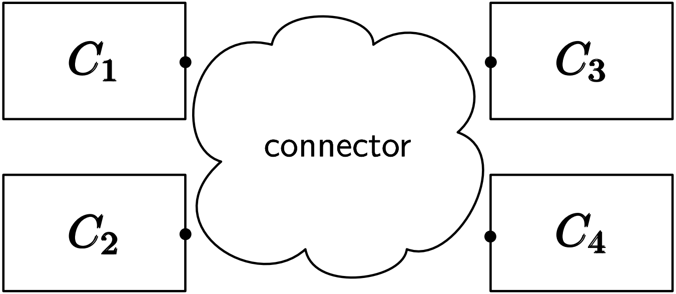
Modeling and Verification of Components and Connectors

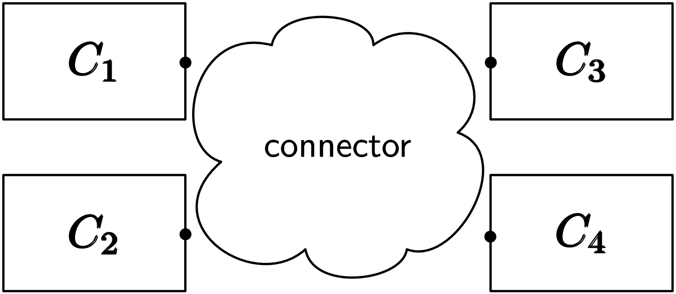
Christel Baier
Technische Universität Dresden

joint work with Tobias Blechmann
Joachim Klein
Sascha Klüppelholz



coordination provided by a component connector

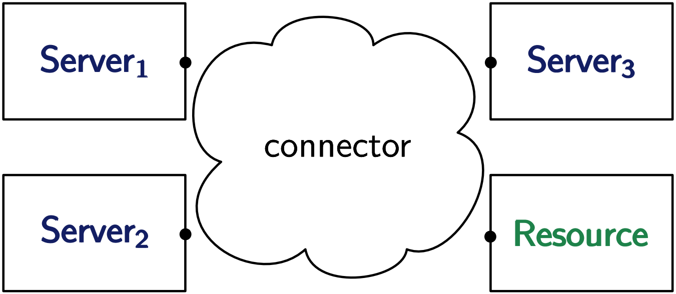
Components and connectors



coordination provided by a component connector

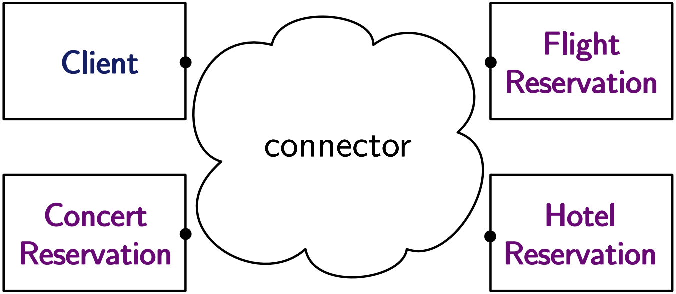
- glue clode: orchestrates the interactions of possibly heterogenous components
- synchronous and asynchronous communication, possibly data-dependent

Components and connectors



example: connector for orchestrating the interactions of multiple servers and shared resources

Components and connectors

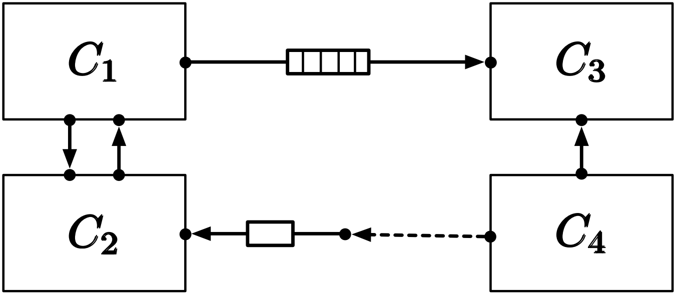


example: orchestrating multiple webservices:

"only book flight and hotel if booking of the concert ticket is successfull"

Components and connectors

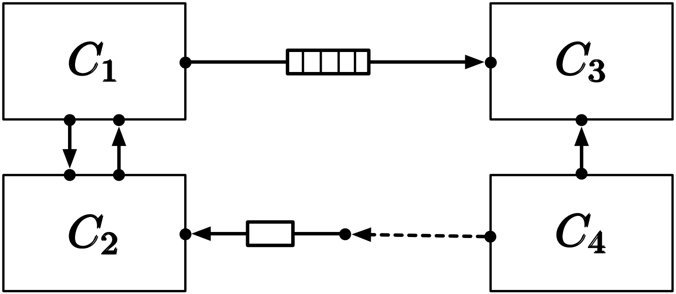
013



here: coordination arises from combination basic channels forming a network

Components and connectors

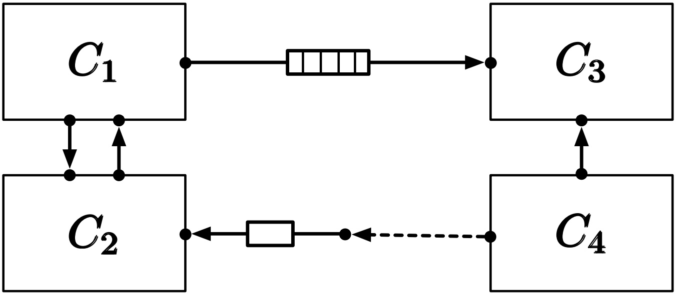
013



modeling approach for components and connectors, relying on

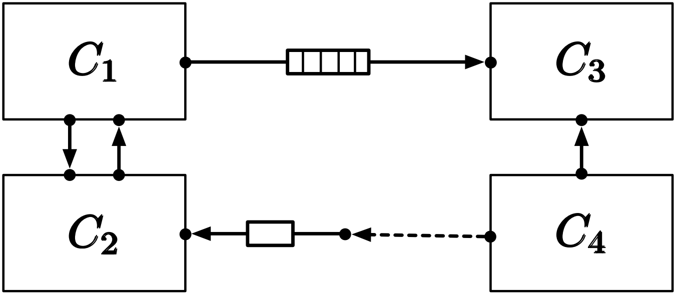
- calculus of channels
- operational LTS-like semantics

Components and connectors



modeling approach for components and connectors, relying on

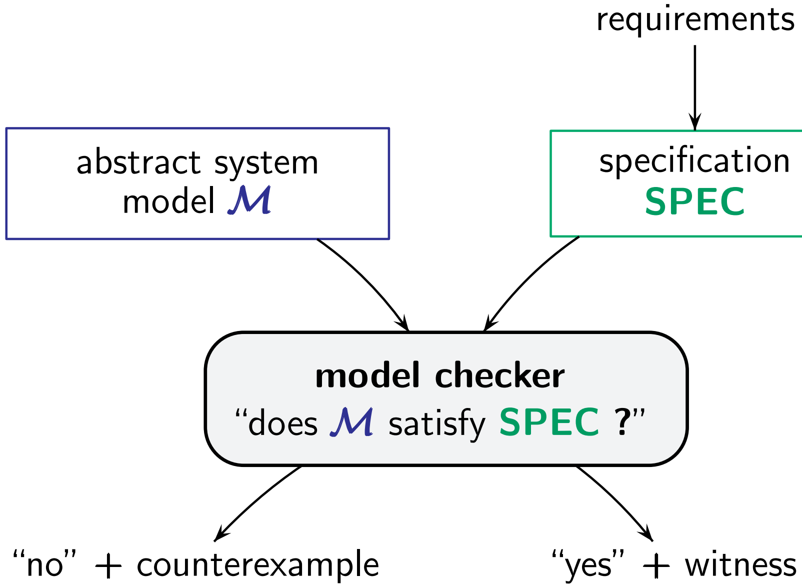
- calculus of channels $\longleftarrow$ $\boxed{\text{Reo}}$
- operational LTS-like semantics



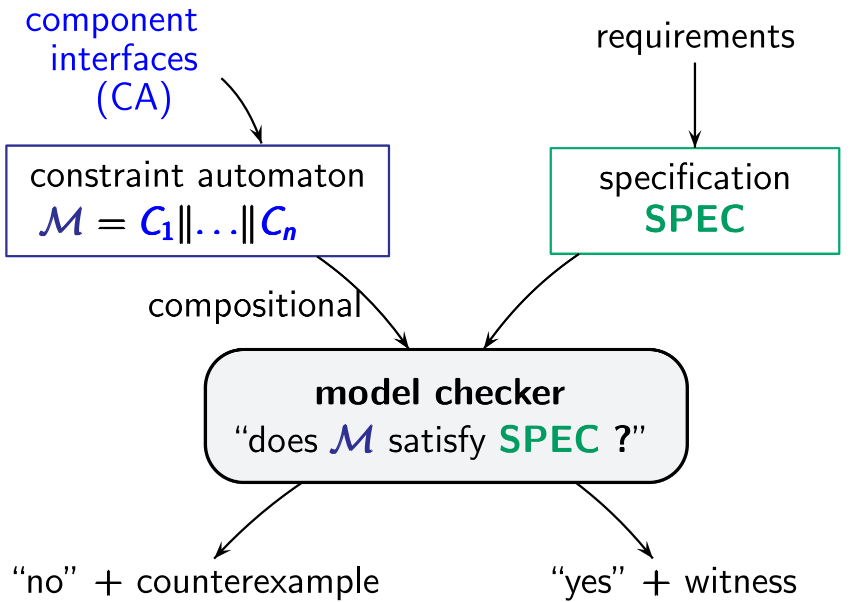
modeling approach for components and connectors, relying on

- calculus of channels ⟵ Reo
- operational LTS-like semantics ⟵ constraint automata

Model checking

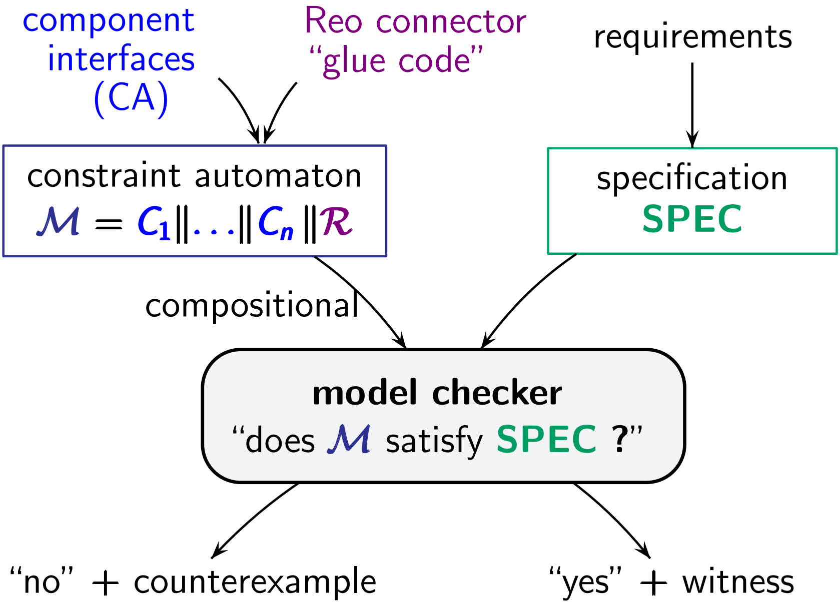


Model checking components and connectors



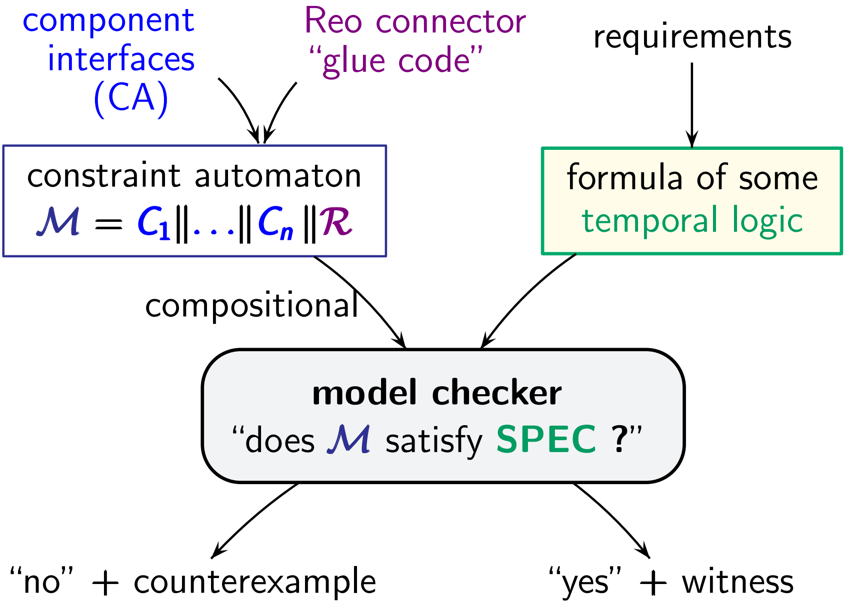
Model checking components and connectors

020



Model checking components and connectors

020



Overview

1. Modelling components and connectors
   - constraint automata (CA)
   - coordination language Reo
2. Model checking with CA
   - Linear Temporal Logic
   - Alternating Stream Logic
3. Synthesis of connectors

Overview

1. **Modelling components and connectors**
   - constraint automata (CA) $\longleftarrow$
   - coordination language Reo
2. Model checking with CA
   - Linear Temporal Logic
   - Alternating Stream Logic
3. Synthesis of connectors

Constraint automaton (CA)

an LTS-like automata model for

- component interfaces $C_1, \ldots, C_n$
- glue code, i.e., Reo network of channels $\mathcal{R}$
- composite system $C_1 \| \ldots \| C_n \| \mathcal{R}$

A constraint automaton is a tuple $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$

Constraint automaton (CA)

A constraint automaton is a tuple $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$

- Q is the state space

Constraint automaton (CA)

A constraint automaton is a tuple $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$

- Q is the state space
- $Q_0 \subseteq Q$ the set of initial states

Constraint automaton (CA)

A constraint automaton is a tuple $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$

- Q is the state space
- $Q_0 \subseteq Q$ the set of initial states
- $\mathcal{N}$ is the set of port or node names

A constraint automaton is a tuple $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$

- Q is the state space
- $Q_0 \subseteq Q$ the set of initial states
- $\mathcal{N}$ is the set of port or node names

$\mathcal{N}_{\mathbf{in}}$ and $\mathcal{N}_{\mathbf{out}}$ are disjoint subsets of $\mathcal{N}$, specifying I/O-ports of a component or connector

A constraint automaton is a tuple $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$

- Q is the state space
- $Q_0 \subseteq Q$ the set of initial states
- $\mathcal{N}$ is the set of port or node names
- transition relation $\longrightarrow \subseteq Q \times 2^{\mathcal{N}} \times DC \times Q$

Constraint automaton (CA)

A constraint automaton is a tuple $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$

- Q is the state space
- $Q_0 \subseteq Q$ the set of initial states
- $\mathcal{N}$ is the set of port or node names
- transition relation $\longrightarrow \subseteq Q \times 2^{\mathcal{N}} \times DC \times Q$

transitions have the form $q \xrightarrow{N, g} q'$ where

$q, q' \in Q$	are states,
$N \subseteq \mathcal{N}$	set of active ports

A constraint automaton is a tuple $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$

- Q is the state space
- $Q_0 \subseteq Q$ the set of initial states
- $\mathcal{N}$ is the set of port or node names
- transition relation $\longrightarrow \subseteq Q \times 2^{\mathcal{N}} \times DC \times Q$

transitions have the form $q \xrightarrow{N, g} q'$ where

$q, q' \in Q$	are states,
$N \subseteq \mathcal{N}$	set of active ports
$g \in DC(N)$	data constraint for the data items sent or received at the active ports

Boolean conditions on the sent or received data items at the I/O-ports of components or nodes of the network

Boolean conditions on the sent or received data items at the I/O-ports of components or nodes of the network

$$g \ ::= \ d_A = d \ \big| \ d_A = d_B \ \big| \ g_1 \vee g_2 \ \big| \ \neg g$$

finite, global data domain ***Data***

Boolean conditions on the sent or received data items at the I/O-ports of components or nodes of the network

$$g \ ::= \ \boxed{d_A = d} \ \mid \ d_A = d_B \ \mid \ g_1 \vee g_2 \ \mid \ \neg g$$

data value at port A equals $d \in \mathit{Data}$

finite, global data domain Data

Boolean conditions on the sent or received data items at the I/O-ports of components or nodes of the network

$$g \;::=\; d_A = d \;\;\big|\;\; d_A = d_B \;\;\big|\;\; g_1 \vee g_2 \;\;\big|\;\; \neg g$$

data value at port A equals $d \in \mathit{Data}$

data values at ports A and B agree

finite, global data domain Data

Executions in constraint automata

240

Executions in constraint automata

sequence of transitions in CA

$$q_0 \xrightarrow{N_1, g_1} q_1 \xrightarrow{N_2, g_2} q_2 \xrightarrow{N_3, g_3} \ldots$$

induces a set of executions

$$\eta = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} q_2 \xrightarrow{c_3} \ldots$$

where $c_1, c_2, c_3, \ldots$ are concurrent I/O operations sequence of transitions in CA

$$q_0 \xrightarrow{N_1, g_1} q_1 \xrightarrow{N_2, g_2} q_2 \xrightarrow{N_3, g_3} \ldots$$

induces a set of executions

$$\eta = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} q_2 \xrightarrow{c_3} \ldots$$

where $c_1, c_2, c_3, \ldots$ are concurrent I/O operations

simultaneous
send/receive operations
at the active ports

Executions in constraint automata

sequence of transitions in CA

$$q_0 \xrightarrow{N_1, g_1} q_1 \xrightarrow{N_2, g_2} q_2 \xrightarrow{N_3, g_3} \ldots$$

induces a set of executions

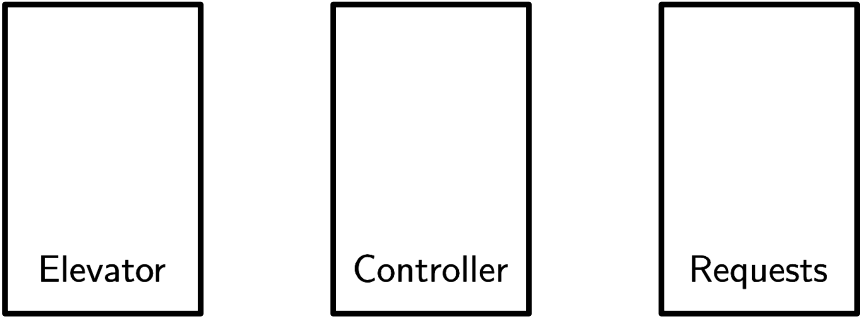
$$\eta = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} q_2 \xrightarrow{c_3} \ldots$$

where $c_1, c_2, c_3, \ldots$ are concurrent I/O operations

$c_i : N_i \to \mathit{Data}$ is a data assignment with $c_i \models g_i$

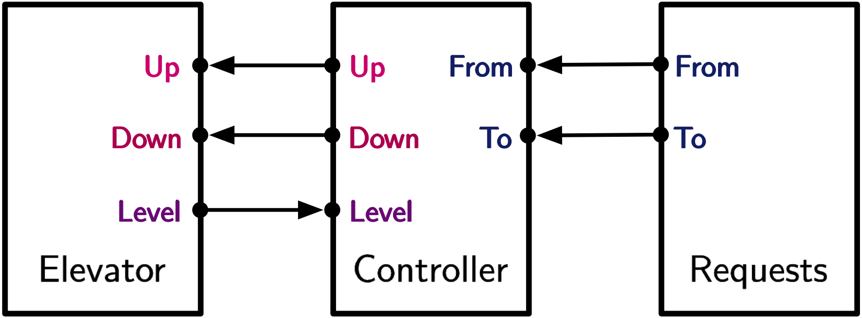
assigns to each port $A \in N_i$ the data item sent or received at A

