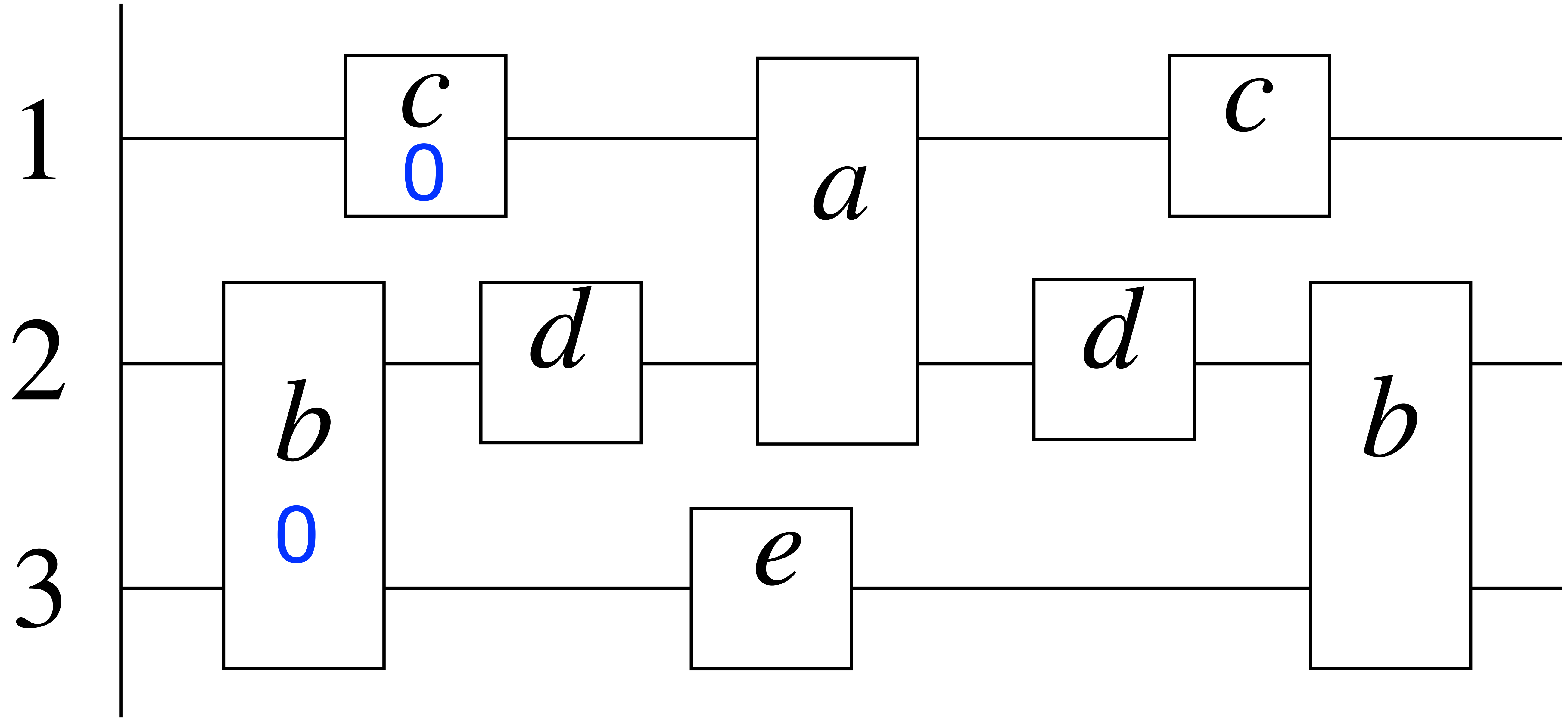
Labeling function



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

$\Gamma = \{0,1\}$ and $\mathrm{Y}_3 \leq \mathrm{Y}_2$

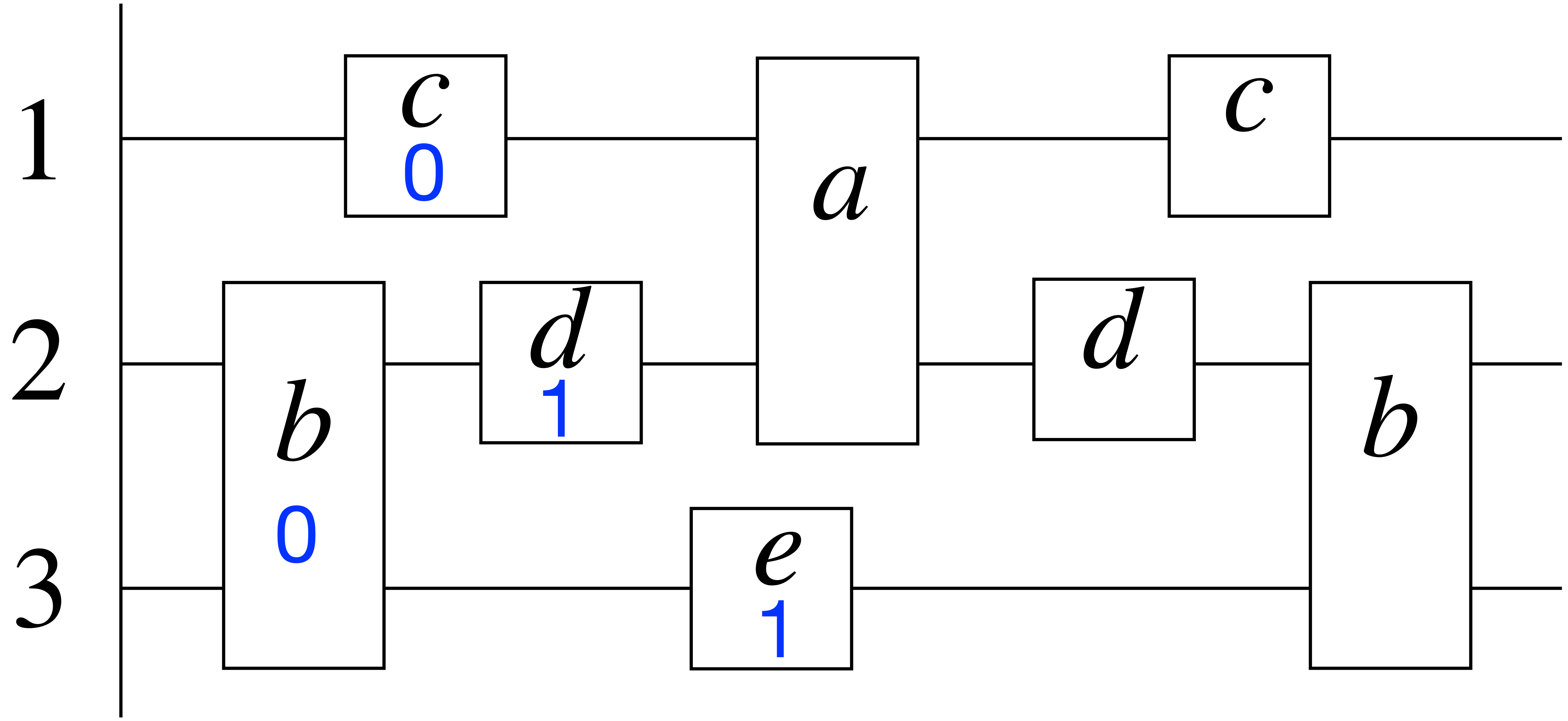
Labeling function



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

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