Example: elevator system

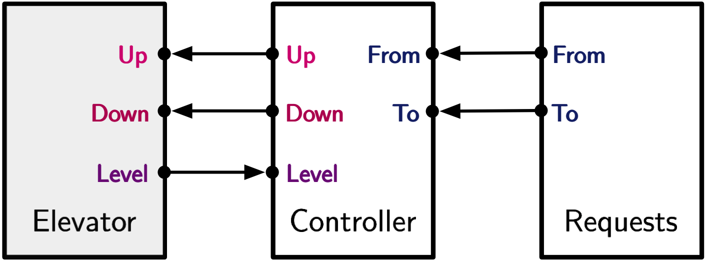


Example: elevator system

250



Example: CA for elevator component

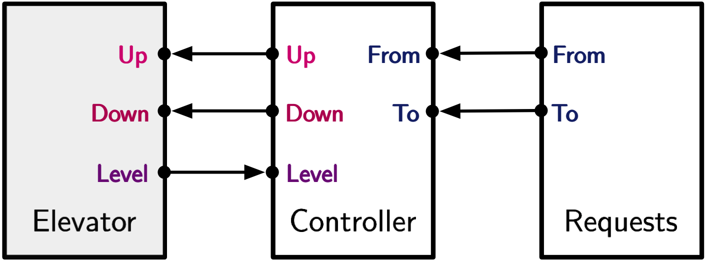


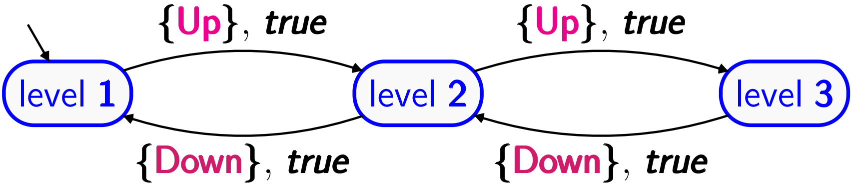
port set $\mathcal{N} = \{\text{Up}, \text{Down}, \text{Level}\}$

$\mathcal{N}_{\text{in}} = \{\text{Up}, \text{Down}\}$, $\mathcal{N}_{\text{out}} = \{\text{Level}\}$

$\mathit{Data} = \{1, 2, 3\}$ three floors

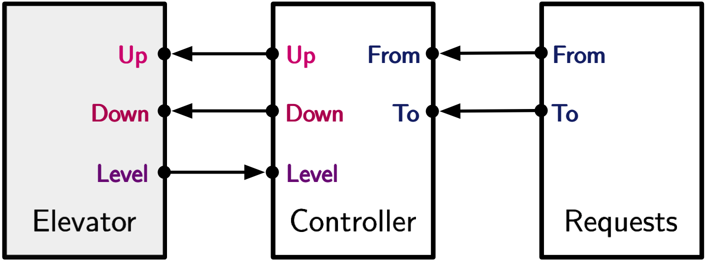
Example: CA for elevator component

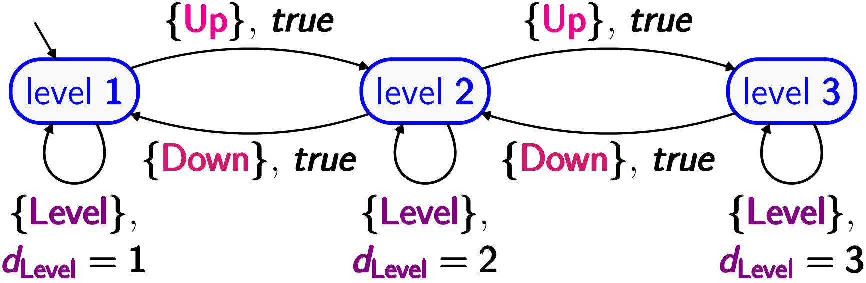




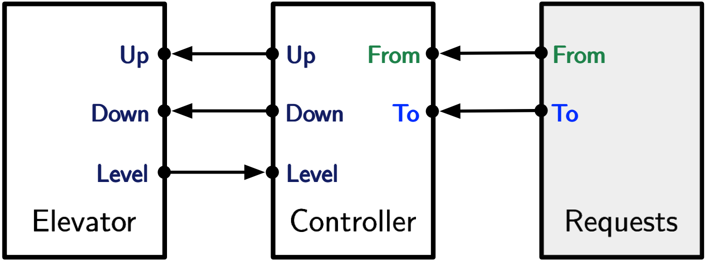
$Data = \{1, 2, 3\}$ three floors

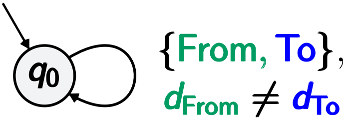
Example: CA for elevator component





Example: CA for request component





$$\mathcal{N} = \mathcal{N}_{\text{out}} = \{\mathsf{From}, \mathsf{To}\} \qquad \mathcal{N}_{\text{in}} = \varnothing$$

Parallel composition of CA

constraint automaton for the composite system is built compositionally using parallel operator $\bowtie$

constraint automaton for the composite system is built compositionally using parallel operator $\bowtie$

$$\underbrace{C_1 \bowtie C_2 \bowtie \ldots \bowtie C_n}_{\text{CA for the component interfaces}}$$

constraint automaton for the composite system is built compositionally using parallel operator $\bowtie$

$$\underbrace{C_1 \bowtie C_2 \bowtie \ldots \bowtie}_{\text{CA for the component interfaces}} C_n \bowtie \underset{\substack{\uparrow \\ \text{CA for the connector, e.g., Reo network}}}{\mathcal{R}}$$

Parallel composition of CA

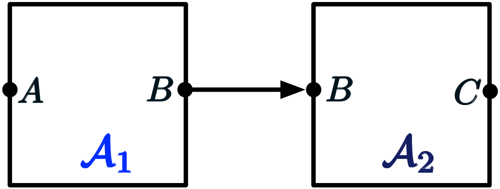
constraint automaton for the composite system is built compositionally using parallel operator $\bowtie$

$$\underbrace{C_1 \bowtie C_2 \bowtie \ldots \bowtie}_{\text{CA for the component interfaces}} C_n \bowtie \underset{\substack{\uparrow \\ \text{CA for the connector, e.g., Reo network}}}{\mathcal{R}}$$

relies on a product construction with

- synchronization over shared ports
- interleaving of I/O-operations without shared ports

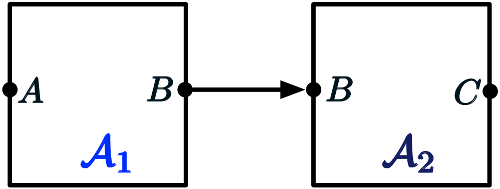
Product for constraint automata



$\mathcal{N}_1 = \{A, B\}$ $\qquad$ $\mathcal{N}_2 = \{B, C\}$

such that classification into input or output ports is "compatible"

Product for constraint automata

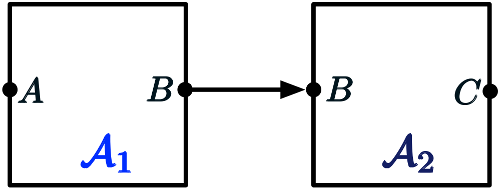


$\mathcal{N}_1 = \{A, B\}$ $\qquad$ $\mathcal{N}_2 = \{B, C\}$

$$\mathcal{A}_1 = (Q_1, \mathcal{N}_1, \longrightarrow_1, Q_0^1) \qquad \mathcal{A}_2 = (Q_2, \mathcal{N}_2, \longrightarrow_2, Q_0^2)$$

such that classification into input or output ports is "compatible"

Product for constraint automata

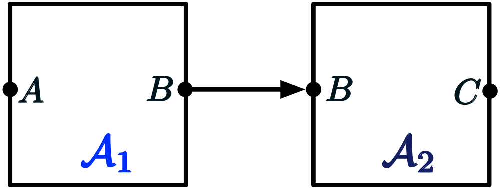


$\mathcal{N}_1 = \{A, B\}$ $\qquad$ $\mathcal{N}_2 = \{B, C\}$

$$\mathcal{A}_1 = (Q_1, \mathcal{N}_1, \longrightarrow_1, Q_0^1) \qquad \mathcal{A}_2 = (Q_2, \mathcal{N}_2, \longrightarrow_2, Q_0^2)$$

$$\mathcal{A}_1 \bowtie \mathcal{A}_2 = (\, Q_1 \times Q_2, \qquad \ldots\,)$$

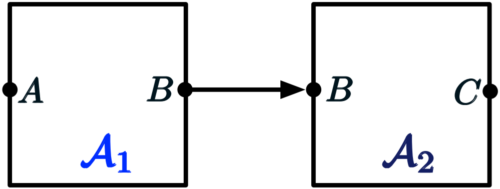
Product for constraint automata



$\mathcal{N}_1 = \{A, B\}$ $\mathcal{N}_2 = \{B, C\}$

$$\mathcal{A}_1 = (Q_1, \mathcal{N}_1, \longrightarrow_1, Q_0^1) \qquad \mathcal{A}_2 = (Q_2, \mathcal{N}_2, \longrightarrow_2, Q_0^2)$$

$$\mathcal{A}_1 \bowtie \mathcal{A}_2 = (\, Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \ldots \,)$$

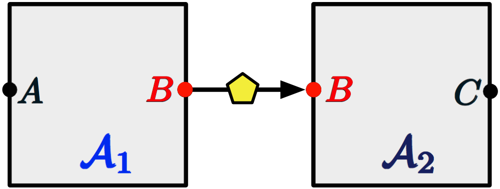


$\mathcal{N}_1 = \{A, B\}$ $\mathcal{N}_2 = \{B, C\}$

$$\mathcal{A}_1 = (Q_1, \mathcal{N}_1, \longrightarrow_1, Q_0^1) \qquad \mathcal{A}_2 = (Q_2, \mathcal{N}_2, \longrightarrow_2, Q_0^2)$$

$$\mathcal{A}_1 \bowtie \mathcal{A}_2 = (\, Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \ \longrightarrow, Q_0^1 \times Q_0^2)$$

Product for constraint automata

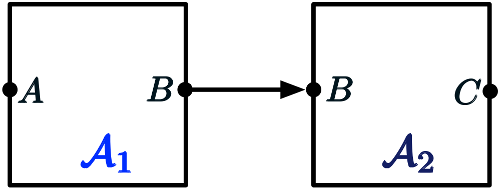


$\mathcal{N}_1 = \{A, B\}$ $\qquad$ $\mathcal{N}_2 = \{B, C\}$

$$\mathcal{A}_1 = (Q_1, \mathcal{N}_1, \longrightarrow_1, Q_0^1) \qquad \mathcal{A}_2 = (Q_2, \mathcal{N}_2, \longrightarrow_2, Q_0^2)$$

$$\mathcal{A}_1 \bowtie \mathcal{A}_2 = (\, Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \; \longrightarrow, \; Q_0^1 \times Q_0^2)$$

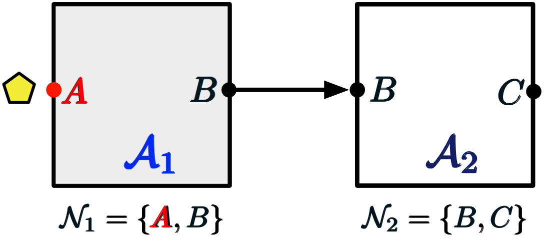
synchronizing over the shared ports
$B \in \mathcal{N}_1 \cap \mathcal{N}_2$



$\mathcal{N}_1 = \{A, B\}$ $\qquad$ $\mathcal{N}_2 = \{B, C\}$

I/O-operations of $\mathcal{A}_1$ without shared ports

$$\frac{q_1 \xrightarrow{N_1, g_1} q_1' \quad \text{and} \quad N_1 \cap \mathcal{N}_2 = \varnothing}{\langle q_1, q_2 \rangle \xrightarrow{N_1, g_1} \langle q_1', q_2 \rangle}$$

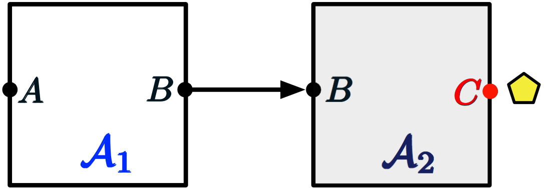


I/O-operations of $\mathcal{A}_1$ without shared ports

$$\frac{q_1 \xrightarrow{N_1, g_1} q_1' \quad \text{and} \quad \boxed{N_1 \cap \mathcal{N}_2 = \varnothing}}{\langle q_1, q_2 \rangle \xrightarrow{N_1, g_1} \langle q_1', q_2 \rangle}$$

"data flow only at non-shared ports of $\mathcal{A}_1$"

Product for constraint automata

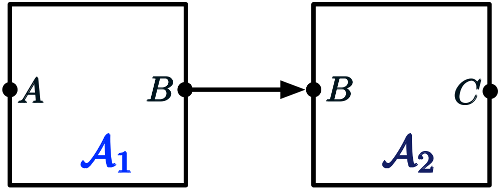


$\mathcal{N}_1 = \{A, B\}$ $\qquad$ $\mathcal{N}_2 = \{B, C\}$

I/O-operations of $\mathcal{A}_2$ without shared ports

$$\frac{q_2 \xrightarrow{N_2, g_2} q_2' \quad \text{and} \quad N_2 \cap \mathcal{N}_1 = \varnothing}{\langle q_1, q_2 \rangle \xrightarrow{N_2, g_2} \langle q_1, q_2' \rangle}$$

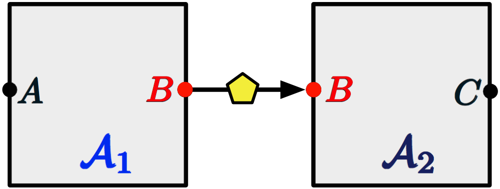
"data flow only at non-shared ports of $\mathcal{A}_2$"



$\mathcal{N}_1 = \{A, B\}$ $\qquad$ $\mathcal{N}_2 = \{B, C\}$

synchronized data flow at shared ports:

$$\frac{q_1 \xrightarrow{N_1, g_1} q_1' \;\wedge\; q_2 \xrightarrow{N_2, g_2} q_2' \;\wedge\; N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2,\, g_1 \wedge g_2} \langle q_1', q_2' \rangle}$$

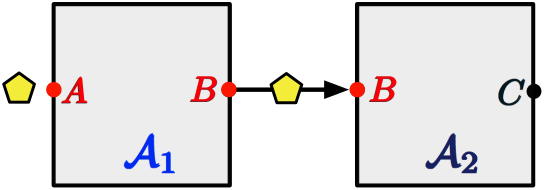


$\mathcal{N}_1 = \{A, B\}$ $\mathcal{N}_2 = \{B, C\}$

synchronized data flow at shared ports:

$$\frac{q_1 \xrightarrow{N_1, g_1} q_1' \ \wedge \ q_2 \xrightarrow{N_2, g_2} q_2' \ \wedge \ N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, \, g_1 \wedge g_2} \langle q_1', q_2' \rangle}$$

Product for constraint automata

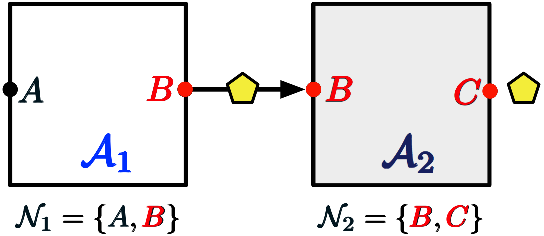


$\mathcal{N}_1 = \{A, B\}$ $\qquad$ $\mathcal{N}_2 = \{B, C\}$

synchronized data flow at shared ports:

$$\frac{q_1 \xrightarrow{N_1, g_1} q_1' \ \wedge\ q_2 \xrightarrow{N_2, g_2} q_2' \ \wedge\ N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2,\, g_1 \wedge g_2} \langle q_1', q_2' \rangle}$$

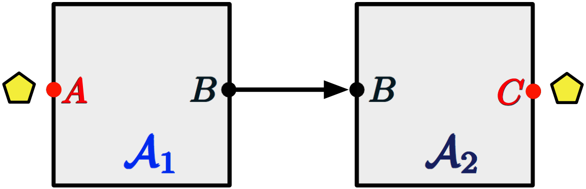
possibly in parallel with data flow at non-shared ports



synchronized data flow at shared ports:

$$\frac{q_1 \xrightarrow{N_1, g_1} q_1' \;\wedge\; q_2 \xrightarrow{N_2, g_2} q_2' \;\wedge\; N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2,\; g_1 \wedge g_2} \langle q_1', q_2' \rangle}$$

possibly in parallel with data flow at non-shared ports



$\mathcal{N}_1 = \{A, B\}$ $\mathcal{N}_2 = \{B, C\}$

synchronized data flow at shared ports:

$$\frac{q_1 \xrightarrow{N_1,\, g_1} q_1' \;\wedge\; q_2 \xrightarrow{N_2,\, g_2} q_2' \;\wedge\; N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2,\, g_1 \wedge g_2} \langle q_1', q_2' \rangle}$$

... covers the case where $N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1 = \varnothing$ (needed to ensure associativity)

Overview

1. **Modelling components and connectors**
   - constraint automata (CA)
   - coordination language Reo ⟵
2. Model checking with CA
   - Linear Temporal Logic
   - Alternating Stream Logic
3. Synthesis of connectors

Coordination language Reo [Arbab'04]

Coordination language Reo [Arbab'04]

- exogenous coordination aims to provide a strict separation between computation and coordination

Coordination language Reo [Arbab'04]

- exogenous coordination aims to provide a strict separation between computation and coordination
- offers a compositional framework to construct component connectors based on a calculus of channels

Coordination language Reo [Arbab'04] 110

- exogenous coordination aims to provide a strict separation between computation and coordination
- offers a compositional framework to construct component connectors based on a calculus of channels
  - creating a new channel
  - joining channel ends in a node
  - building components by hiding nodes

Coordination language Reo [Arbab'04]

- exogenous coordination aims to provide a strict separation between computation and coordination
- offers a compositional framework to construct component connectors based on a calculus of channels
  - creating a new channel
  - joining channel ends in a node
  - building components by hiding nodes

Reo network: graph of channels
edges: channels
nodes: combine several channel ends

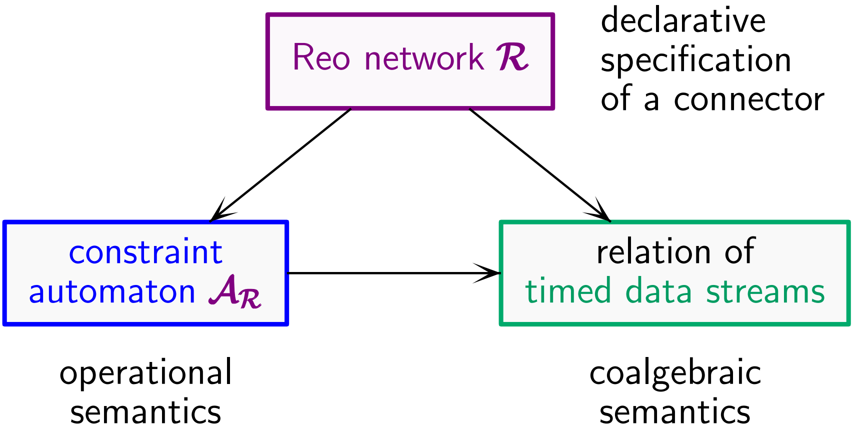
- exogenous coordination aims to provide a strict separation between computation and coordination
- offers a compositional framework to construct component connectors based on a calculus of channels
  - ∗ creating a new channel
  - ∗ joining channel ends in a node
  - ∗ building components by hiding nodes

Reo network: graph of channels ⟵ semantics: constraint automaton

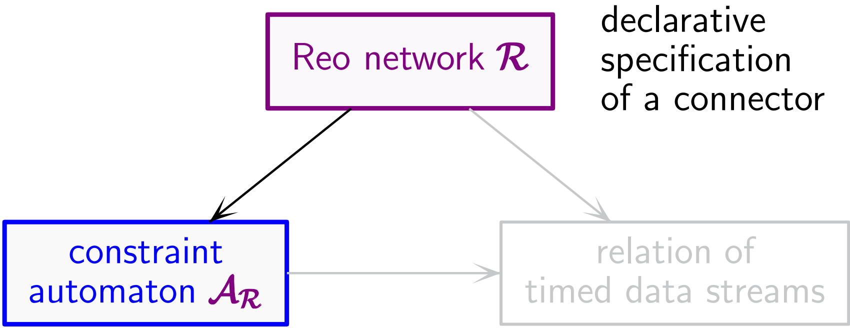
edges: channels

nodes: combine several channel ends

140

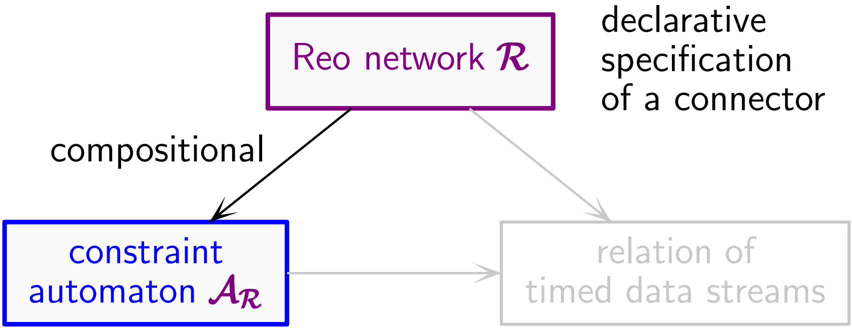

Reo network $\mathcal{R}$
declarative specification of a connector
constraint automaton $\mathcal{A}_{\mathcal{R}}$
relation of timed data streams
operational semantics
coalgebraic semantics

Semantics of Reo



states $\widehat{=}$ configurations of $\mathcal{R}$
(e.g., content of FIFO channels)

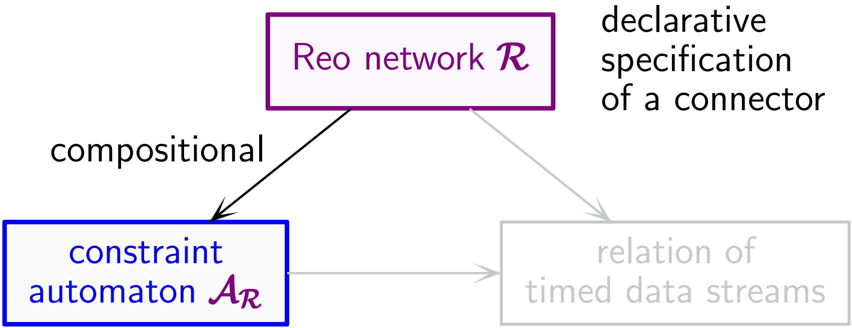
Semantics of Reo



states $\;\widehat{=}\;$ configurations of $\mathcal{R}$
(e.g., content of FIFO channels)

compositional approach: CA $\mathcal{A}_{\mathcal{R}}$ arises from the product of the CA for the channels, nodes, (sub-)connectors of $\mathcal{R}$

Semantics of Reo



states $\widehat{=}$ configurations of $\mathcal{R}$
(e.g., content of FIFO channels)

compositional approach: CA $\mathcal{A}_{\mathcal{R}}$ arises from the product of the CA for the channels, nodes, (sub-)connectors of $\mathcal{R}$

composite system: $\mathcal{C}_1 \bowtie \ldots \bowtie \mathcal{C}_n \bowtie \mathcal{A}_{\mathcal{R}}$

1 **Modelling components and connectors**

constraint automata (CA)

coordination language Reo

Reo channels $\longleftarrow$

Reo nodes

2 Model checking with CA

Linear Temporal Logic

Alternating Stream Logic

3 Synthesis of connectors

Reo channels

- provide the basic building blocks
  for the connector glue code

- provide the basic building blocks
  for the connector glue code
- have two channel ends,
  each can be either a source or a sink end
  - source end:
    data item enters channel
  - sink end:
    data item leaves channel

Reo channels

- provide the basic building blocks
  for the connector glue code
- have two channel ends,
  each can be either a source or a sink end
  - source end:
    data item enters channel $\longleftarrow$ input port
  - sink end:
    data item leaves channel $\longleftarrow$ output port

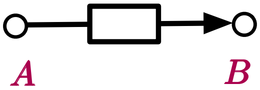
- provide the basic building blocks
  for the connector glue code
- have two channel ends,
  each can be either a source or a sink end
  - source end:
    data item enters channel $\longleftarrow$ input port
  - sink end:
    data item leaves channel $\longleftarrow$ output port
- from library of basic channels
  or user defined (CA + types of channel end)

Reo basic channels

- FIFO channel
- synchronous channel
- synchronous drain
- synchronous spout
- filter channel
- non-deterministic lossy channel

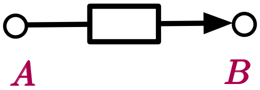
- FIFO channel ⟵ asynchronous communication via buffer
- synchronous channel
- synchronous drain
- synchronous spout
- filter channel
- non-deterministic lossy channel

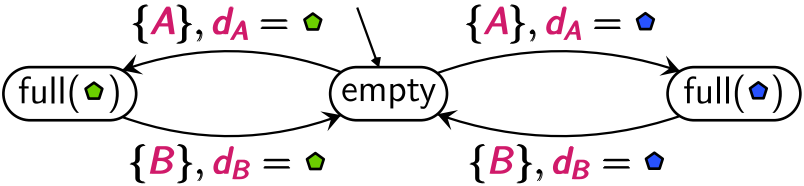
FIFO channel with 1 buffer cell



$$\mathcal{N}_{\mathbf{in}} = \{A\} \qquad \mathcal{N}_{\mathbf{out}} = \{B\}$$

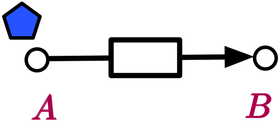
FIFO channel with 1 buffer cell



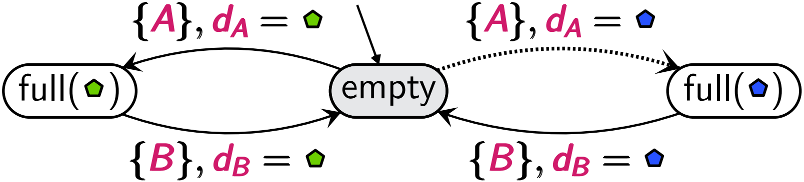


$\mathcal{N}_{\text{in}} = \{A\}$ $\qquad$ $\mathcal{N}_{\text{out}} = \{B\}$

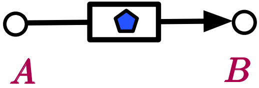
FIFO channel with 1 buffer cell



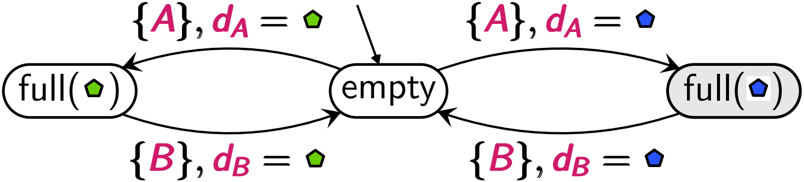
	A	B
t_1		



$\mathcal{N}_{\text{in}} = \{A\}$ $\mathcal{N}_{\text{out}} = \{B\}$

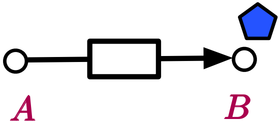


	A	B
t_1	⬟	

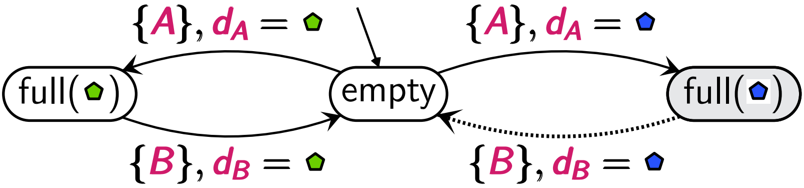


$$\mathcal{N}_{\text{in}} = \{A\} \qquad \mathcal{N}_{\text{out}} = \{B\}$$

FIFO channel with 1 buffer cell

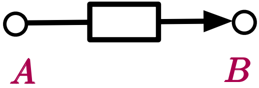


	A	B
t_1	⬟	
t_2		⬟

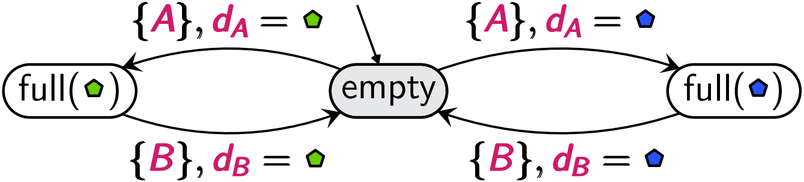


$$\mathcal{N}_{\text{in}} = \{A\} \qquad \mathcal{N}_{\text{out}} = \{B\}$$

FIFO channel with 1 buffer cell



	A	B
t_1	⬟	
t_2		⬟

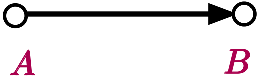


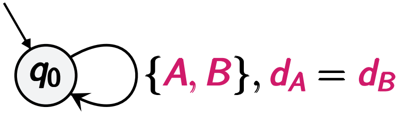
$$\mathcal{N}_{\text{in}} = \{A\} \qquad \mathcal{N}_{\text{out}} = \{B\}$$

Reo basic channels

- FIFO channel
- synchronous channel ⟵
- synchronous drain
- synchronous spout
- filter channel
- non-deterministic lossy channel

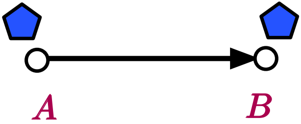
Synchronous channel



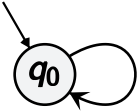


$\mathcal{N}_{\text{in}} = \{A\}$ $\mathcal{N}_{\text{out}} = \{B\}$

Synchronous channel

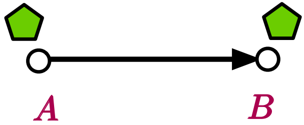


	A	B
t_1	⬟	⬟

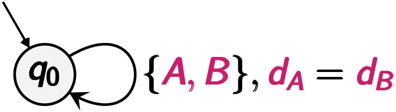


$\{A, B\}, d_A = d_B$

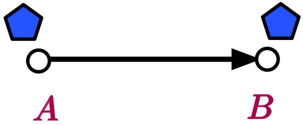
$\mathcal{N}_{\text{in}} = \{A\}$ $\mathcal{N}_{\text{out}} = \{B\}$

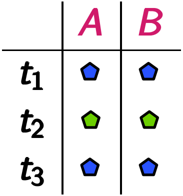


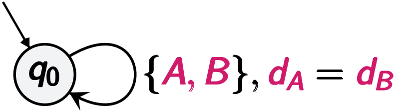
	A	B
t_1		
t_2		



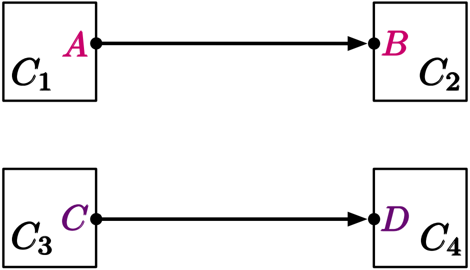
$$\mathcal{N}_{\text{in}} = \{A\} \qquad \mathcal{N}_{\text{out}} = \{B\}$$







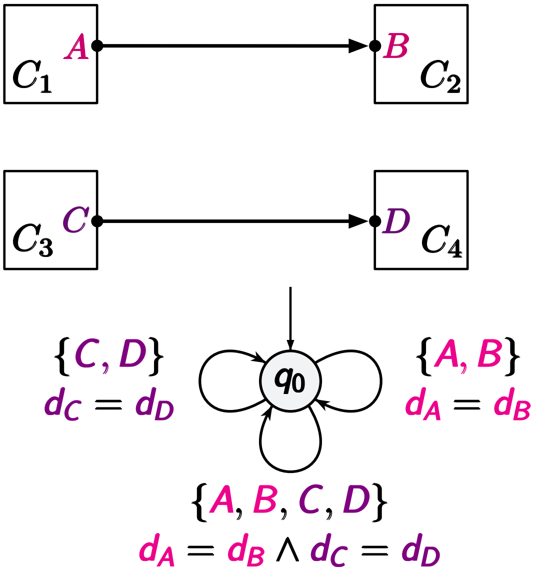
$\mathcal{N}_{\text{in}} = \{A\}$ $\mathcal{N}_{\text{out}} = \{B\}$



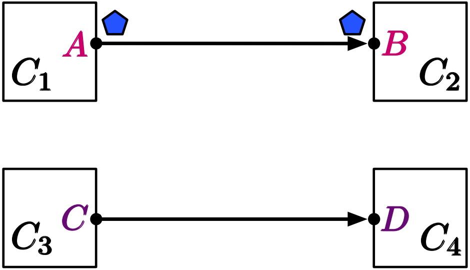
Reo network with two channels:

- synchronous channel $A\,B$ connecting components C_1 with C_2
- synchronous channel $C\,D$ connecting components C_3 with C_4

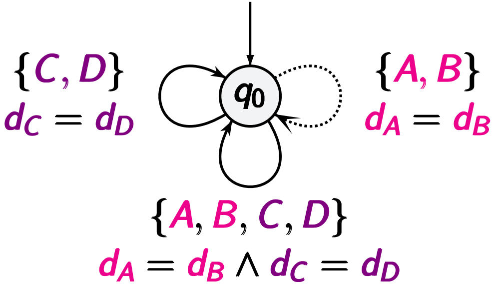
Example: Reo network and four components



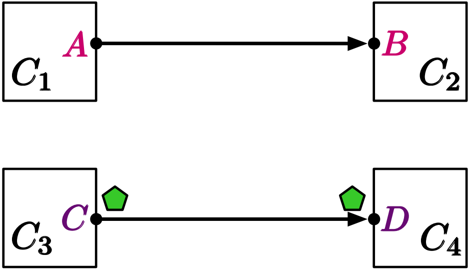
Example: Reo network and four components



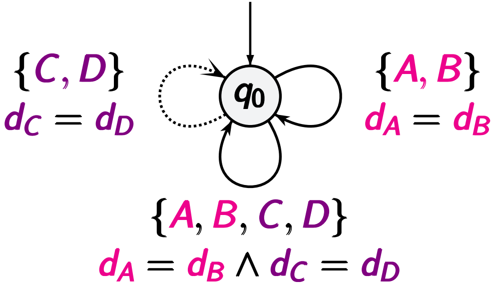
	A	B	C	D
t_1	⬟	⬟		

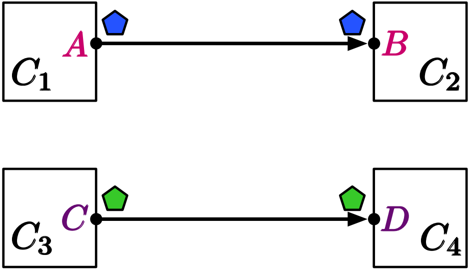


Example: Reo network and four components

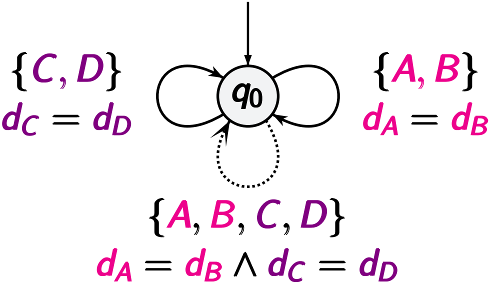


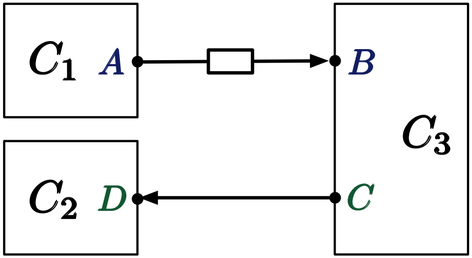
	A	B	C	D
t_1	⬟	⬟		
t_2			⬟	⬟





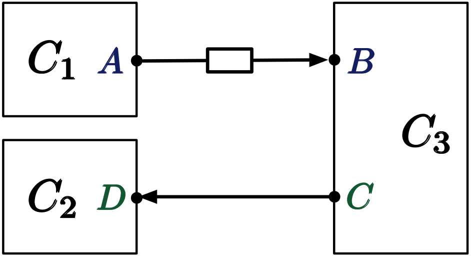
	A	B	C	D
t_1	⬟	⬟		
t_2			⬟	⬟
t_3	⬟	⬟	⬟	⬟



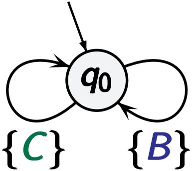


system with three components

- fifo channel AB
- synchronous channel CD



component interface for C_3

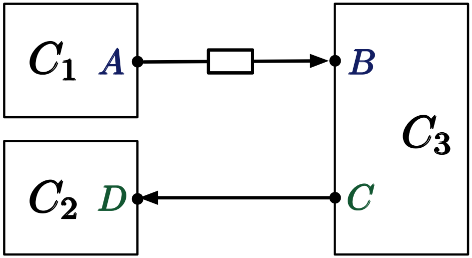


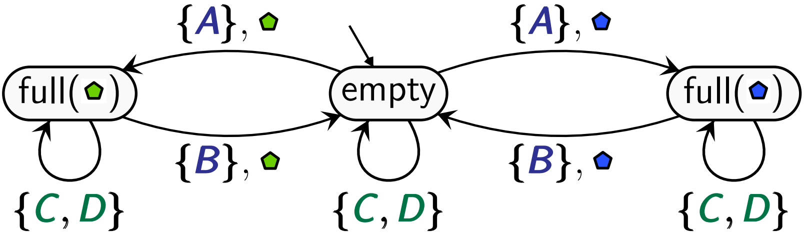
system with three components

- fifo channel $A\,B$
- synchronous channel $C\,D$

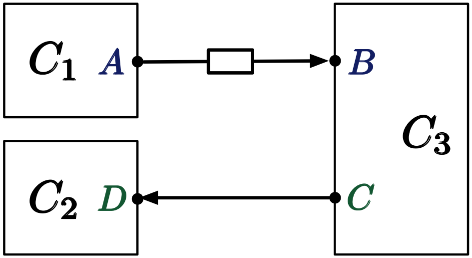
$$\mathcal{N}_{\text{in}} = \{B\}$$
$$\mathcal{N}_{\text{out}} = \{C\}$$