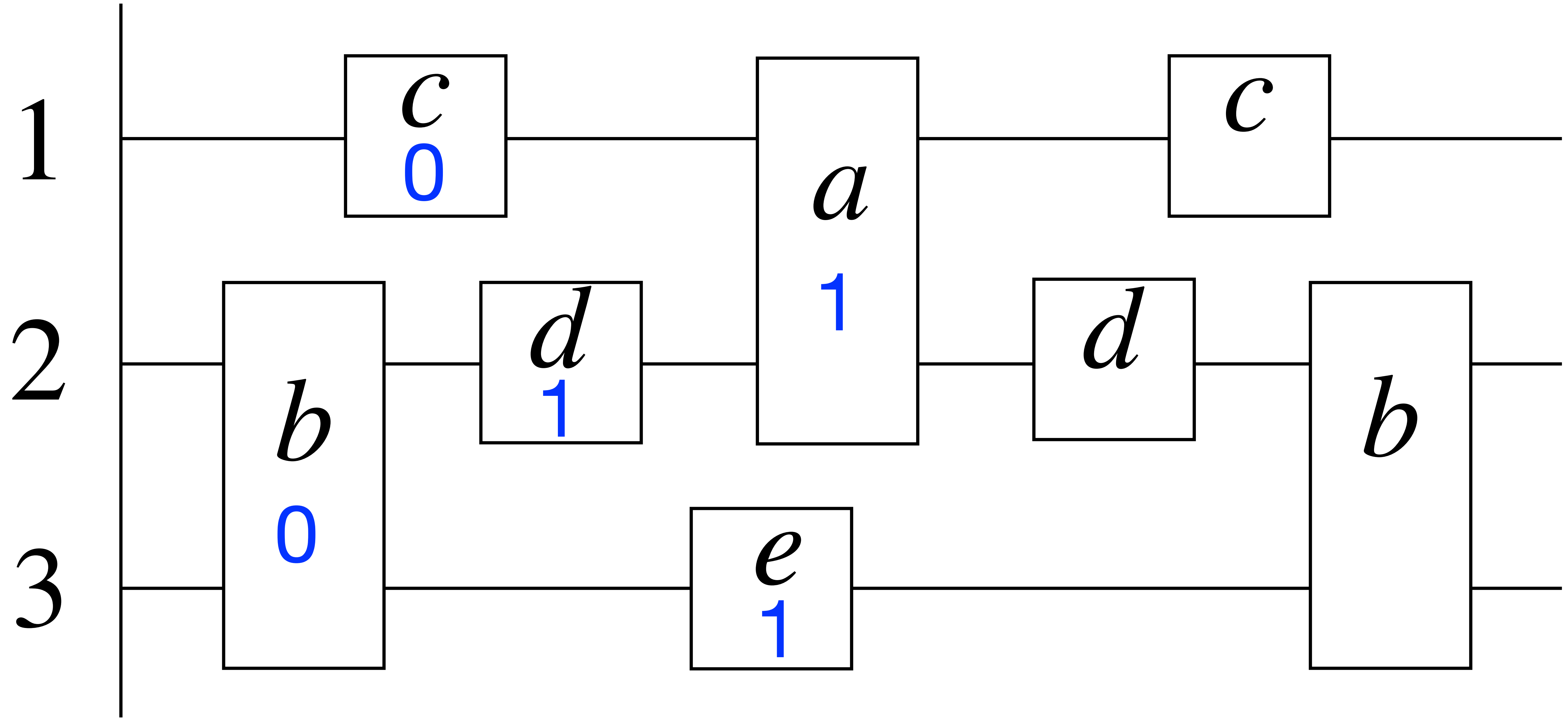
Labeling function



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

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

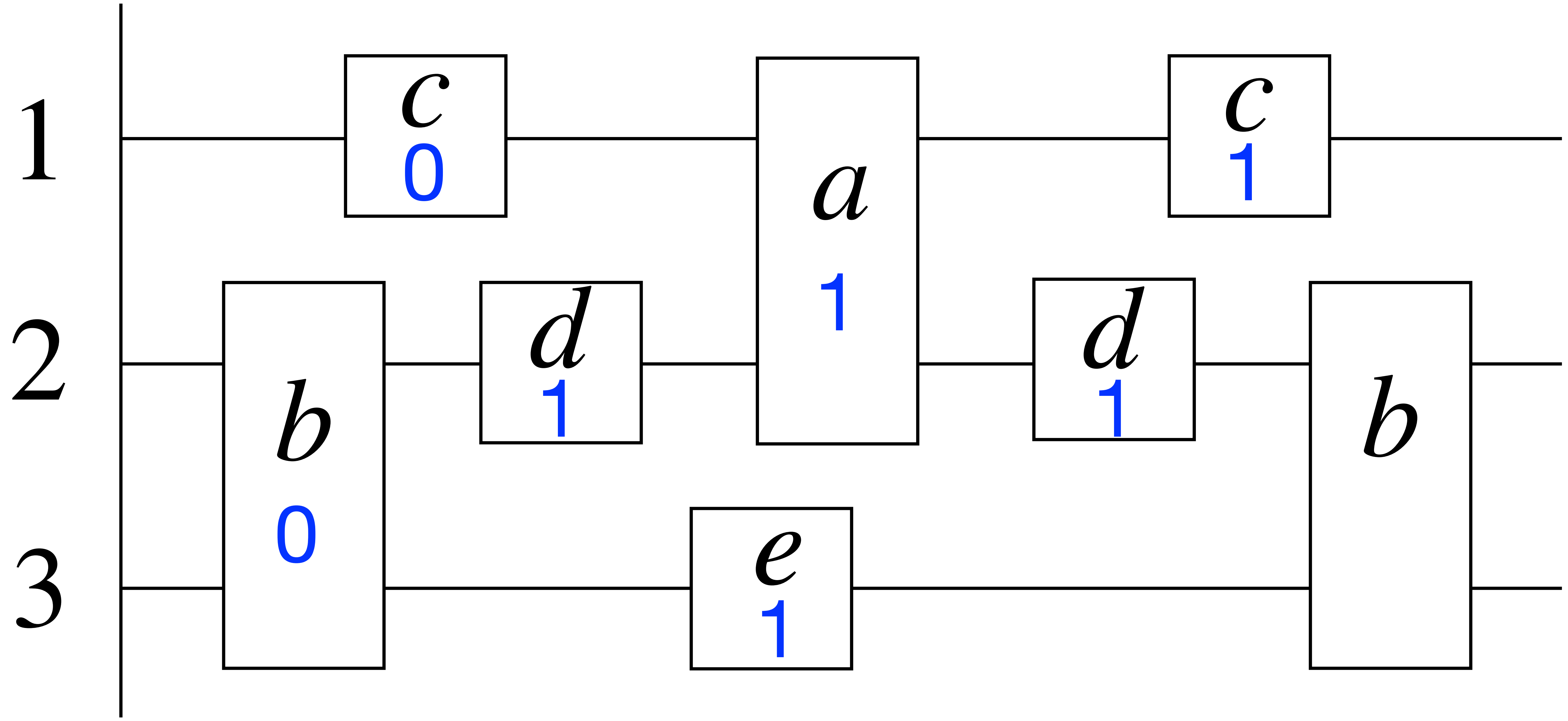
Labeling function



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

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

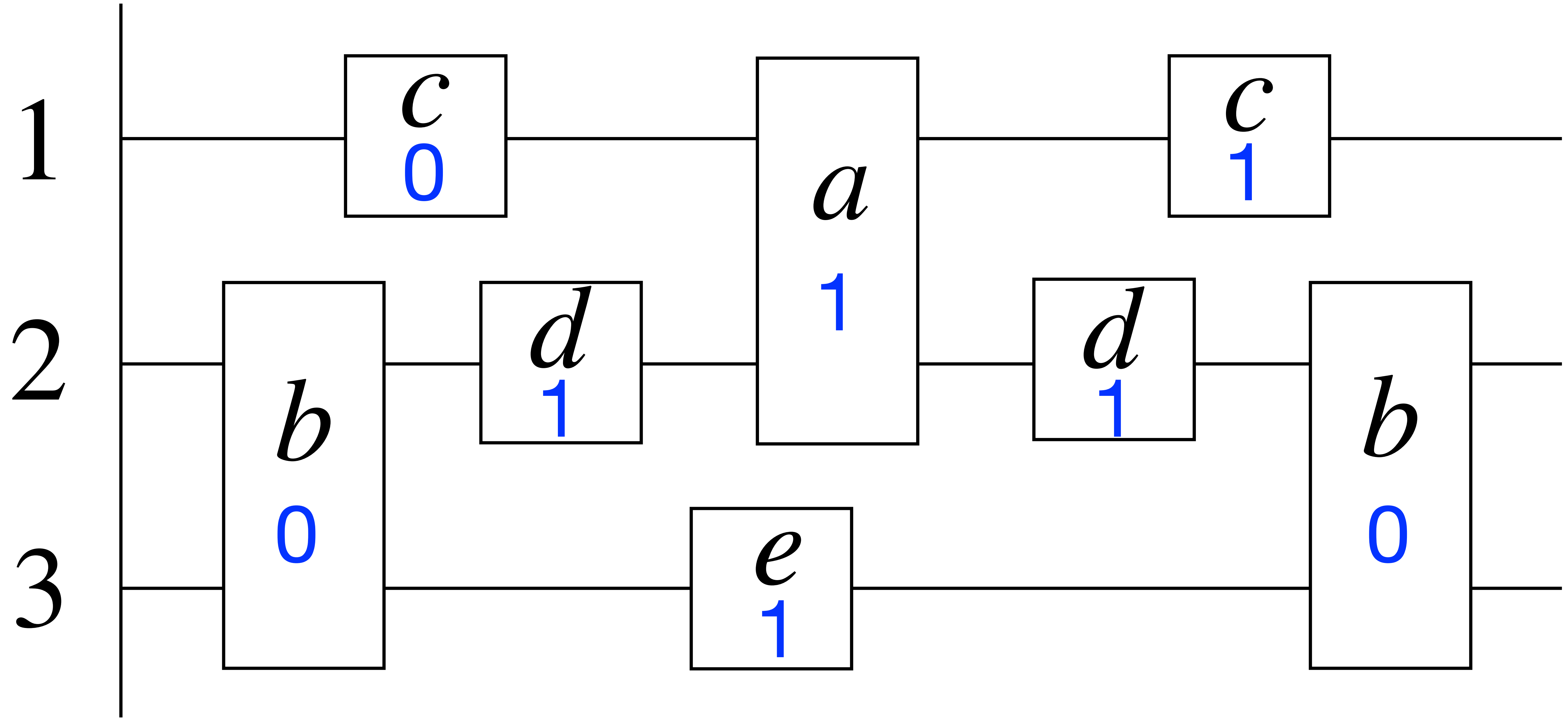
Labeling function



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

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

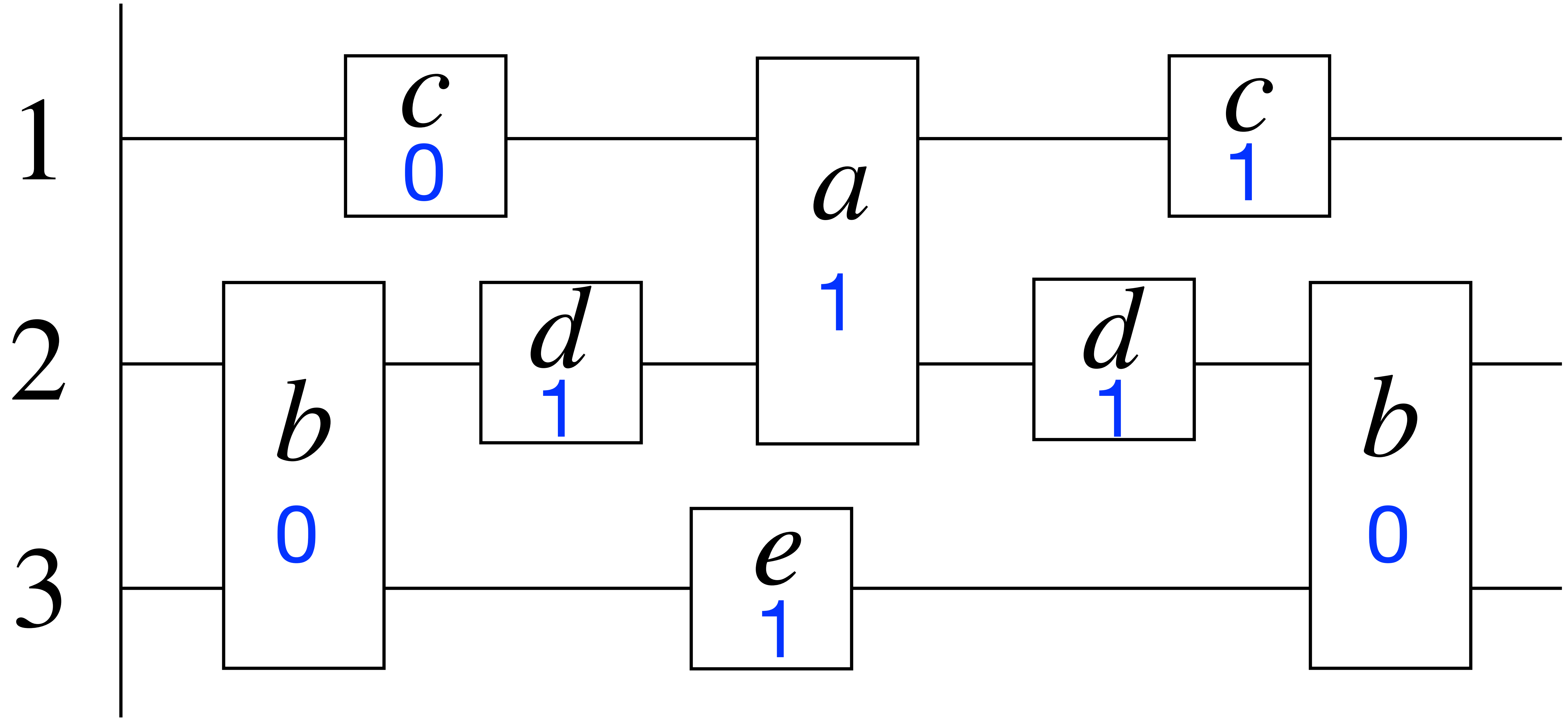
Labeling function



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