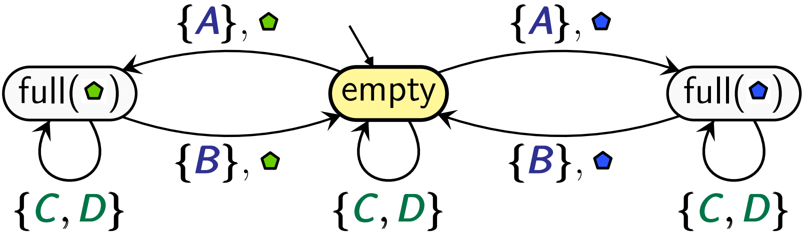
Example: network and its CA-semantics



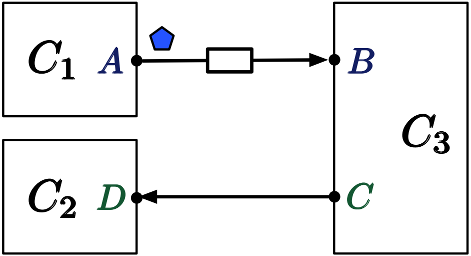


Example: network and its CA-semantics

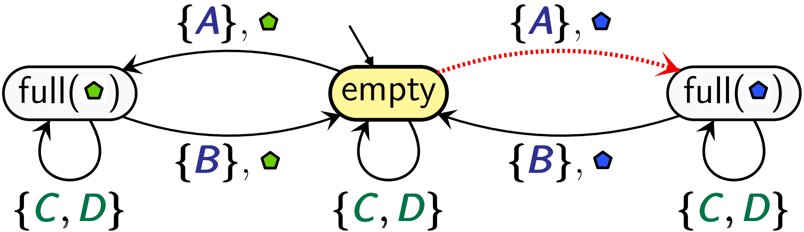


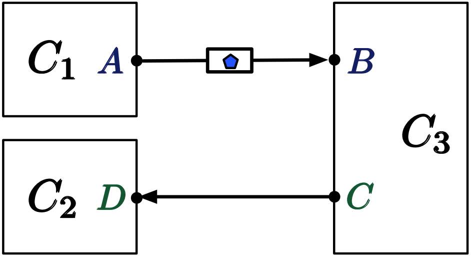


Example: network and its CA-semantics

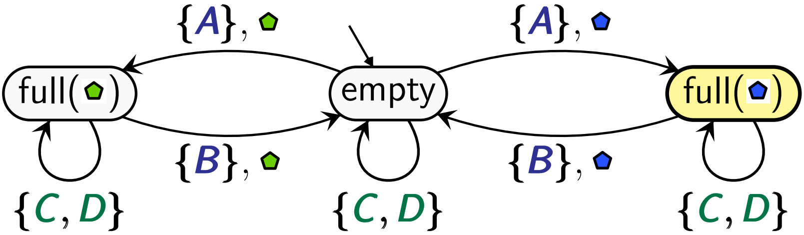


	A	B	C	D
t_1				

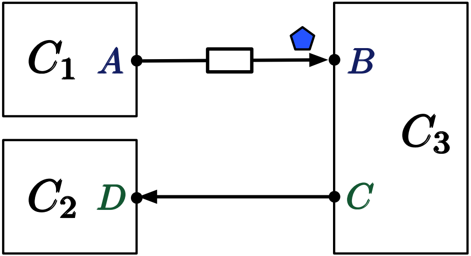


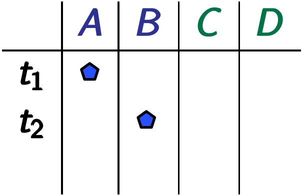


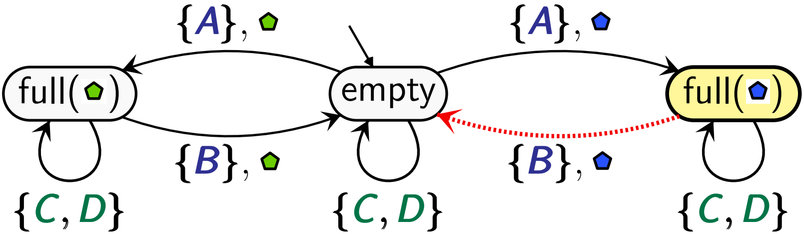
	A	B	C	D
t_1				



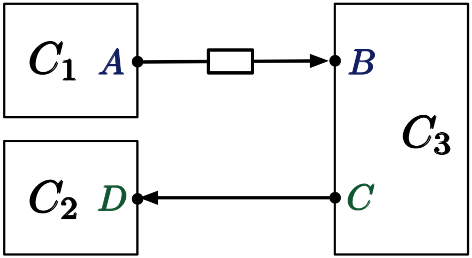
Example: network and its CA-semantics



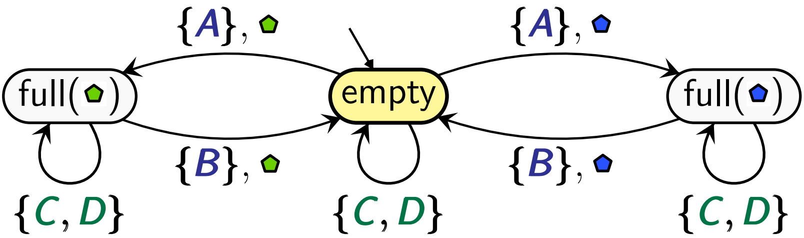




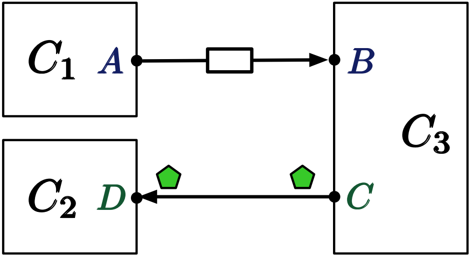
Example: network and its CA-semantics



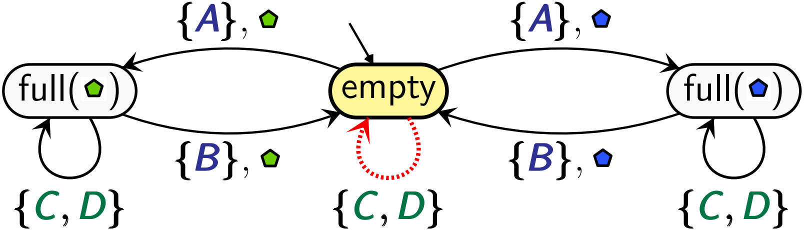
	A	B	C	D
t_1	⬟			
t_2		⬟		



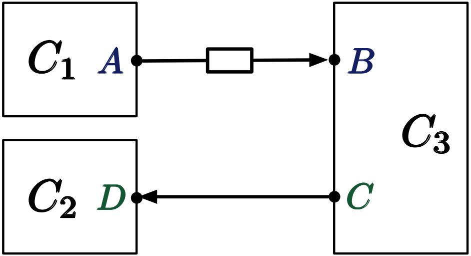
Example: network and its CA-semantics

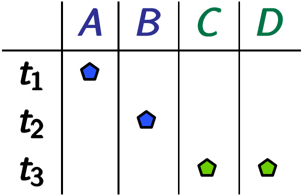


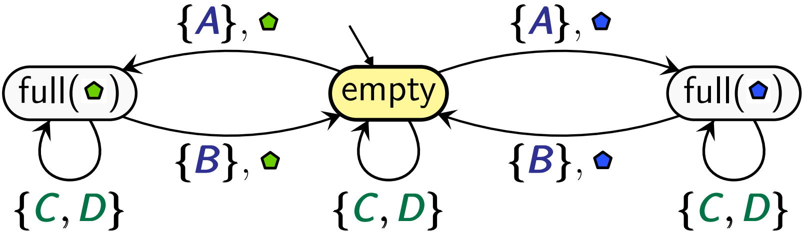
	A	B	C	D
t_1	⬟			
t_2		⬟		
t_3			⬟	⬟



Example: network and its CA-semantics







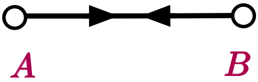
Reo basic channels

- FIFO channel
- synchronous channel
- synchronous drain
- synchronous spout
- filter channel
- non-deterministic lossy channel

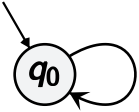
variants of synchronous channels

Synchronous drain

330



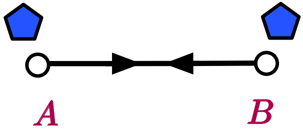
two source ends
(input ports)

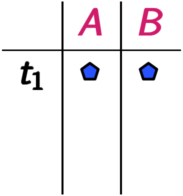


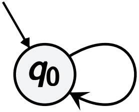
$\{A, B\}, \; d_A = d_B$

$\mathcal{N}_{\mathrm{in}} = \{A, B\} \qquad \mathcal{N}_{\mathrm{out}} = \varnothing$

Synchronous drain



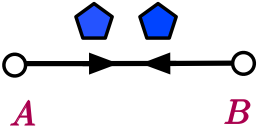




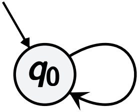
$\{A, B\}, \; d_A = d_B$

$$\mathcal{N}_{\text{in}} = \{A, B\} \qquad \mathcal{N}_{\text{out}} = \varnothing$$

Synchronous drain



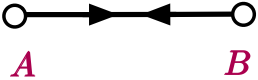
	A	B
t_1		

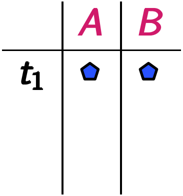


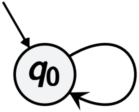
$\{A, B\}, \ d_A = d_B$

$\mathcal{N}_{\text{in}} = \{A, B\}$ $\qquad$ $\mathcal{N}_{\text{out}} = \varnothing$

Synchronous drain

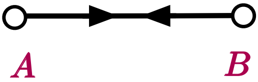




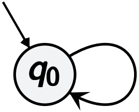


$\{A, B\}, \ d_A = d_B$

$$\mathcal{N}_{\text{in}} = \{A, B\} \qquad \mathcal{N}_{\text{out}} = \varnothing$$

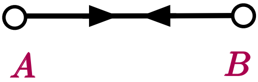


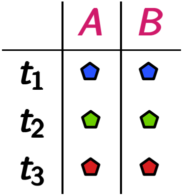
	A	B
t_1	⬟	⬟
t_2	⬟	⬟

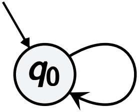


$\{A, B\}$, $d_A = d_B$ $\longleftarrow$ same data items at ports A, B

$$\mathcal{N}_{\mathrm{in}} = \{A, B\} \qquad \mathcal{N}_{\mathrm{out}} = \varnothing$$





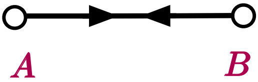


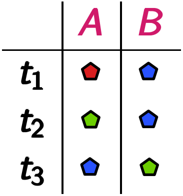
$\{A, B\}, \; d_A = d_B \longleftarrow$

same
data items
at ports A, B

$\mathcal{N}_{\text{in}} = \{A, B\} \qquad \mathcal{N}_{\text{out}} = \varnothing$

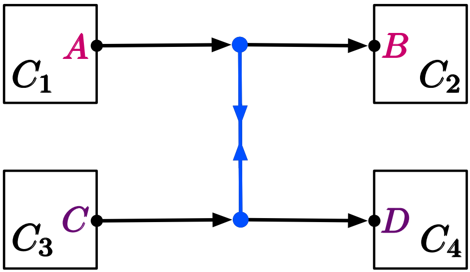
Synchronous drain ⟵ data-abstract





⟵ arbitrary data items at ports A, B

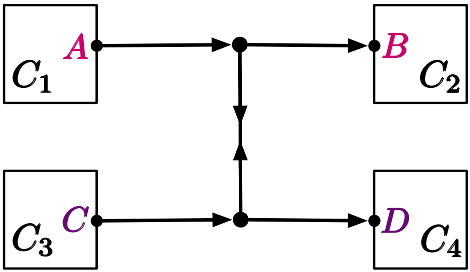
$$\mathcal{N}_{\text{in}} = \{A, B\} \qquad \mathcal{N}_{\text{out}} = \varnothing$$

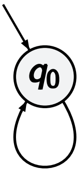


Reo network with four components

- synchronous data flow from ***A*** to ***B***
- synchronous data flow from ***C*** to ***D***
- synchronous drain ensures synchronization of all four components

Example: Reo network with synchronous drain

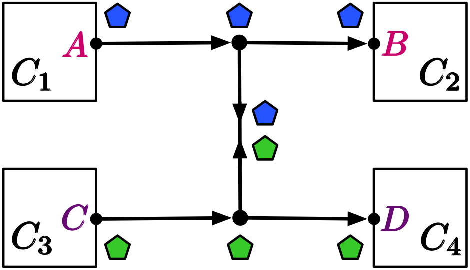




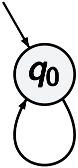
$\{A, B, C, D\}$

$d_A = d_B \wedge d_C = d_D$

Example: Reo network with synchronous drain



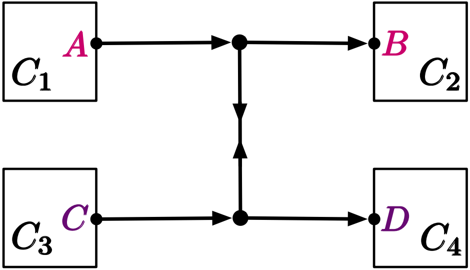
	A	B	C	D
t_1				



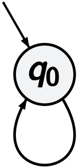
$\{A, B, C, D\}$

$d_A = d_B \wedge d_C = d_D$

Example: Reo network with synchronous drain

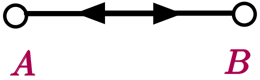


	A	B	C	D
t_1	⬟	⬟	⬟	⬟
t_2	⬟	⬟	⬟	⬟
t_3	⬟	⬟	⬟	⬟



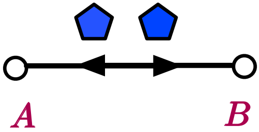
$\{A, B, C, D\}$

$d_A = d_B \wedge d_C = d_D$



two sink ends (output ports)

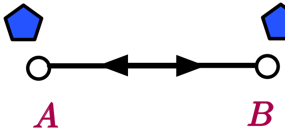
$$\mathcal{N}_{\text{in}} = \varnothing \qquad \mathcal{N}_{\text{out}} = \{A, B\}$$



	A	B
t_1	⬟	⬟

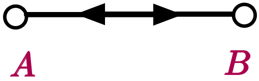
$$\mathcal{N}_{\text{in}} = \varnothing \qquad \mathcal{N}_{\text{out}} = \{A, B\}$$

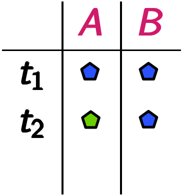
Synchronous spout



	A	B
t_1	⬟	⬟

$$\mathcal{N}_{\text{in}} = \varnothing \qquad \mathcal{N}_{\text{out}} = \{A, B\}$$

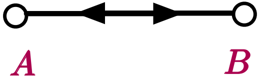


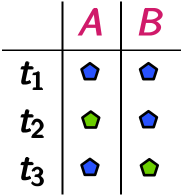


$\mathcal{N}_{\text{in}} = \varnothing \qquad \mathcal{N}_{\text{out}} = \{A, B\}$

Synchronous spout

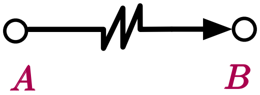
350



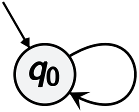


$$\mathcal{N}_{\text{in}} = \varnothing \qquad \mathcal{N}_{\text{out}} = \{A, B\}$$

Filter channel

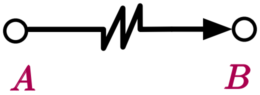


"ordinary" synchronous channel with filter condition



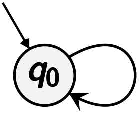
$$\{A, B\},\ d_A = d_B \in \mathit{FilterCond}$$

$$\mathcal{N}_{\text{in}} = \{A\} \qquad \mathcal{N}_{\text{out}} = \{B\}$$



	A	B
t_1	⬟	⬟
t_2	⬟	⬟
t_3	⬟	⬟

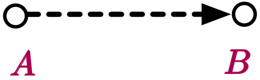
⬟, ⬟ $\in$ *FilterCond*



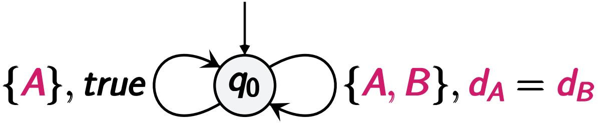
$\{A, B\}, d_A = d_B \in \mathit{FilterCond}$

$$\mathcal{N}_{\text{in}} = \{A\} \qquad \mathcal{N}_{\text{out}} = \{B\}$$

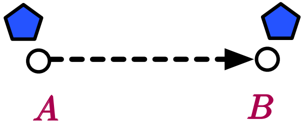
Nondeterministic lossy channel



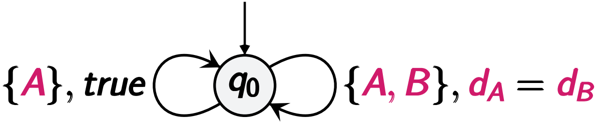
synchronous channel
where written data items
can be lost



$\mathcal{N}_{\text{in}} = \{A\}$ $\qquad$ $\mathcal{N}_{\text{out}} = \{B\}$

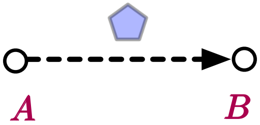


	A	B
t_1	⬟	⬟

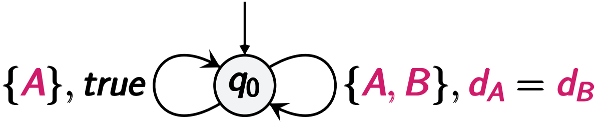


$$\mathcal{N}_{\text{in}} = \{A\} \qquad \mathcal{N}_{\text{out}} = \{B\}$$

Nondeterministic lossy channel

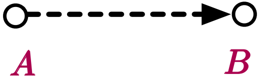


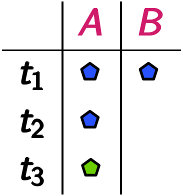
	A	B
t_1	⬟	⬟
t_2	⬟	

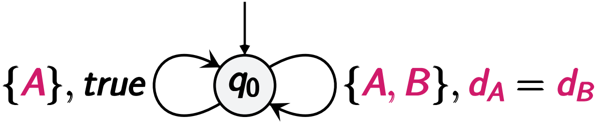


$$\mathcal{N}_{\text{in}} = \{A\} \qquad \mathcal{N}_{\text{out}} = \{B\}$$

Nondeterministic lossy channel







$\mathcal{N}_{\text{in}} = \{A\}$ $\quad\quad$ $\mathcal{N}_{\text{out}} = \{B\}$

1 **Modelling components and connectors**
- constraint automata (CA)
- coordination language Reo
  - Reo channels
  - Reo nodes $\longleftarrow$

2 Model checking with CA
- Linear Temporal Logic
- Alternating Stream Logic

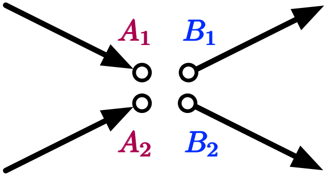
3 Synthesis of connectors

Reo nodes

- arise when channel ends are joined
- coordinate the coincident sink and source ends

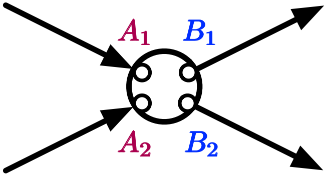
Reo nodes

- arise when channel ends are joined
- coordinate the coincident sink and source ends



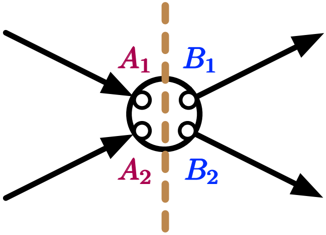
Reo nodes

- arise when channel ends are joined
- coordinate the coincident sink and source ends



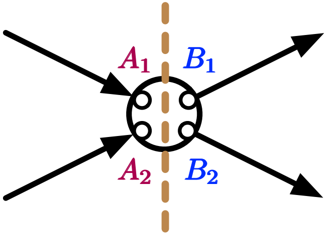
Reo nodes

- arise when channel ends are joined
- coordinate the coincident sink and source ends



Reo nodes with standard semantics

- arise when channel ends are joined
- coordinate the coincident sink and source ends

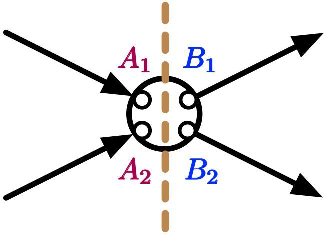


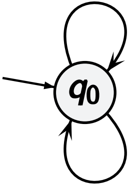
nondeterministic merge
for receive operations
at the sink ends

synchronous
send operation
at the source ends

Reo nodes with standard semantics

- arise when channel ends are joined
- coordinate the coincident sink and source ends



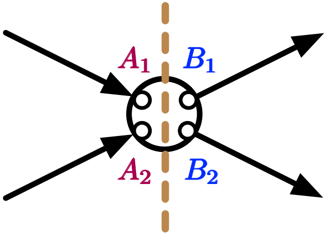


$$\{A_1, B_1, B_2\}, \quad d_{A_1} = d_{B_1} = d_{B_2}$$

$$\{A_2, B_1, B_2\}, \quad d_{A_2} = d_{B_1} = d_{B_2}$$

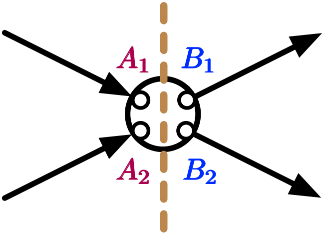
Reo nodes with route semantics

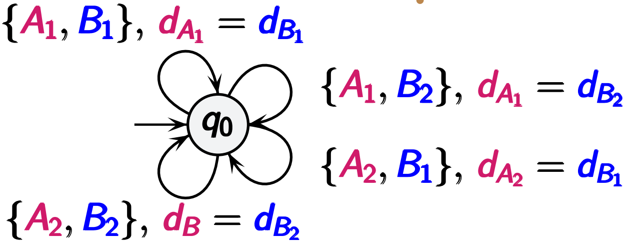
- arise when channel ends are joined
- coordinate the coincident sink and source ends



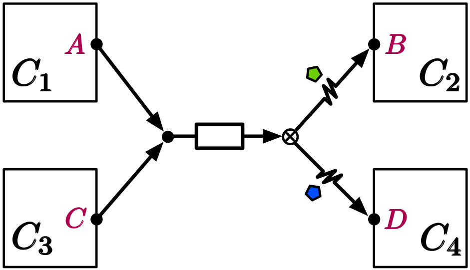
nondeterministic merge for receive operations at the sink ends	nondeterministic choice for the send operations at the source ends

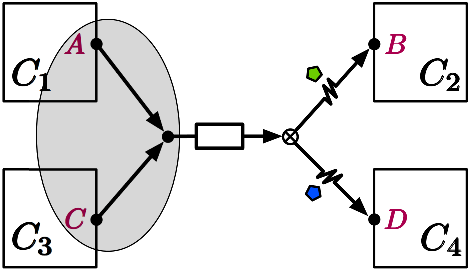
- arise when channel ends are joined
- coordinate the coincident sink and source ends



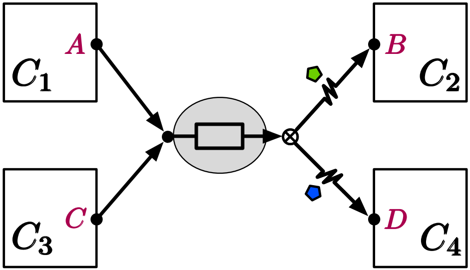


Example: Reo network with buffering & filtering

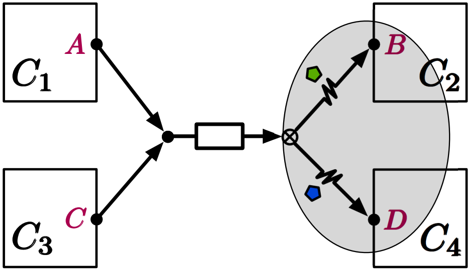




- nondeterministic merge between A and C . . .



- nondeterministic merge between *A* and *C* ...
- data item *d* written at *A* or *C* is stored in the buffer

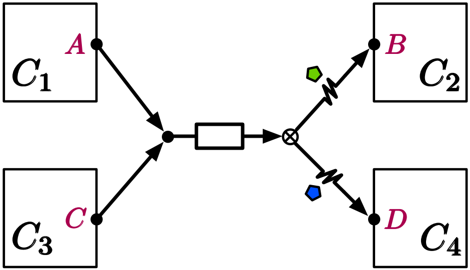


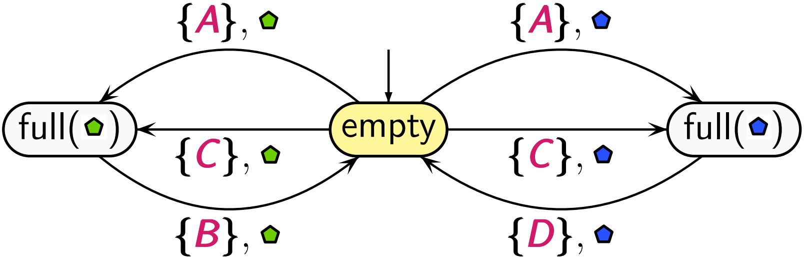
- nondeterministic merge between A and C ...
- data item d written at A or C is stored in the buffer

  if d is green then it will be routed to B

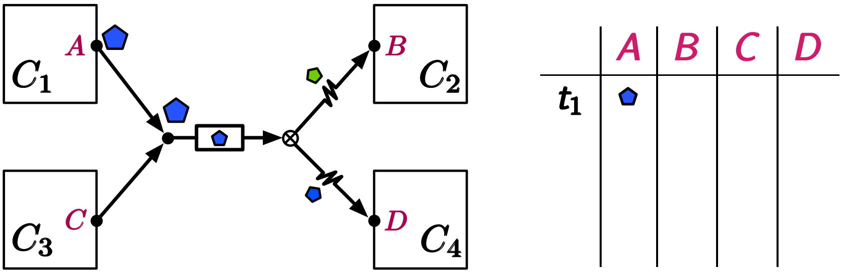
  if d is blue then it will be routed to D

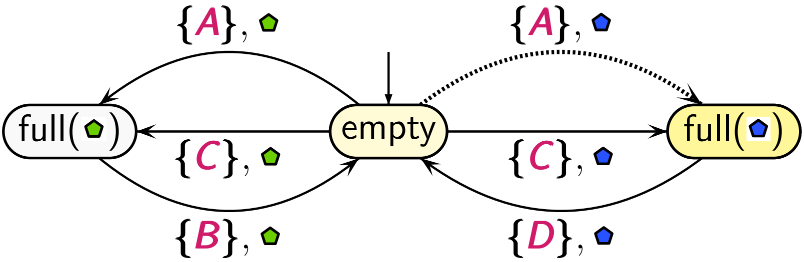
Example: Reo network with buffering & filtering



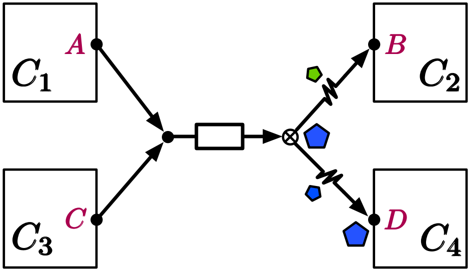


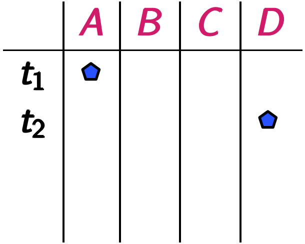
Example: Reo network with buffering & filtering

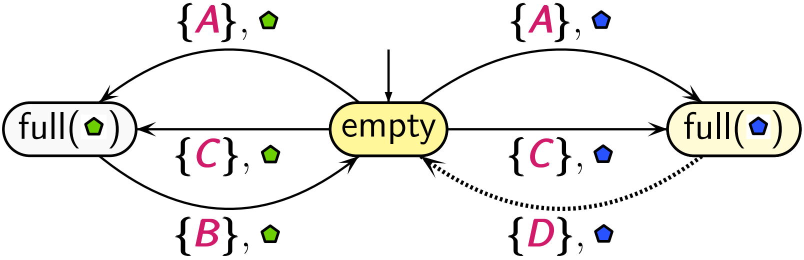




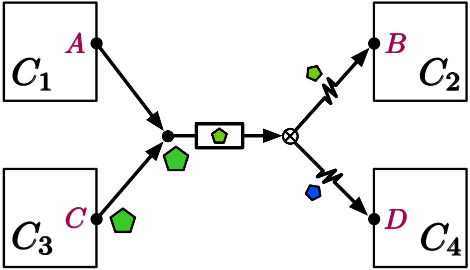
Example: Reo network with buffering & filtering

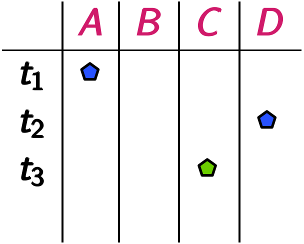


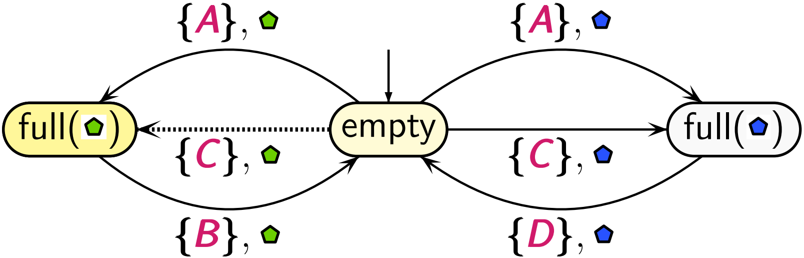




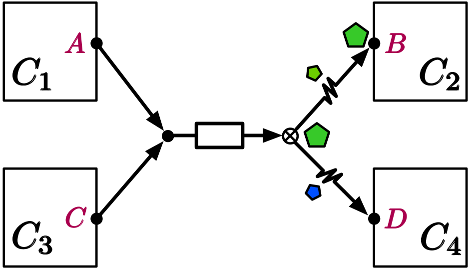
Example: Reo network with buffering & filtering

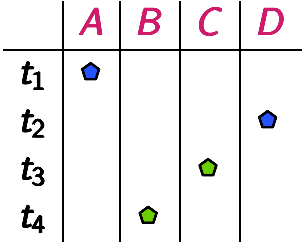


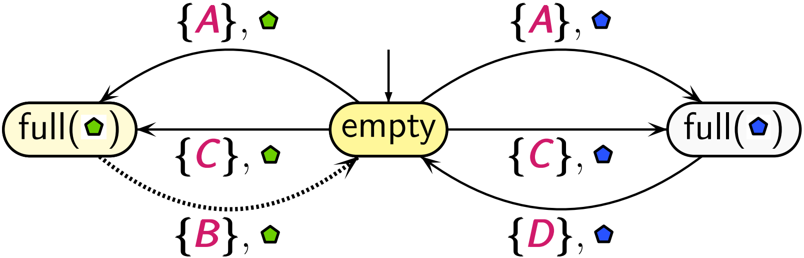




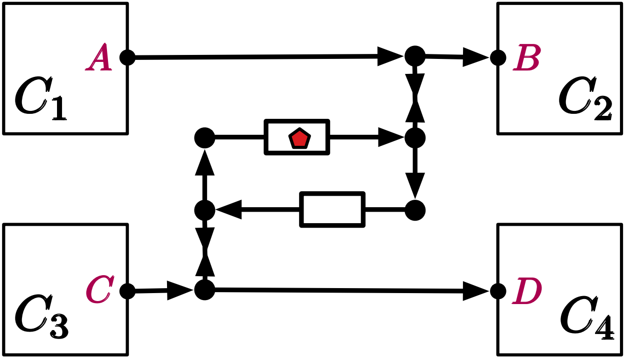
Example: Reo network with buffering & filtering





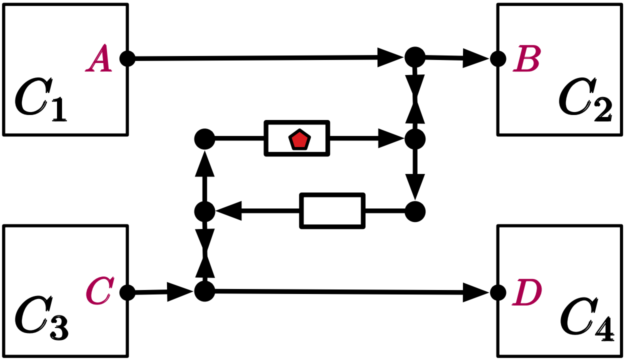


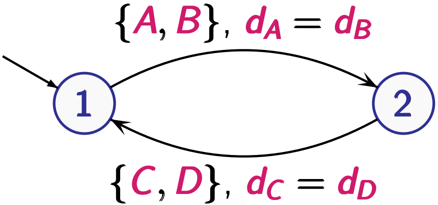
Example: Reo network for a sequencer



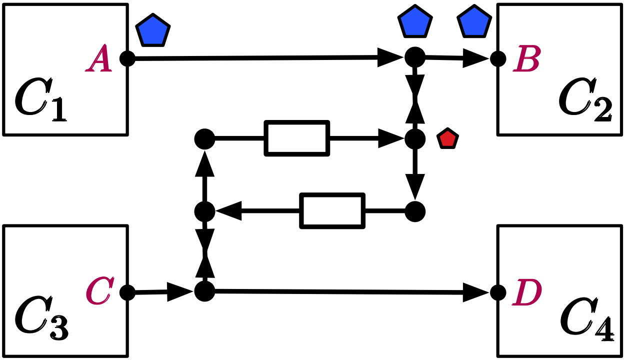
alternating data flow between C_1 and C_2 via $A\,B$
and between C_3 and C_4 via $C\,D$

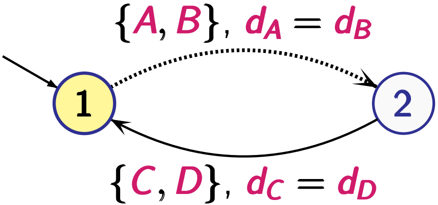
Example: Reo network for a sequencer





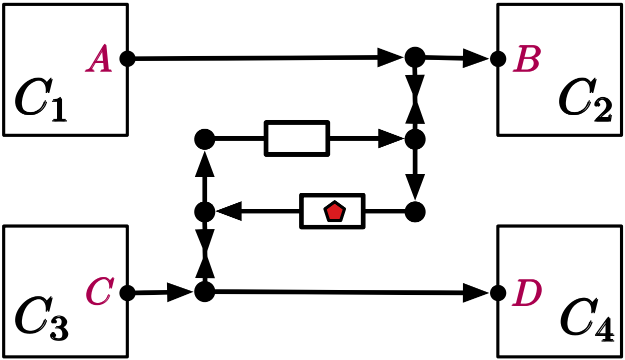
Example: Reo network for a sequencer

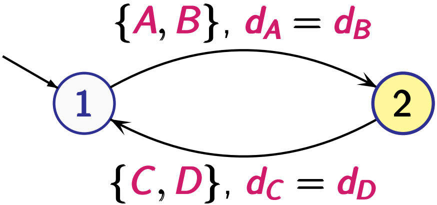




	A	B	C	D
t_1	⬟	⬟		

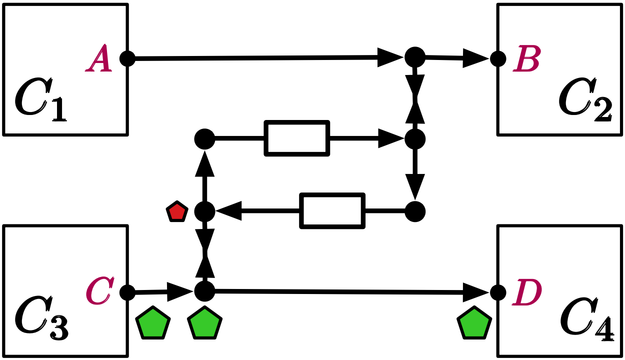
Example: Reo network for a sequencer

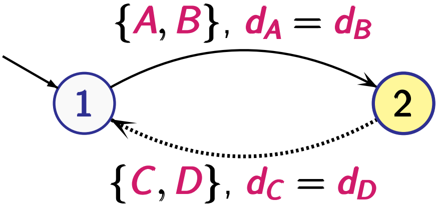


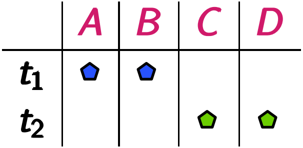


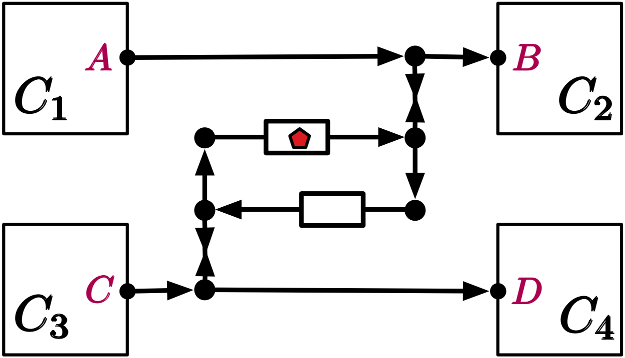
	A	B	C	D
t_1	⬟	⬟		

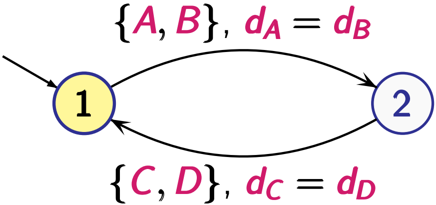
Example: Reo network for a sequencer





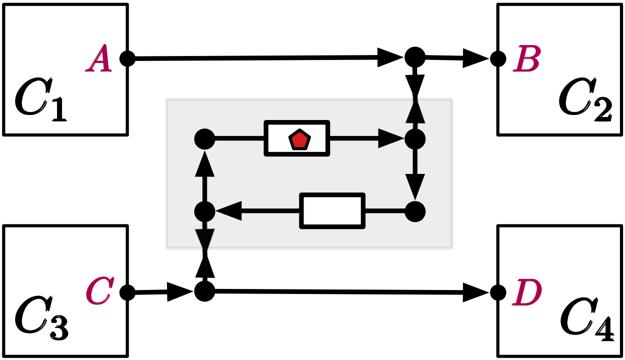


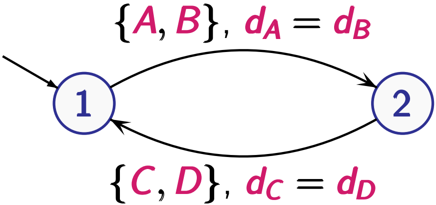




	A	B	C	D
t_1	⬟	⬟		
t_2			⬟	⬟

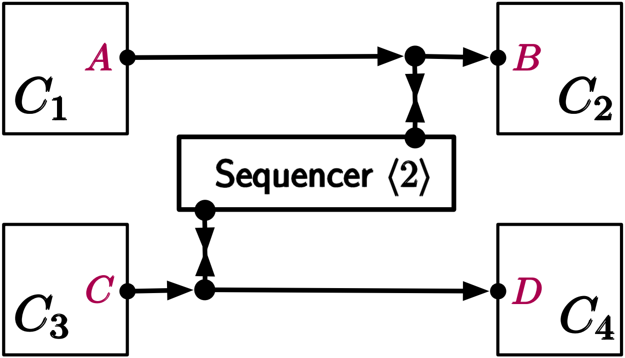
Example: Reo network for a sequencer

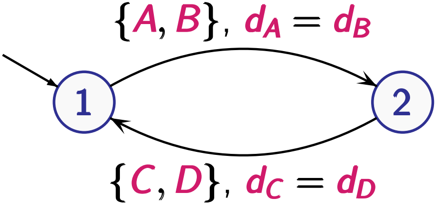




	A	B	C	D
t_1	⬠	⬠		
t_2			⬠	⬠

Example: Reo network for a sequencer





	A	B	C	D
t_1	⬟	⬟		
t_2			⬟	⬟

Reo's hiding operator

serves to build a new component or connector
by "hiding internal ports"

serves to build a new component or connector
by "hiding internal ports"

hiding operator for constraint automata:

given a CA $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$ and port $A \in \mathcal{N}$

$$\mathcal{A} \setminus A \stackrel{\text{def}}{=} (Q, \mathcal{N} \setminus \{A\}, \longrightarrow, Q_0) \text{ where } \quad \ldots$$

serves to build a new component or connector by "hiding internal ports"

hiding operator for constraint automata:

given a CA $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$ and port $A \in \mathcal{N}$

$$\mathcal{A} \setminus A \stackrel{\text{def}}{=} (Q, \mathcal{N} \setminus \{A\}, \longrightarrow, Q_0) \text{ where}$$

$$\frac{q \xrightarrow{N,g} p \text{ in } \mathcal{A}}{q \xrightarrow{N',g'} p \text{ in } \mathcal{A} \setminus A} \qquad \begin{aligned} N' &= N \setminus \{A\} \\ g' &= \exists d.g[d_A/d] \end{aligned}$$

serves to build a new component or connector
by "hiding internal ports"

hiding operator for constraint automata:

given a CA $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0)$ and port $A \in \mathcal{N}$