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

Labeling function

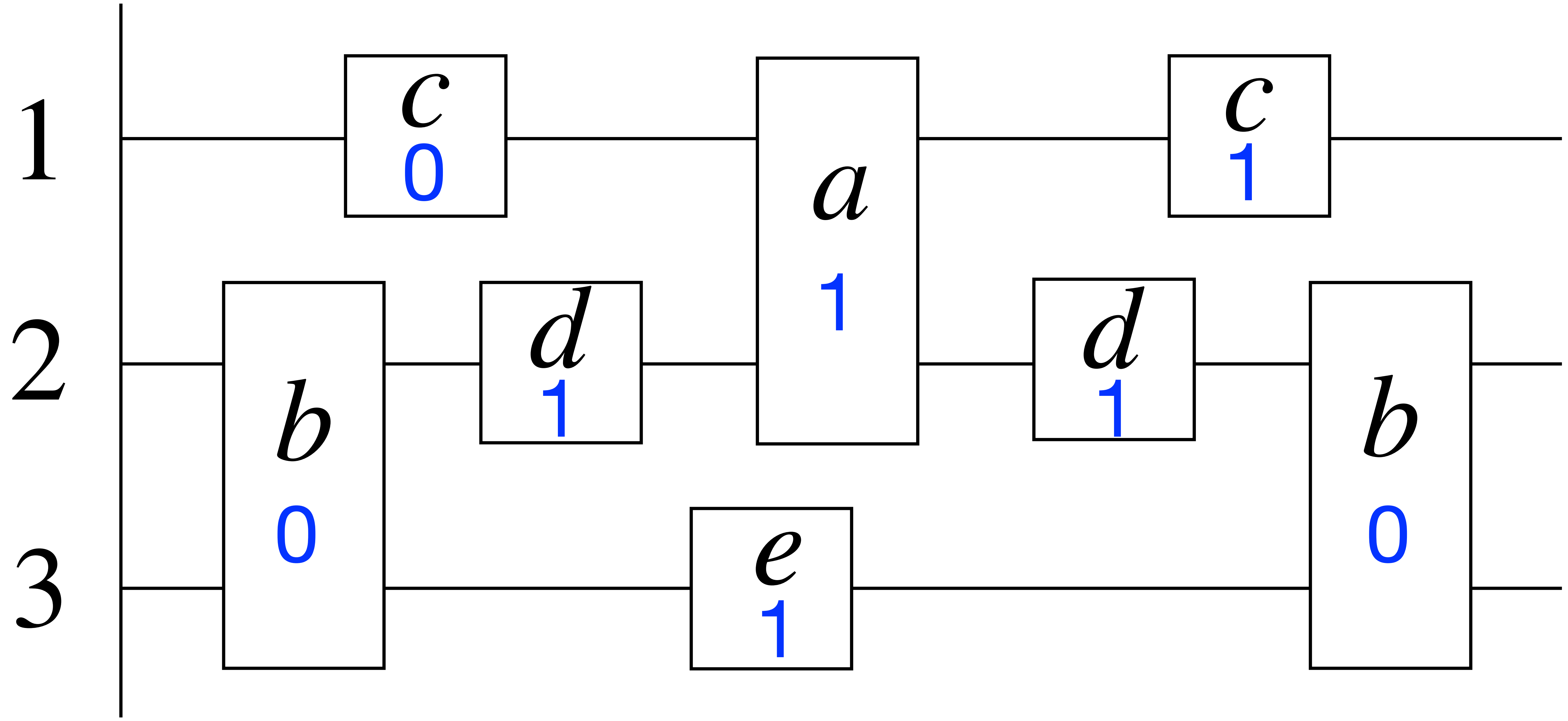


$\theta\colon Tr(\Sigma) \to Tr(\Sigma \times \Gamma)$

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

- $\mathscr{F}$ set of event formulas
- $\theta^{\mathscr{F}} : Tr(\Sigma) \to Tr\left(\Sigma \times \{0,1\}^{\mathscr{F}}\right)$