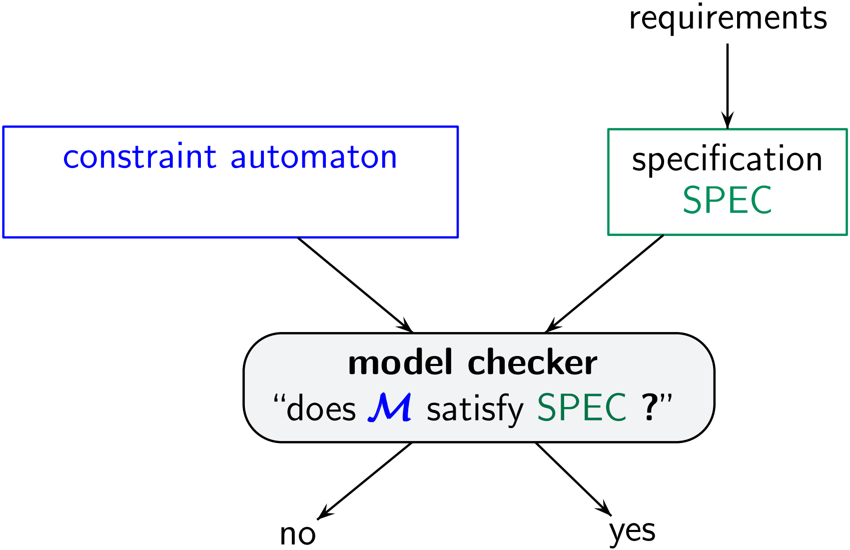
$$\mathcal{A} \setminus A \stackrel{\text{def}}{=} (Q, \mathcal{N} \setminus \{A\}, \longrightarrow, Q_0) \text{ where}$$

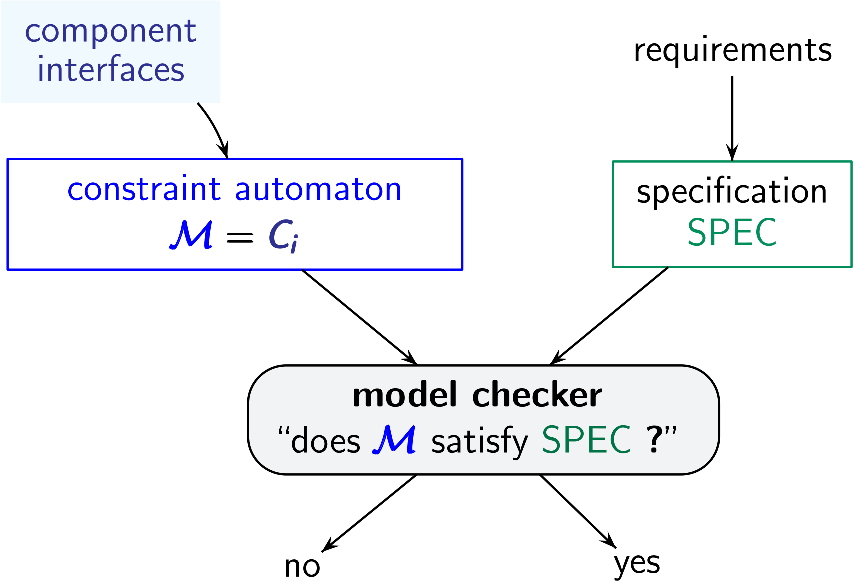
$$\frac{q \xrightarrow{N, g} p \quad \text{in } \mathcal{A}}{q \xrightarrow{N', g'} p \quad \text{in } \mathcal{A} \setminus A} \qquad \begin{array}{lcl} N' & = & N \setminus \{A\} \\ g' & = & \exists d. g[d_A/d] \end{array}$$

variant: additionally transitive closure over the "empty" (internal) transitions

1. Modelling components and connectors
   - constraint automata (CA)
   - coordination language Reo
2. **Model checking with CA**
   - Linear Temporal Logic
   - Alternating Stream Logic
3. Synthesis of connectors

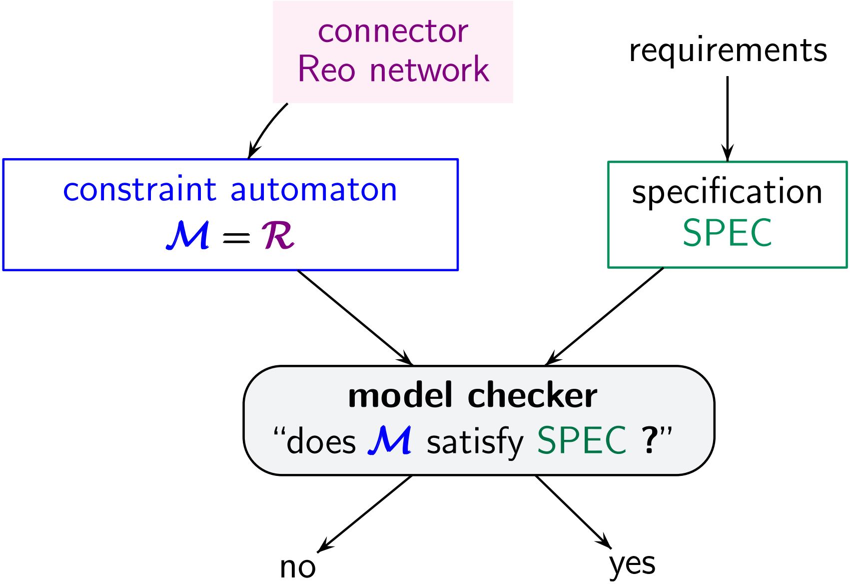
Model checking components and connectors

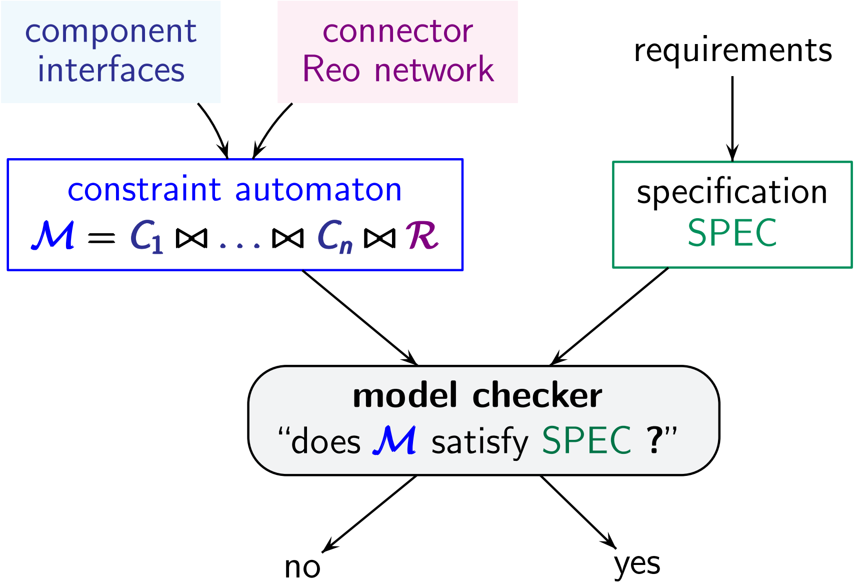




component interfaces
requirements
constraint automaton
$\mathcal{M} = C_i$
specification
SPEC
model checker
"does $\mathcal{M}$ satisfy SPEC ?"
no
yes

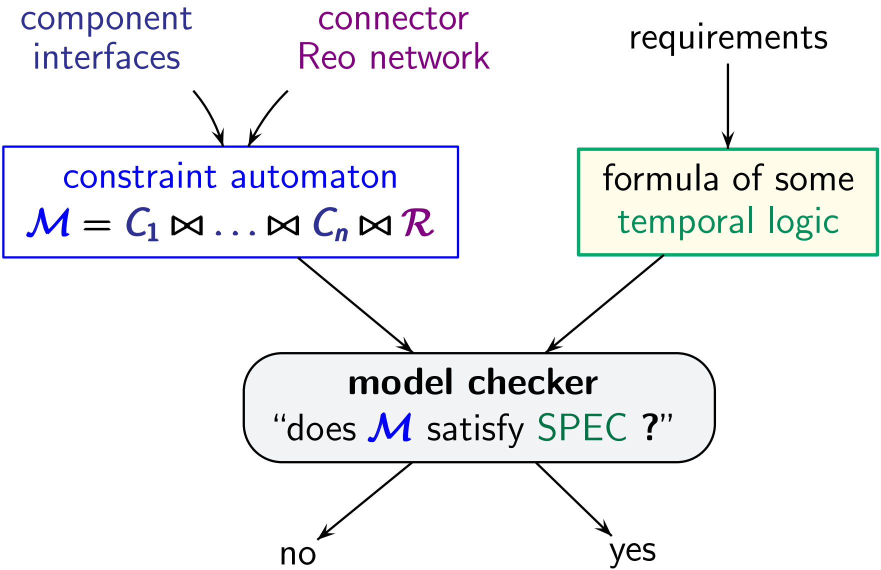
Model checking components and connectors





component interfaces
connector Reo network
requirements
constraint automaton
$\mathcal{M} = C_1 \bowtie \ldots \bowtie C_n \bowtie \mathcal{R}$
specification
SPEC
model checker
"does $\mathcal{M}$ satisfy SPEC **?**"
no
yes

Model checking components and connectors



Overview

1 Modelling components and connectors

constraint automata (CA)
coordination language Reo

2 **Model checking with CA**

Linear Temporal Logic $\longleftarrow$
Alternating Stream Logic

3 Synthesis of connectors

Linear Temporal Logic ($\mathsf{LTL}_{\mathsf{IO}}$)

linear-time logic LTL with I/O-constraints,
adapted to the CA framework

- linear temporal logic (LTL) [Pnueli '77]

  temporal modalities over atomic propositions
  to express safety, liveness properties

- regular expressions for specifying conditions on the interactions of components/connectors

  similar to dynamic LTL [Henriksen,Thiagarajan'99]

Linear Temporal Logic ($LTL_{\bigcirc}$)

$$\varphi \ ::= \ \mathit{true} \ \Big| \ a \ \Big| \ \varphi_1 \wedge \varphi_2 \ \Big| \ \neg\varphi$$

Linear Temporal Logic (LTL_{IO})

$$\varphi \ ::= \ \textit{true} \ \big| \ a \ \big| \ \varphi_1 \wedge \varphi_2 \ \big| \ \neg\varphi$$

atomic
proposition

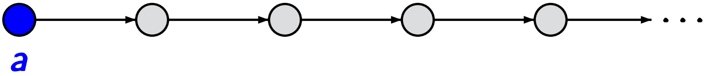
$$\varphi \;::=\; \textit{true} \;\Big|\; a \;\Big|\; \varphi_1 \wedge \varphi_2 \;\Big|\; \neg\varphi$$

atomic proposition

i.e., local condition on the states of a CA

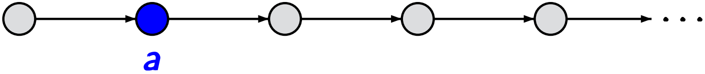
"FIFO buffer is full"

"$x < 3$ for integer variable x"



$$\varphi \;::=\; \mathit{true} \;\Big|\; \underset{\substack{\uparrow \\ \text{atomic} \\ \text{proposition}}}{a} \;\Big|\; \varphi_1 \wedge \varphi_2 \;\Big|\; \neg\varphi \;\Big|\; \underset{\substack{\uparrow \\ \text{next}}}{\bigcirc}\, \varphi$$

$\varphi = \bigcirc a$:



Linear Temporal Logic ($\mathrm{LTL}_{\mathrm{IO}}$)

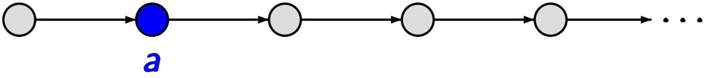
$$\varphi \;::=\; \mathit{true} \;\big|\; a \;\big|\; \varphi_1 \wedge \varphi_2 \;\big|\; \neg\varphi \;\big|\; \bigcirc\varphi \;\big|\; \varphi_1 \,\mathsf{U}\, \varphi_2$$

atomic proposition

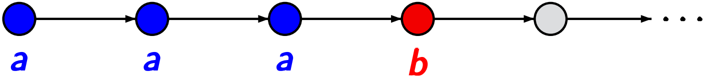
next

until

$\varphi = \bigcirc a$:



$\varphi = a \,\mathsf{U}\, b$:



$$\varphi \ ::= \ \mathit{true} \ \big| \ a \ \big| \ \varphi_1 \wedge \varphi_2 \ \big| \ \neg\varphi \ \big| \ \bigcirc \varphi \ \big| \ \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

atomic proposition

next

until

U^{α} $\widehat{=}$ until indexed by a stream expression α

Linear Temporal Logic (LTL_{IO})

$$\varphi \; ::= \; \textit{true} \;\big|\; a \;\big|\; \varphi_1 \wedge \varphi_2 \;\big|\; \neg\varphi \;\big|\; \bigcirc \varphi \;\big|\; \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

atomic proposition (a), next ($\bigcirc$), until (U^{α})

U^{α} $\widehat{=}$ until indexed by a stream expression α, i.e.,

regular expression specifiying a set of finite data streams

$$\alpha \; ::= \; \textit{ioc} \;\big|\; \alpha_1 ; \alpha_2 \;\big|\; \alpha_1 \cup \alpha_2 \;\big|\; \alpha^*$$

$$\varphi \;::=\; \textit{true} \;\Big|\; a \;\Big|\; \varphi_1 \wedge \varphi_2 \;\Big|\; \neg\varphi \;\Big|\; \bigcirc \varphi \;\Big|\; \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

atomic proposition

next

until

$\mathsf{U}^{\alpha} \;\widehat{=}\;$ until indexed by a stream expression α, i.e.,

regular expression specifiying a set of finite data streams

$$\alpha \;::=\; \textit{ioc} \;\Big|\; \alpha_1 ; \alpha_2 \;\Big|\; \alpha_1 \cup \alpha_2 \;\Big|\; \alpha^*$$

I/O-constraint, i.e., Boolean condition that specifies conditions on the active ports and their I/O-operations

$$\varphi \ ::= \ \textit{true} \ \big| \ a \ \big| \ \varphi_1 \wedge \varphi_2 \ \big| \ \neg\varphi \ \big| \ \bigcirc \varphi \ \big| \ \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

atomic proposition (a) — next ($\bigcirc$) — until (U^{α})

U^{α} $\widehat{=}$ until indexed by a stream expression α, i.e.,

regular expression specifiying a set of finite data streams

$$\alpha \ ::= \ \textit{ioc} \ \big| \ \alpha_1 ; \alpha_2 \ \big| \ \alpha_1 \cup \alpha_2 \ \big| \ \alpha^*$$

semantics of α: regular language $\mathcal{L}(\alpha)$
(set of finite data streams)

Linear Temporal Logic (LTL_{IO})

$$\varphi \;::=\; \textit{true} \;\mid\; a \;\mid\; \varphi_1 \wedge \varphi_2 \;\mid\; \neg\varphi \;\mid\; \bigcirc\,\varphi \;\mid\; \varphi_1\,\mathsf{U}^{\alpha}\,\varphi_2$$

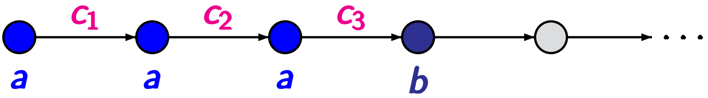
atomic proposition

next

until

U^{α} $\hat{=}$ until indexed by a stream expression α, i.e.,

$\varphi = a\,\mathsf{U}^{\alpha}\,b$ $\qquad$ $c_1\,c_2\,c_3 \in \mathcal{L}(\alpha)$



$$\varphi \ ::= \ \mathit{true} \ \mid \ a \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \neg\varphi \ \mid \ \bigcirc \varphi \ \mid \ \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

atomic proposition

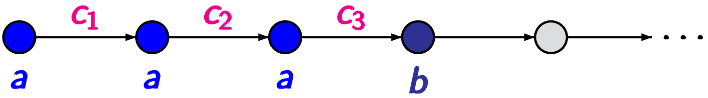
next

until

$\mathsf{U}^{\alpha} \ \widehat{=}$ until indexed by a stream expression α, i.e.,

$\varphi = a \, \mathsf{U}^{\alpha} \, b$

$c_1 \, c_2 \, c_3 \in \mathcal{L}(\alpha)$



"standard until": $\varphi_1 \, \mathsf{U} \, \varphi_2 \ \widehat{=} \ \varphi_1 \, \mathsf{U}^{\mathit{true}^*} \, \varphi_2$

Derived operators in $\mathsf{LTL}_{\mathsf{IO}}$

$$\varphi \ ::= \ \textit{true} \ \big| \ a \ \big| \ \varphi_1 \wedge \varphi_2 \ \big| \ \neg\varphi \ \big| \ \bigcirc \varphi \ \big| \ \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

derived operators:

$\vee, \rightarrow, \ldots$ as usual

$$\varphi \ ::= \ \mathit{true} \ \big| \ a \ \big| \ \varphi_1 \wedge \varphi_2 \ \big| \ \neg\varphi \ \big| \ \bigcirc \varphi \ \big| \ \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

derived operators:

$\vee, \rightarrow, \ldots$ as usual

$$\Diamond\varphi \ \stackrel{\text{def}}{=} \ \mathit{true} \, \mathsf{U} \, \varphi \qquad \text{eventually}$$

Derived operators in $\mathrm{LTL}_{\mathrm{IO}}$

$$\varphi \ ::= \ \mathit{true} \ \big| \ a \ \big| \ \varphi_1 \wedge \varphi_2 \ \big| \ \neg\varphi \ \big| \ \bigcirc \varphi \ \big| \ \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

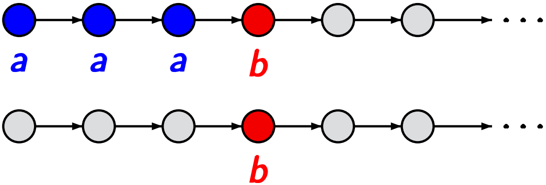
derived operators:

$\vee, \rightarrow, \ldots$ as usual

$$\Diamond\varphi \ \stackrel{\text{def}}{=} \ \mathit{true} \, \mathsf{U} \, \varphi \quad \text{eventually}$$

until operator
$a \, \mathsf{U} \, b$

eventually
$\Diamond b$



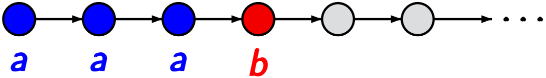
$$\varphi \ ::= \ \textit{true} \ \big| \ a \ \big| \ \varphi_1 \wedge \varphi_2 \ \big| \ \neg\varphi \ \big| \ \bigcirc\varphi \ \big| \ \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

derived operators:

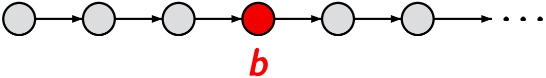
$\vee, \rightarrow, \ldots$ as usual

$\Diamond\varphi$	$\stackrel{\text{def}}{=}$	$\textit{true}\,\mathsf{U}\,\varphi$	eventually
$\Box\varphi$	$\stackrel{\text{def}}{=}$	$\neg\Diamond\neg\varphi$	always

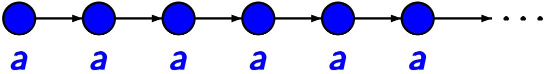
until operator
$a\,\mathsf{U}\,b$



eventually
$\Diamond b$



always
$\Box a$



Derived operators in $\mathrm{LTL}_{\mathrm{IO}}$

$$\varphi \;::=\; \textit{true} \;\Big|\; a \;\Big|\; \varphi_1 \wedge \varphi_2 \;\Big|\; \neg\varphi \;\Big|\; \bigcirc\varphi \;\Big|\; \varphi_1 \,\mathrm{U}^{\alpha}\, \varphi_2$$

derived operators:

$\vee, \rightarrow, \ldots$ as usual

$\Diamond\varphi$	$\stackrel{\text{def}}{=}$	$\textit{true}\,\mathrm{U}\,\varphi$	eventually
$\Box\varphi$	$\stackrel{\text{def}}{=}$	$\neg\Diamond\neg\varphi$	always

infinitely often $\Box\Diamond\varphi$

eventually forever $\Diamond\Box\varphi$

Derived operators in $\mathsf{LTL}_{\mathsf{IO}}$

$$\varphi \; ::= \; \mathit{true} \;\big|\; a \;\big|\; \varphi_1 \wedge \varphi_2 \;\big|\; \neg\varphi \;\big|\; \bigcirc \varphi \;\big|\; \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

$$\langle\alpha\rangle\varphi \;\stackrel{\text{def}}{=}\; \mathit{true} \, \mathsf{U}^{\alpha} \, \varphi$$

Derived operators in $\mathsf{LTL}_{\mathsf{IO}}$

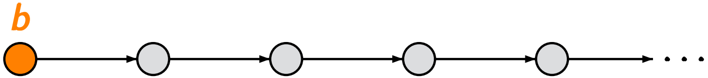
$$\varphi \ ::= \ \textit{true} \ \big| \ a \ \big| \ \varphi_1 \wedge \varphi_2 \ \big| \ \neg\varphi \ \big| \ \bigcirc \varphi \ \big| \ \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

$\langle\alpha\rangle\varphi \ \stackrel{\text{def}}{=} \ \textit{true}\,\mathsf{U}^{\alpha}\,\varphi$ "there exists a prefix s.t. ..."

$$\varphi \ ::= \ \mathit{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

$$\langle\alpha\rangle\varphi \ \stackrel{\text{def}}{=} \ \mathit{true}\, \mathsf{U}^{\alpha}\, \varphi$$ "there exists a prefix s.t. ..."

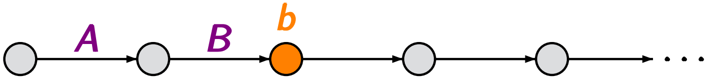
$\varphi = \langle (A; B)^* \rangle b$:



$$\varphi \ ::= \ \mathit{true} \ \big| \ a \ \big| \ \varphi_1 \wedge \varphi_2 \ \big| \ \neg\varphi \ \big| \ \bigcirc \varphi \ \big| \ \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

$\langle\alpha\rangle\varphi \ \stackrel{\text{def}}{=} \ \mathit{true}\,\mathsf{U}^{\alpha}\,\varphi$ "there exists a prefix s.t. ..."

$\varphi = \langle(A;B)^*\rangle b$:



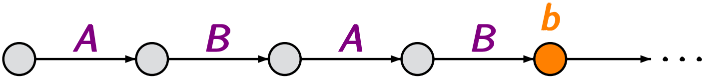
Derived operators in $\mathsf{LTL}_{\mathsf{IO}}$

550

$$\varphi \;::=\; \mathit{true} \;\big|\; a \;\big|\; \varphi_1 \wedge \varphi_2 \;\big|\; \neg\varphi \;\big|\; \bigcirc\varphi \;\big|\; \varphi_1 \,\mathsf{U}^{\alpha}\, \varphi_2$$

$\langle\alpha\rangle\varphi \;\stackrel{\text{def}}{=}\; \mathit{true}\,\mathsf{U}^{\alpha}\,\varphi$ “there exists a prefix s.t. ...”

$\varphi = \langle(A;B)^*\rangle b$:



Derived operators in $\mathsf{LTL}_{\mathsf{IO}}$

$$\varphi \ ::= \ \textit{true} \ \big| \ a \ \big| \ \varphi_1 \wedge \varphi_2 \ \big| \ \neg\varphi \ \big| \ \bigcirc \varphi \ \big| \ \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

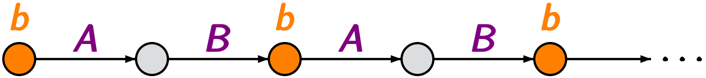
$\langle\alpha\rangle\varphi$	$\stackrel{\text{def}}{=}$	$\textit{true}\, \mathsf{U}^{\alpha}\, \varphi$	"there exists a prefix s.t. ..."
$[\alpha]\varphi$	$\stackrel{\text{def}}{=}$	$\neg\langle\alpha\rangle\neg\varphi$	"for all prefixes ..."

Derived operators in $\mathsf{LTL}_{\mathsf{IO}}$

$$\varphi \; ::= \; \textit{true} \; \big| \; a \; \big| \; \varphi_1 \wedge \varphi_2 \; \big| \; \neg\varphi \; \big| \; \bigcirc \varphi \; \big| \; \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

$\langle\alpha\rangle\varphi$	$\stackrel{\text{def}}{=}$	$\textit{true} \, \mathsf{U}^{\alpha} \, \varphi$	"there exists a prefix s.t. ..."
$[\alpha]\varphi$	$\stackrel{\text{def}}{=}$	$\neg\langle\alpha\rangle\neg\varphi$	"for all prefixes ..."

$\varphi = [(A;B)^*]b$:

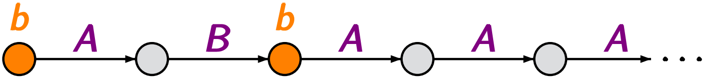


Derived operators in $\mathsf{LTL}_{\mathsf{IO}}$

$$\varphi \ ::= \ \mathit{true} \ \big| \ a \ \big| \ \varphi_1 \wedge \varphi_2 \ \big| \ \neg\varphi \ \big| \ \bigcirc \varphi \ \big| \ \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2$$

$\langle\alpha\rangle\varphi$	$\stackrel{\text{def}}{=}$	$\mathit{true}\,\mathsf{U}^{\alpha}\,\varphi$	"there exists a prefix s.t. ..."
$[\alpha]\varphi$	$\stackrel{\text{def}}{=}$	$\neg\langle\alpha\rangle\neg\varphi$	"for all prefixes ..."

$\varphi = [(A;B)^*]b$:



given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ in a CA:

Semantics of $\mathrm{LTL_{IO}}$ formulas

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ in a CA:

$\pi \models \textit{true}$

$\pi \models a$ iff atomic proposition a holds for q_0

Semantics of $\mathrm{LTL}_{\mathrm{IO}}$ formulas

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ in a CA:

$\pi \models \mathit{true}$		
$\pi \models a$	iff	atomic proposition a holds for q_0
$\pi \models \varphi_1 \vee \varphi_2$	iff	$\pi \models \varphi_1$ or $\pi \models \varphi_2$
$\pi \models \neg\varphi$	iff	$\pi \not\models \varphi$

Semantics of LTL_{IO} formulas

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ in a CA:

$\pi \models \textit{true}$		
$\pi \models a$	iff	atomic proposition a holds for q_0
$\pi \models \varphi_1 \vee \varphi_2$	iff	$\pi \models \varphi_1$ or $\pi \models \varphi_2$
$\pi \models \neg\varphi$	iff	$\pi \not\models \varphi$
$\pi \models \bigcirc\varphi$	iff	$\textit{suffix}(\pi, 1) \models \varphi$

$$\textit{suffix}(\pi, 1) = q_1 \xrightarrow{c_2} q_2 \xrightarrow{c_3} \ldots$$

Semantics of $\mathsf{LTL_{IO}}$ formulas

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ in a CA:

$$
\begin{array}{lll}
\pi \models \mathit{true} & & \\
\pi \models a & \text{iff} & \text{atomic proposition } a \text{ holds for } q_0 \\
\pi \models \varphi_1 \vee \varphi_2 & \text{iff} & \pi \models \varphi_1 \text{ or } \pi \models \varphi_2 \\
\pi \models \neg\varphi & \text{iff} & \pi \not\models \varphi \\
\pi \models \bigcirc\varphi & \text{iff} & \mathit{suffix}(\pi, 1) \models \varphi \\
\pi \models \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2 & \text{iff} & \text{there exists a finite prefix } \sigma \text{ of } \pi \\
 & & \text{with } \mathit{stream}(\sigma) \in \mathcal{L}(\alpha) \text{ and } \ldots
\end{array}
$$

$$\boxed{\text{if } |\sigma| = n \text{ then } \mathit{stream}(\sigma) = c_1\, c_2 \ldots c_n}$$

Semantics of LTL_{IO} formulas

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ in a CA:

$$
\begin{array}{lll}
\pi \models \mathit{true} & & \\
\pi \models a & \text{iff} & \text{atomic proposition } a \text{ holds for } q_0 \\
\pi \models \varphi_1 \vee \varphi_2 & \text{iff} & \pi \models \varphi_1 \text{ or } \pi \models \varphi_2 \\
\pi \models \neg\varphi & \text{iff} & \pi \not\models \varphi \\
\pi \models \bigcirc\varphi & \text{iff} & \mathit{suffix}(\pi, 1) \models \varphi \\
\pi \models \varphi_1 \, \mathsf{U}^{\alpha} \, \varphi_2 & \text{iff} & \text{there exists a finite prefix } \sigma \text{ of } \pi \\
 & & \text{with } \mathit{stream}(\sigma) \in \mathcal{L}(\alpha) \text{ and} \\
 & & \quad \mathit{suffix}(\pi, |\sigma|) \models \varphi_2 \text{ and} \\
 & & \quad \mathit{suffix}(\pi, i) \models \varphi_1 \text{ for } 0 \leqslant i < |\sigma|
\end{array}
$$

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ in a CA:

$\vdots$

$\pi \models \Diamond\varphi$ iff $suffix(\pi, i) \models \varphi$ for some $i \geqslant 0$

"eventually φ"

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ in a CA:

$$\vdots$$

$\pi \models \Diamond\varphi$ iff $\mathit{suffix}(\pi, i) \models \varphi$ for some $i \geqslant 0$

$\pi \models \Box\varphi$ iff $\mathit{suffix}(\pi, i) \models \varphi$ for all $i \geqslant 0$

“always φ”

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ in a CA:

$$\vdots$$

$\pi \models \Diamond\varphi$ iff $suffix(\pi, i) \models \varphi$ for some $i \geqslant 0$

$\pi \models \Box\varphi$ iff $suffix(\pi, i) \models \varphi$ for all $i \geqslant 0$

$\pi \models \langle\alpha\rangle\varphi$ iff there exists a finite prefix σ of π s.t.
$stream(\sigma) \in \mathcal{L}(\alpha)$ and
$suffix(\pi, |\sigma|) \models \varphi$

$$\boxed{\langle\alpha\rangle\varphi \stackrel{\text{def}}{=} true\, \mathsf{U}^{\alpha}\, \varphi}$$

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ in a CA:

$$\vdots$$

$\pi \models \Diamond\varphi$	iff	$\mathit{suffix}(\pi, i) \models \varphi$ for some $i \geqslant 0$
$\pi \models \Box\varphi$	iff	$\mathit{suffix}(\pi, i) \models \varphi$ for all $i \geqslant 0$
$\pi \models \langle\alpha\rangle\varphi$	iff	there exists a finite prefix σ of π s.t. $\mathit{stream}(\sigma) \in \mathcal{L}(\alpha)$ and $\mathit{suffix}(\pi, \lvert\sigma\rvert) \models \varphi$
$\pi \models [\alpha]\varphi$	iff	for all finite prefixes σ of π we have: if $\mathit{stream}(\sigma) \in \mathcal{L}(\alpha)$ then $\mathit{suffix}(\pi, \lvert\sigma\rvert) \models \varphi$

$\mathsf{LTL}_{\mathsf{IO}}$ model checking

given: finite constraint automaton $\mathcal{A}$
$\mathsf{LTL}_{\mathsf{IO}}$ formula φ

question: does $\mathcal{A} \models \varphi$ hold **?**

$\mathsf{LTL}_{\mathsf{IO}}$ model checking

given: finite constraint automaton $\mathcal{A}$
$\mathsf{LTL}_{\mathsf{IO}}$ formula φ

question: does $\mathcal{A} \models \varphi$ hold ?

$\mathcal{A} \models \varphi$ iff for all executions π in $\mathcal{A}$ we have: $\pi \models \varphi$

$\mathsf{LTL_{IO}}$ model checking

given: finite constraint automaton $\mathcal{A}$
$\mathsf{LTL_{IO}}$ formula φ

question: does $\boxed{\mathcal{A} \models \varphi}$ hold **?**

$\mathcal{A} \models \varphi$ iff for all executions π in $\mathcal{A}$ we have: $\pi \models \varphi$

iff there is no execution π of $\mathcal{A}$ such that $\pi \models \neg\varphi$

constraint automaton $\mathcal{A}$

LTL$_{\text{IO}}$ formula φ

constraint automaton $\mathcal{A}$

LTL$_{\mathrm{IO}}$ formula φ

nondeterministic Büchi automaton $\mathcal{B}$ for $\neg\varphi$

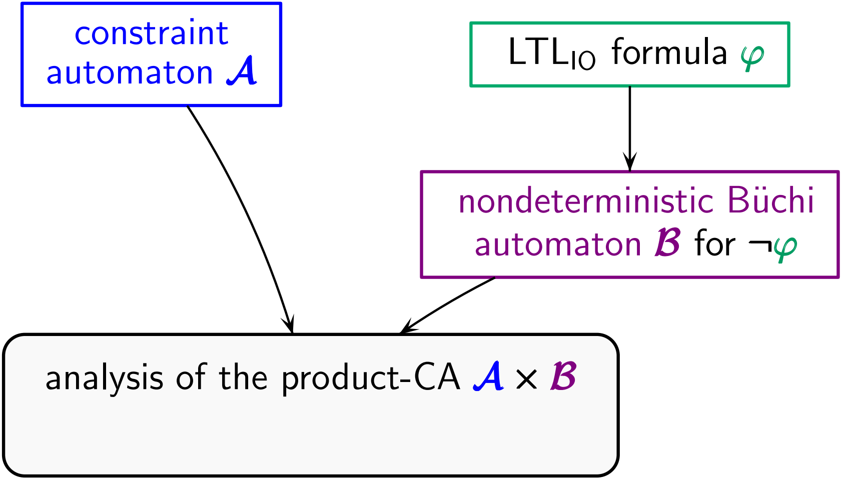
constraint automaton $\mathcal{A}$

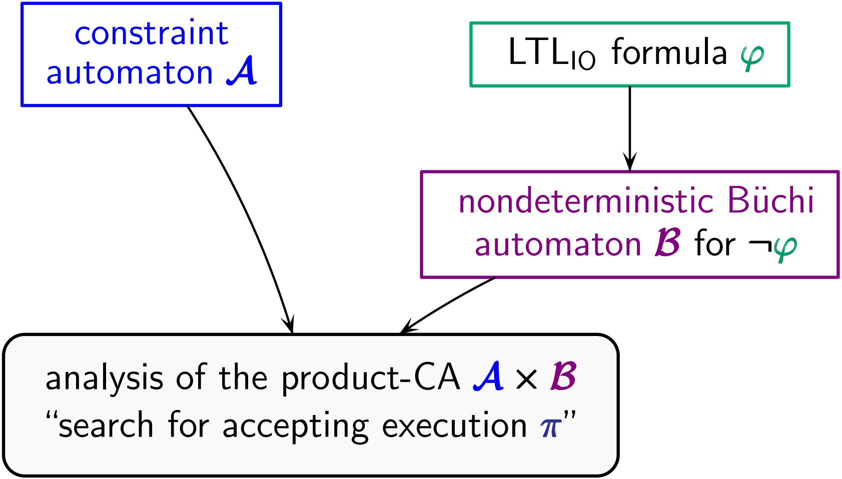
LTL$_{\mathrm{IO}}$ formula φ

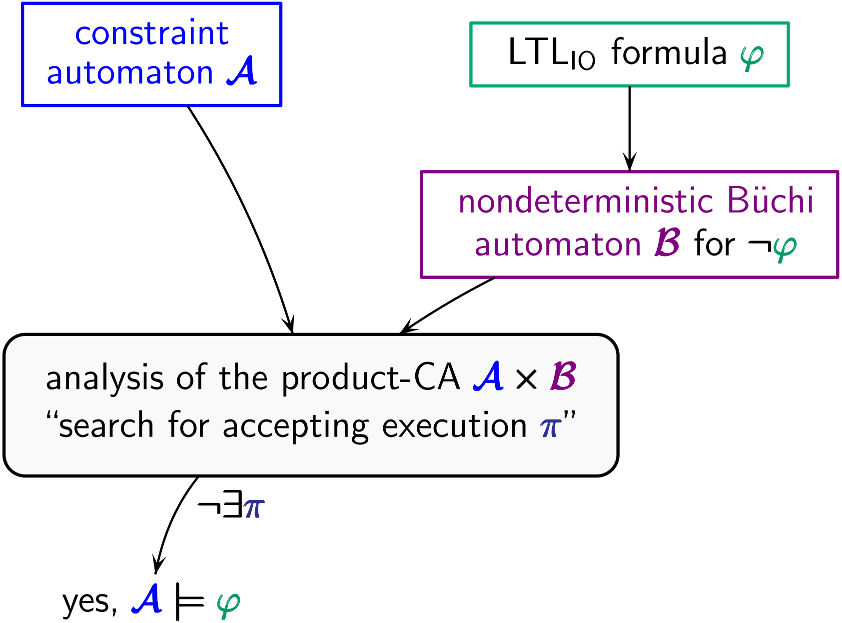
nondeterministic Büchi automaton $\mathcal{B}$ for $\neg\varphi$

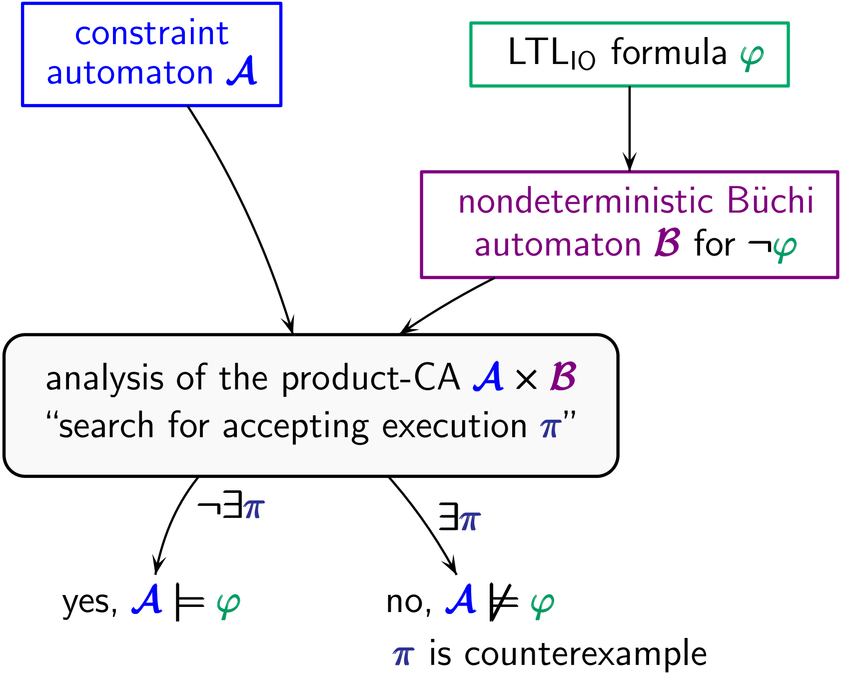
NBA like non-deterministic finite automaton (NFA) with set of accepting states F