Asynchronous Labeling function



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

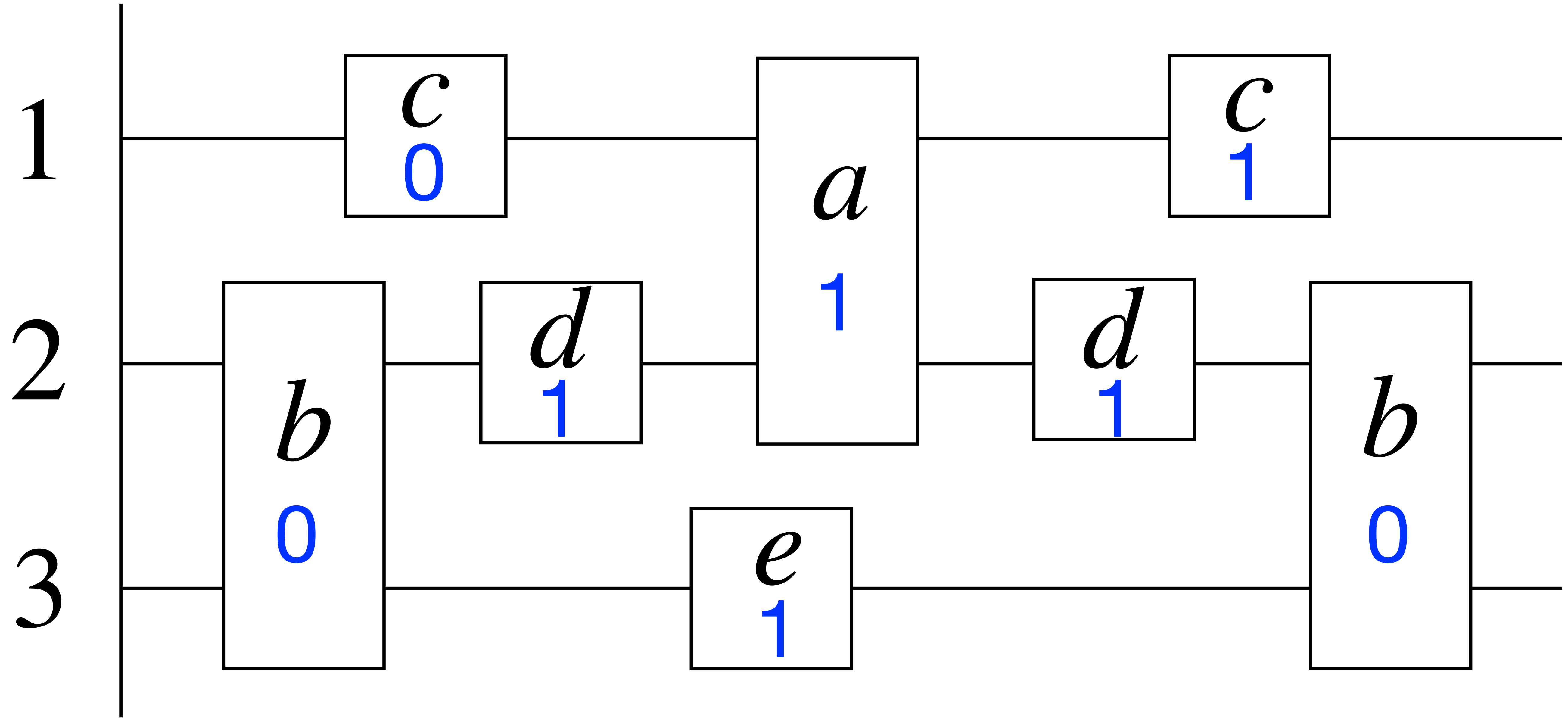
$\Gamma = \{0,1\}$ and $Y_3 \leq Y_2$

- $\mathscr{F}$ set of event formulas
- $\theta^{\mathscr{F}} : Tr(\Sigma) \to Tr\left(\Sigma \times \{0,1\}^{\mathscr{F}}\right)$

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

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

Asynchronous Labeling function



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

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

- $\mathscr{F}$ set of event formulas
- $\theta^{\mathscr{F}}\colon Tr(\Sigma) \to Tr\left(\Sigma \times \{0,1\}^{\mathscr{F}}\right)$

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

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

$\mu_c\colon S_1 \to \Gamma$

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

$\mu_d\colon S_2 \to \Gamma$

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

$\mu_e\colon S_3 \to \Gamma$

Gossip (Asynchronous) Transducer

Theorem (Mukund-Sohoni 1997)

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

There is an asynchronous letter-to-letter transducer $\mathscr{G}$ which computes

$$\theta^{\mathscr{Y}} : Tr(\Sigma) \to Tr\left(\Sigma \times \{0,1\}^{\mathscr{Y}}\right)$$

Outline

- ✅ Model of Concurrency: Mazurkiewicz Traces
- ✅ A propositional dynamic logic for traces
- ✅ Asynchronous automata (Zielonka) and asynchronous labeling functions
- Cascade product / decomposition (Krohn-Rhodes)
- Conclusion

Composition - Cascade product

Composition - Cascade product

- Composition of labelling functions

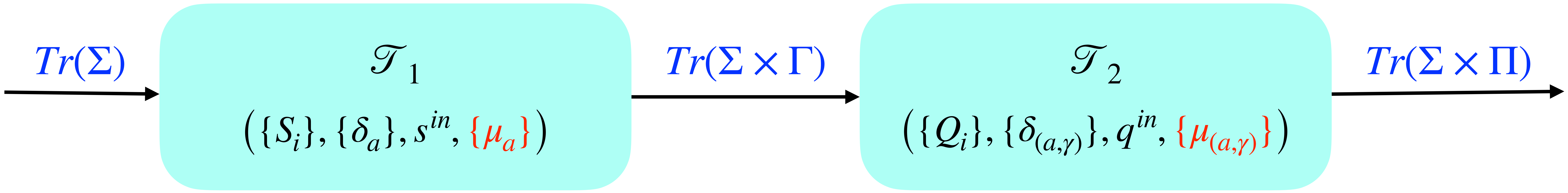
$$Tr(\Sigma) \xrightarrow{\theta_1} Tr(\Sigma \times \Gamma) \xrightarrow{\theta_2} Tr(\Sigma \times \Pi)$$

Composition - Cascade product

- Composition of labelling functions

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

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

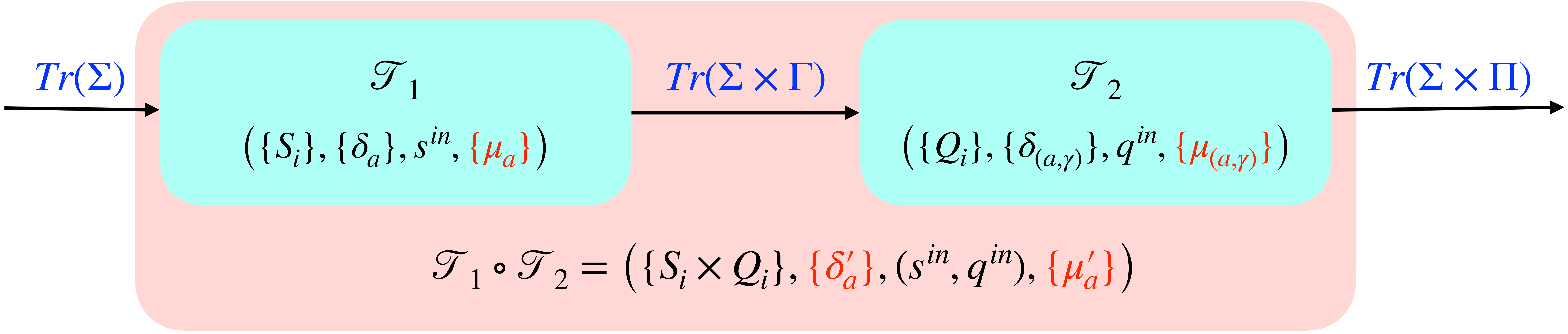


Composition - Cascade product

- Composition of labelling functions

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

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



Composition - Cascade product

- Composition of labelling functions

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

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

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

$$Tr(\Sigma) \longrightarrow \mathcal{T}_1 \; \left(\{S_i\}, \{\delta_a\}, s^{in}, \{\mu_a\}\right) \xrightarrow{Tr(\Sigma \times \Gamma)} \mathcal{T}_2 \; \left(\{Q_i\}, \{\delta_{(a,\gamma)}\}, q^{in}, \{\mu_{(a,\gamma)}\}\right) \longrightarrow Tr(\Sigma \times \Pi)$$

$$\mathcal{T}_1 \circ \mathcal{T}_2 = \left(\{S_i \times Q_i\}, \{\delta'_a\}, (s^{in}, q^{in}), \{\mu'_a\}\right)$$

Cascade Decomposition

Theorem 2

Any regular trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local asynchronous transducers:

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

Cascade Decomposition

Theorem 2

Any regular trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local asynchronous transducers:

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

$\mathcal{T} = \left(\{S_i\}_{i \in \mathcal{P}}, \{\delta_a\}, s^{in}, \{\mu_a\}\right)$ is k-local if $|S_i| = 1$ for all $i \neq k$

Cascade Decomposition

Theorem 2

Any regular trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local asynchronous transducers:

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

$\mathcal{T} = \left(\{S_i\}_{i \in \mathcal{P}}, \{\delta_a\}, s^{in}, \{\mu_a\}\right)$ is k-local if $|S_i| = 1$ for all $i \neq k$

Proof sketch:

- Regular = locPastPDL (Theorem 1)

Cascade Decomposition

Theorem 2

Any regular trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local asynchronous transducers:

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

$\mathcal{T} = \left(\{S_i\}_{i \in \mathcal{P}}, \{\delta_a\}, s^{in}, \{\mu_a\}\right)$ is k-local if $|S_i| = 1$ for all $i \neq k$

Proof sketch:

- Regular = locPastPDL (Theorem 1)
- For each event formula φ we construct by structural induction such a cascade product computing θ^{φ}

$$\varphi ::= a \mid \varphi \vee \varphi \mid \neg \varphi \mid \langle \pi \rangle \varphi \mid \mathsf{Y}_i \leq \mathsf{Y}_j \mid \mathsf{Y}_{i,k} \leq \mathsf{Y}_j$$

Cascade Decomposition

Theorem 2

Any regular trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local asynchronous transducers:

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

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

Proof sketch:

- Regular = locPastPDL (Theorem 1)
- For each event formula φ we construct by structural induction such a cascade product computing θ^φ

$$\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi \mid \mathsf{Y}_i \leq \mathsf{Y}_j \mid \mathsf{Y}_{i,k} \leq \mathsf{Y}_j$$

  - The gossip transducer $\mathscr{G}$ computes $\theta^{\mathscr{Y}}$ with $\mathscr{Y} = \{\mathsf{Y}_i \leq \mathsf{Y}_j\ , \mathsf{Y}_{i,k} \leq \mathsf{Y}_j \mid i, j, k \in \mathscr{P}\}$

Cascade Decomposition

Theorem 2

Any regular trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local asynchronous transducers:

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

$\mathcal{T} = \left(\{S_i\}_{i\in\mathcal{P}}, \{\delta_a\}, s^{in}, \{\mu_a\}\right)$ is k-local if $|S_i| = 1$ for all $i \neq k$

Proof sketch:

- Regular = locPastPDL (Theorem 1)
- For each event formula φ we construct by structural induction such a cascade product computing θ^φ

$$\varphi ::= a \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\pi\rangle\varphi \mid \mathsf{Y}_i \leq \mathsf{Y}_j \mid \mathsf{Y}_{i,k} \leq \mathsf{Y}_j$$

  - The gossip transducer $\mathcal{G}$ computes $\theta^{\mathcal{Y}}$ with $\mathcal{Y} = \{\mathsf{Y}_i \leq \mathsf{Y}_j, \mathsf{Y}_{i,k} \leq \mathsf{Y}_j \mid i,j,k \in \mathcal{P}\}$
  - Constants and boolean connectives are trivial
  - Local past Path expressions $\langle\pi\rangle\,\varphi$ are (rather) easy

Cascade Decomposition

Theorem 2

Any regular trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local asynchronous transducers:

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

Corollary: Zielonka's theorem

Asynchronous Automata = Regular Trace Languages

Cascade Decomposition

Theorem 2

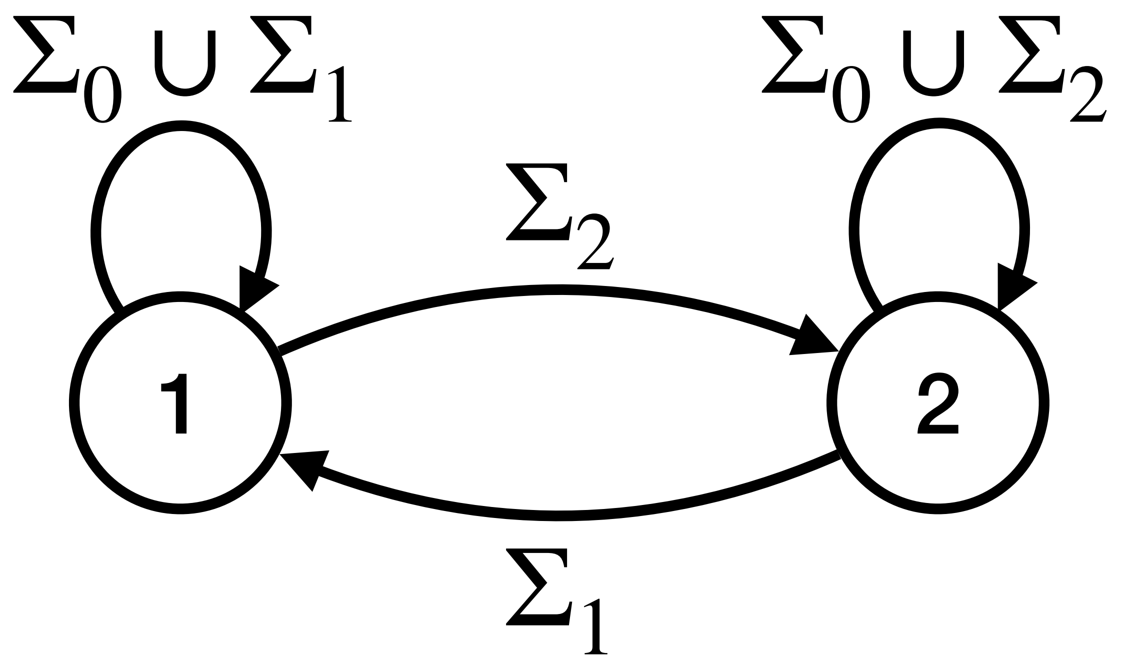
Any regular trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local asynchronous transducers:

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

Bonus: Using Krohn-Rhodes theorem

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

Reset - $\mathscr{U}_2$



Cascade Decomposition

Theorem 2

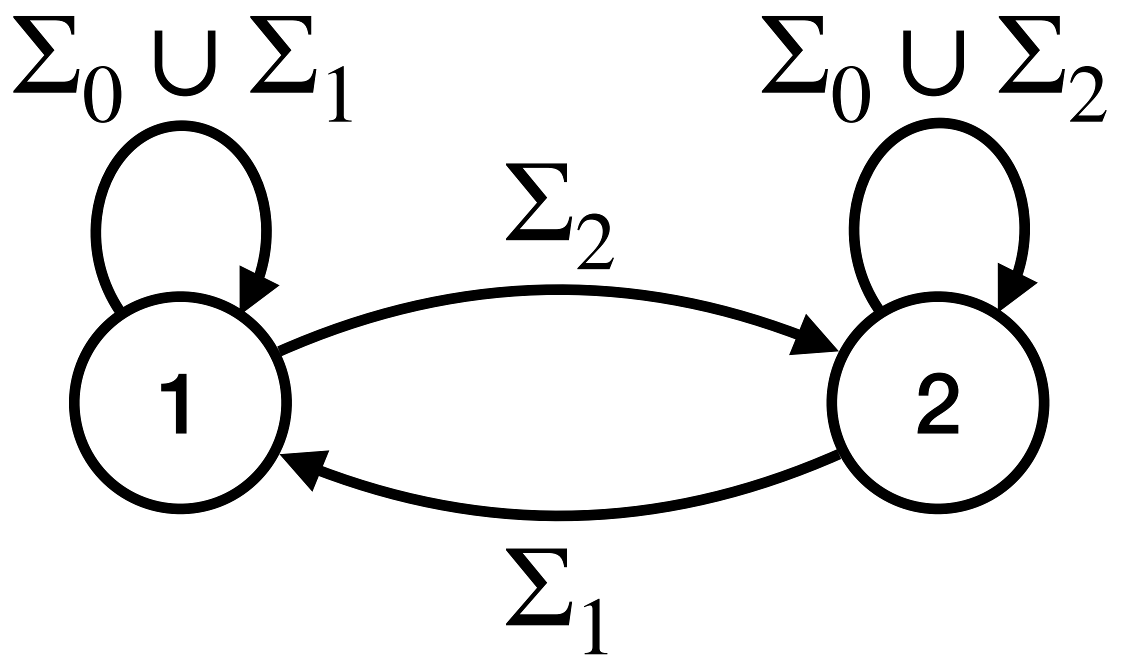
Any regular trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local asynchronous transducers:

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

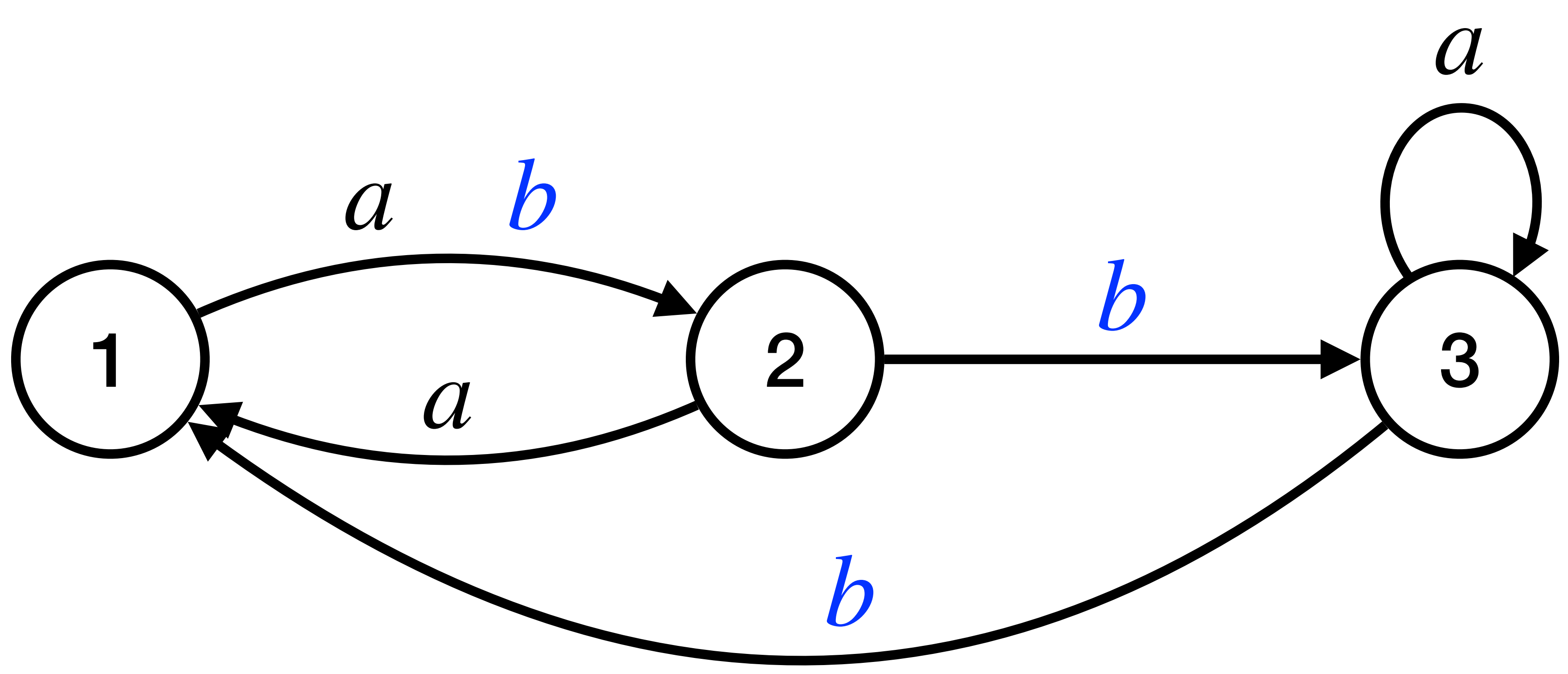
Bonus: Using Krohn-Rhodes theorem

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

Reset - $\mathscr{U}_2$



Permutation



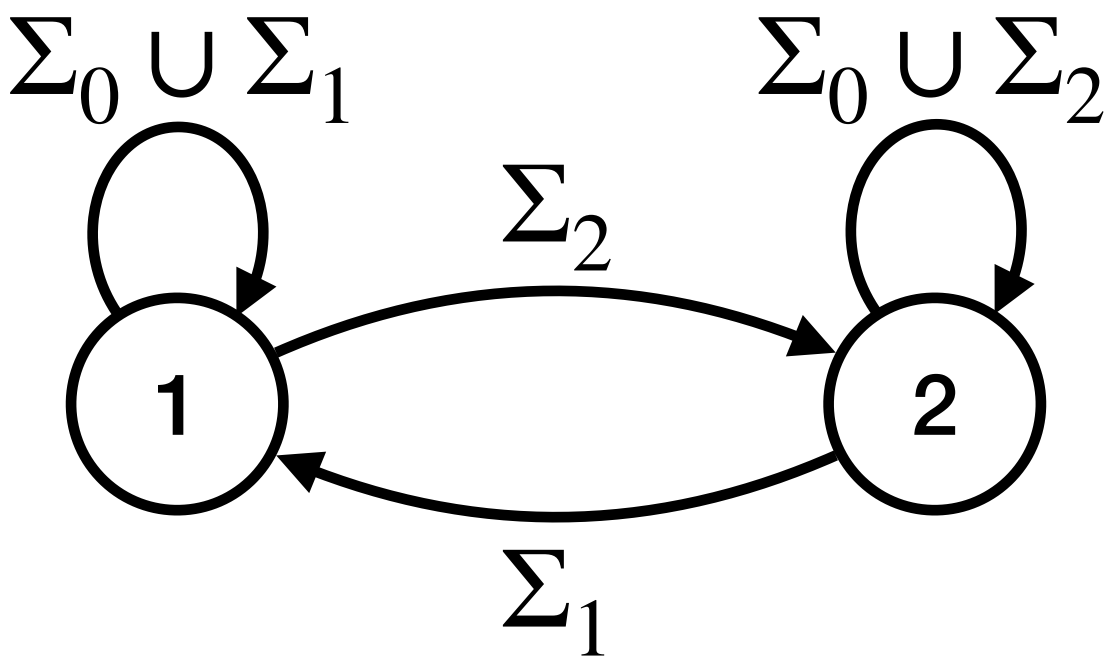
Aperiodic = FO-definable

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:

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

Reset - $\mathcal{U}_2$



Aperiodic = FO-definable

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

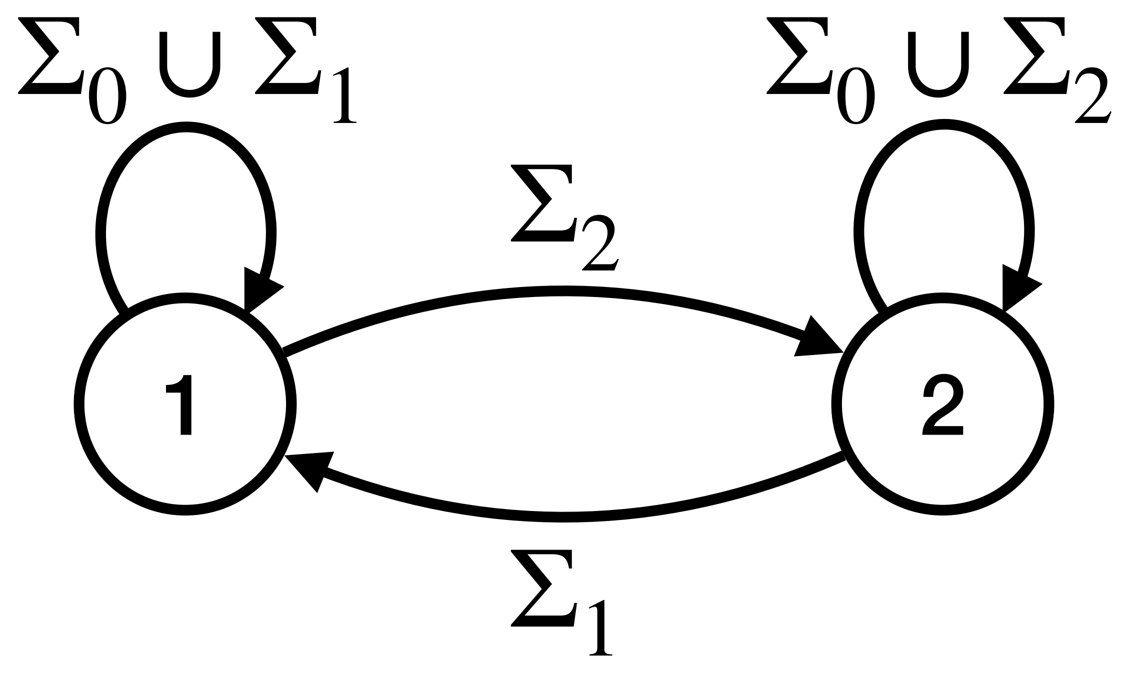
Any aperiodic (FO) trace language is accepted by a cascade product of the gossip transducer followed by a sequence of local reset transducers:

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

Direct proof (Not using Krohn-Rhodes theorem)

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

Reset - $\mathscr{U}_2$



Outline

- ✅ Model of Concurrency: Mazurkiewicz Traces
- ✅ A propositional dynamic logic for traces
- ✅ Asynchronous automata (Zielonka) and asynchronous labeling functions
- ✅ Cascade product / decomposition (Krohn-Rhodes)
- Conclusion

Conclusion

Conclusion

Main results

- (Local) (past) Propositional dynamic logic expressively complete for regular languages
  - Specification language: natural, easy, expressive, good complexity

Conclusion

Main results

- (Local) (past) Propositional dynamic logic expressively complete for regular languages
  - Specification language: natural, easy, expressive, good complexity
- Cascade decomposition using simple & local asynchronous automata/transducers
  - Allows inductive reasoning on automata

Conclusion

Main results

- (Local) (past) Propositional dynamic logic expressively complete for regular languages
  - Specification language: natural, easy, expressive, good complexity
- Cascade decomposition using simple & local asynchronous automata/transducers
  - Allows inductive reasoning on automata

Open problems

- Expressivity of locPastPDL without some constants $Y_{i,k} \leq Y_j$, $Y_i \leq Y_j$, $L_{i,k} \leq L_j$, $L_i \leq L_j$?

Conclusion

Main results

- (Local) (past) Propositional dynamic logic expressively complete for regular languages
  - Specification language: natural, easy, expressive, good complexity
- Cascade decomposition using simple & local asynchronous automata/transducers
  - Allows inductive reasoning on automata

Open problems

- Expressivity of locPastPDL without some constants $Y_{i,k} \leq Y_j$, $Y_i \leq Y_j$, $L_{i,k} \leq L_j$, $L_i \leq L_j$?
- Can we get rid of the gossip transducer in the cascade decomposition?
  - Yes if the architecture (communication graph) is acyclic! (CONCUR'20, LMCS'22)

Conclusion

Main results

- (Local) (past) Propositional dynamic logic expressively complete for regular languages
  - ▷ Specification language: natural, easy, expressive, good complexity
- Cascade decomposition using simple & local asynchronous automata/transducers
  - ▷ Allows inductive reasoning on automata

Open problems

- Expressivity of locPastPDL without some constants $Y_{i,k} \leq Y_j$, $Y_i \leq Y_j$, $L_{i,k} \leq L_j$, $L_i \leq L_j$?
- Can we get rid of the gossip transducer in the cascade decomposition?
  - ▷ Yes if the architecture (communication graph) is acyclic! (CONCUR'20, LMCS'22)
- Can we decompose the gossip transducer in a cascade of local asynchronous transducers?

Conclusion

Main results

- (Local) (past) Propositional dynamic logic expressively complete for regular languages
  - ▷ Specification language: natural, easy, expressive, good complexity
- Cascade decomposition using simple & local asynchronous automata/transducers
  - ▷ Allows inductive reasoning on automata

Open problems

- Expressivity of locPastPDL without some constants $\mathsf{Y}_{i,k} \leq \mathsf{Y}_j$, $\mathsf{Y}_i \leq \mathsf{Y}_j$, $\mathsf{L}_{i,k} \leq \mathsf{L}_j$, $\mathsf{L}_i \leq \mathsf{L}_j$?
- Can we get rid of the gossip transducer in the cascade decomposition?
  - ▷ Yes if the architecture (communication graph) is acyclic! (CONCUR'20, LMCS'22)
- Can we decompose the gossip transducer in a cascade of local asynchronous transducers?
- Generalisation to Message Sequence Charts (Message passing automata)?

Thank you for your attention!