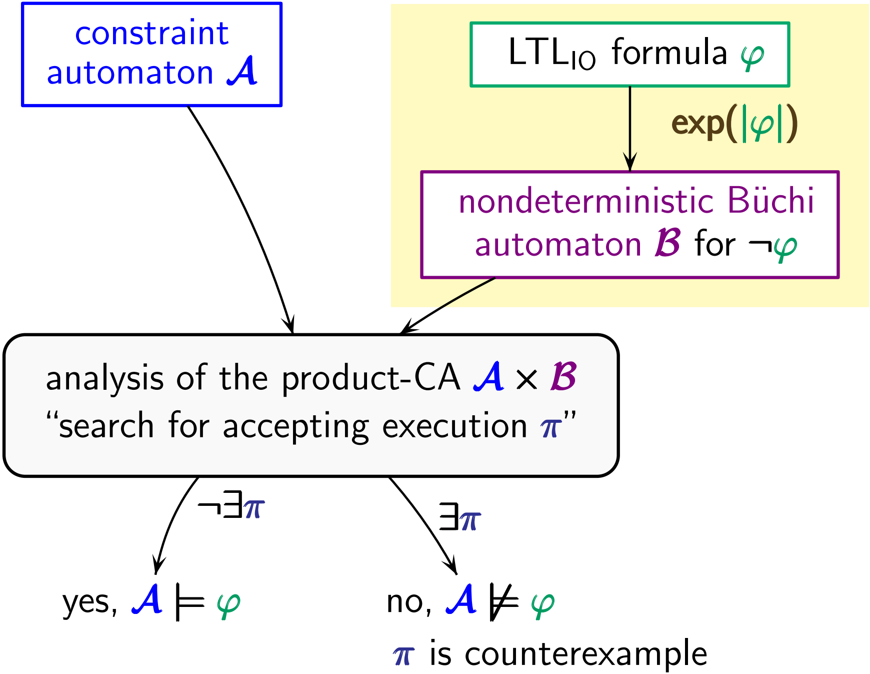
run $q_0\, q_1\, q_2 \ldots$ for an infinite input string $c_1\, c_2\, c_3 \ldots$ is accepting iff $q_i \in F$ for infinitely many $i \geqslant 0$

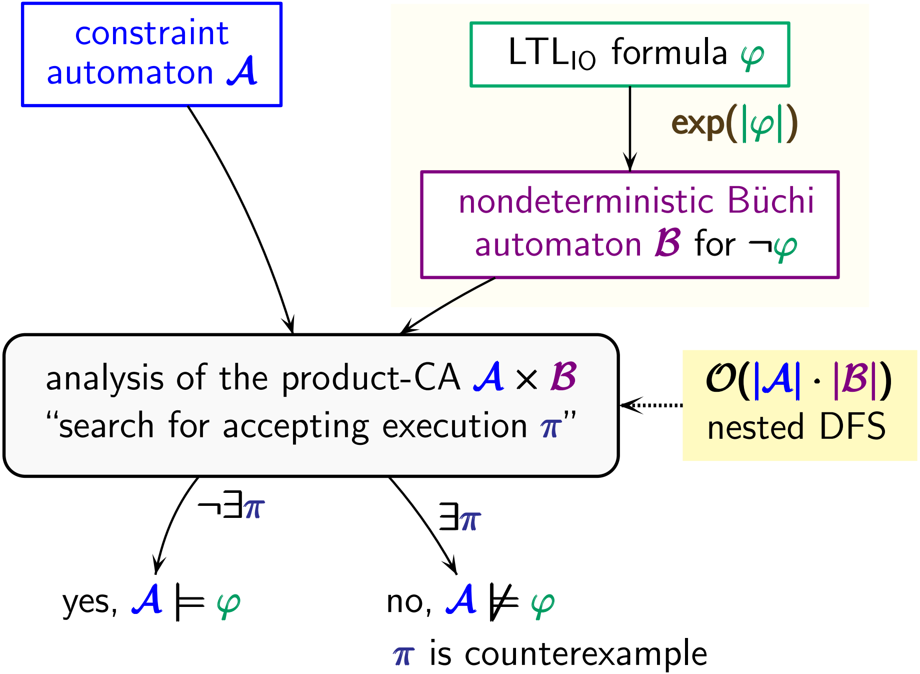

constraint automaton $\mathcal{A}$
$\mathrm{LTL}_{\mathrm{IO}}$ formula φ
nondeterministic Büchi automaton $\mathcal{B}$ for $\neg\varphi$
analysis of the product-CA $\mathcal{A} \times \mathcal{B}$


constraint automaton $\mathcal{A}$
LTL_{IO} formula φ
nondeterministic Büchi automaton $\mathcal{B}$ for $\neg\varphi$
analysis of the product-CA $\mathcal{A} \times \mathcal{B}$
"search for accepting execution π"

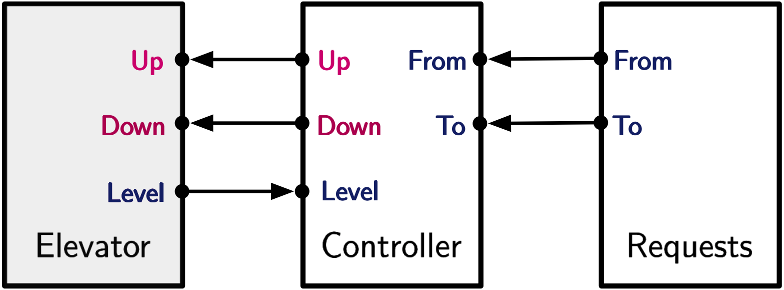

constraint automaton $\mathcal{A}$
$\mathrm{LTL}_{\mathrm{IO}}$ formula φ
nondeterministic Büchi automaton $\mathcal{B}$ for $\neg\varphi$
analysis of the product-CA $\mathcal{A} \times \mathcal{B}$
"search for accepting execution π"
$\neg\exists\pi$
yes, $\mathcal{A} \models \varphi$


constraint automaton $\mathcal{A}$
LTL$_{IO}$ formula φ
nondeterministic Büchi automaton $\mathcal{B}$ for $\neg\varphi$
analysis of the product-CA $\mathcal{A} \times \mathcal{B}$
"search for accepting execution π"
$\neg\exists\pi$
$\exists\pi$
yes, $\mathcal{A} \models \varphi$
no, $\mathcal{A} \not\models \varphi$
π is counterexample


constraint automaton $\mathcal{A}$
LTL$_{IO}$ formula φ
$\exp(|\varphi|)$
nondeterministic Büchi automaton $\mathcal{B}$ for $\neg\varphi$
analysis of the product-CA $\mathcal{A} \times \mathcal{B}$
“search for accepting execution π”
$\neg\exists\pi$
$\exists\pi$
yes, $\mathcal{A} \models \varphi$
no, $\mathcal{A} \not\models \varphi$
π is counterexample

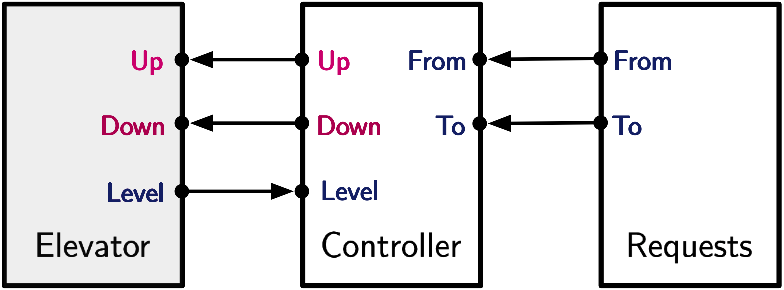

constraint automaton $\mathcal{A}$
$\mathrm{LTL}_{\mathrm{IO}}$ formula φ
$\exp(|\varphi|)$
nondeterministic Büchi automaton $\mathcal{B}$ for $\neg\varphi$
analysis of the product-CA $\mathcal{A} \times \mathcal{B}$
"search for accepting execution π"
$\mathcal{O}(|\mathcal{A}| \cdot |\mathcal{B}|)$
nested DFS
$\neg\exists\pi$
$\exists\pi$
yes, $\mathcal{A} \models \varphi$
no, $\mathcal{A} \not\models \varphi$
π is counterexample

Example: elevator system



“elevator does not move unless **Up** or **Down**”

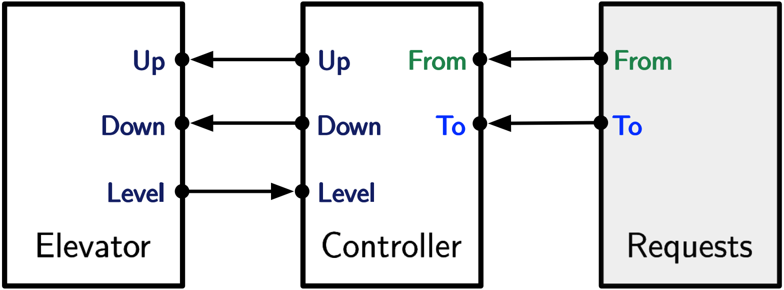
Example: elevator system



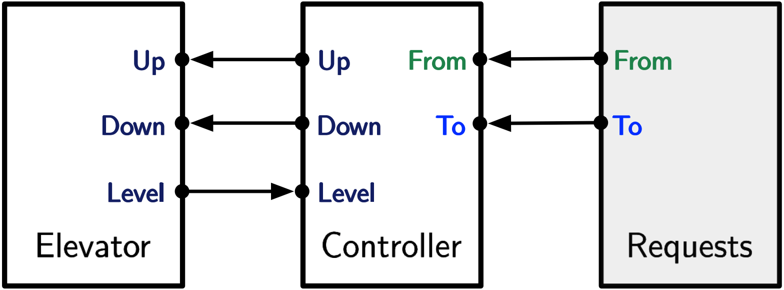
"elevator does not move unless **Up** or **Down**"

$$\bigwedge_{1 \leqslant i \leqslant k} \Big(\Box (\text{ "elevator at } i\text{"} \longrightarrow [(\neg \mathsf{Up} \wedge \neg \mathsf{Down})^*] \text{ "elevator at } i\text{" }) \Big)$$

Example: elevator system



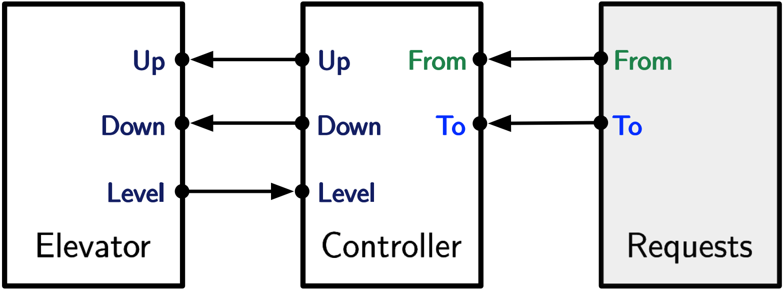
"each request is eventually served"



"each request is eventually served"

$$\bigwedge_{\substack{1\leqslant i\leqslant k\\1\leqslant j\leqslant k}} \Big(\Box\big(\langle \mathsf{From} \wedge \mathsf{To} \wedge d_{\mathsf{From}}{=}i \wedge d_{\mathsf{To}}{=}j\rangle \textit{true} \longrightarrow \Diamond(\text{“elevator at } i\text{”} \wedge \Diamond \text{“elevator at } j\text{”})\big)\Big)$$

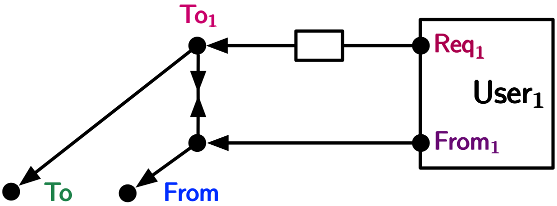
Refined request component



consider a refined version of the request component:

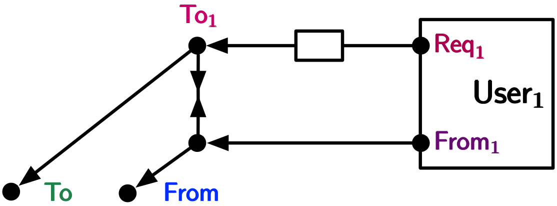
- one component for the users at each level
- port signature of user at level i:

  output port Req_i, input port From_i

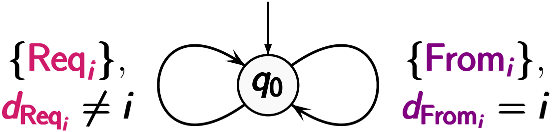

To_1
Req_1
User_1
From_1
To
From

Refined request component

610

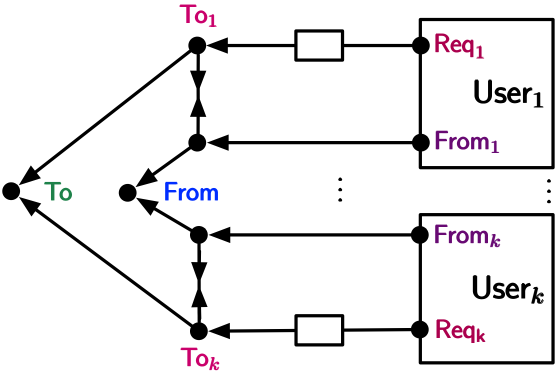


CA for **User**$_i$ at level i:

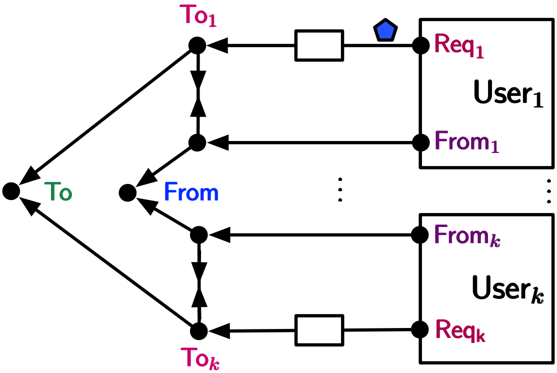


nondeterministic choice of target level

Refined request component

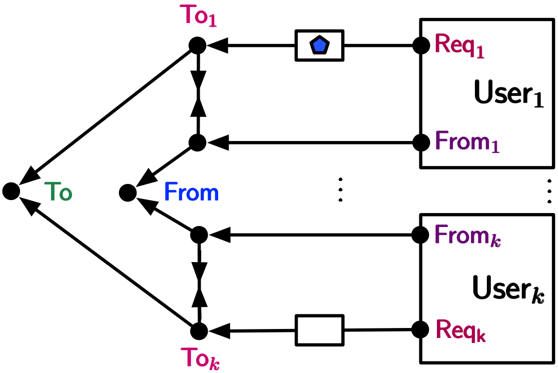


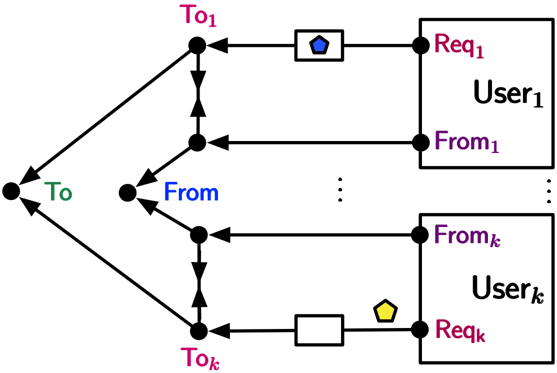
Refined request component



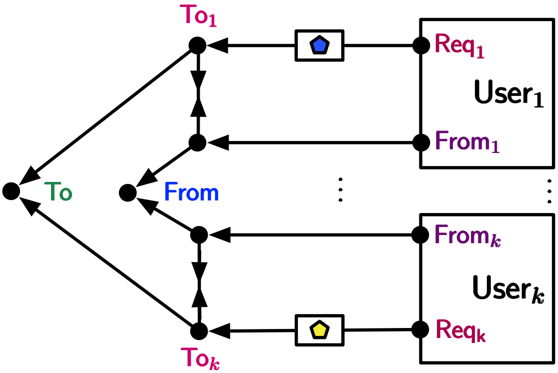
Refined request component

610



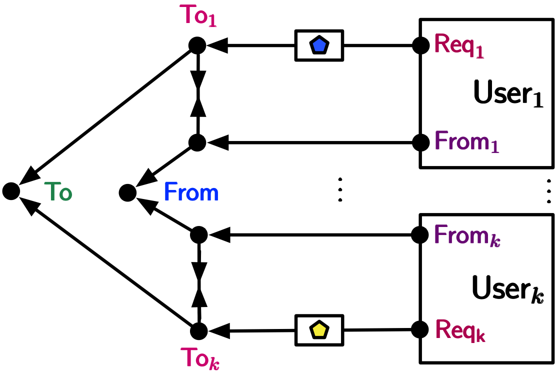


To_1
Req_1
User_1
From_1
To
From
From_k
User_k
Req_k
To_k



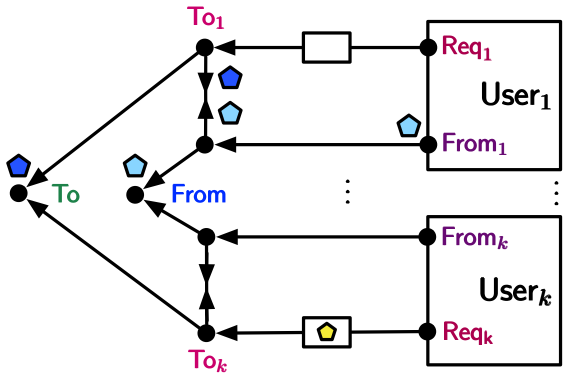
To1
Req1
User1
From1
To
From
Fromk
Userk
Reqk
Tok

Refined request component

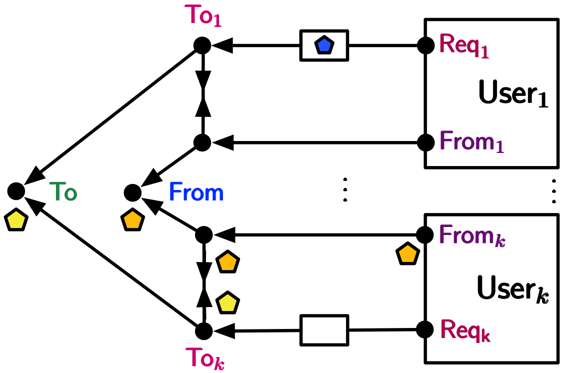


nondeterministic merge at node To between the user-requests submitted via nodes $\mathsf{To}_1, \mathsf{To}_2, \ldots, \mathsf{To}_k$

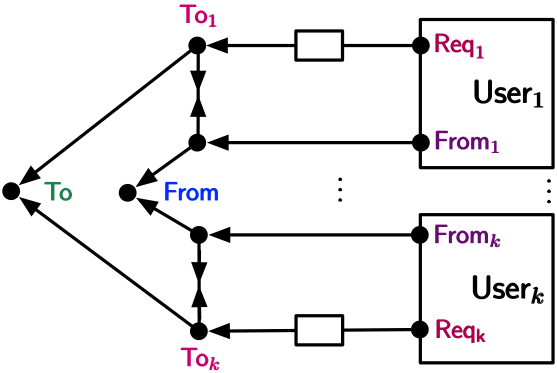
Refined request component



nondeterministic merge at node To between the user-requests submitted via nodes $To_1, To_2, \ldots, To_k$

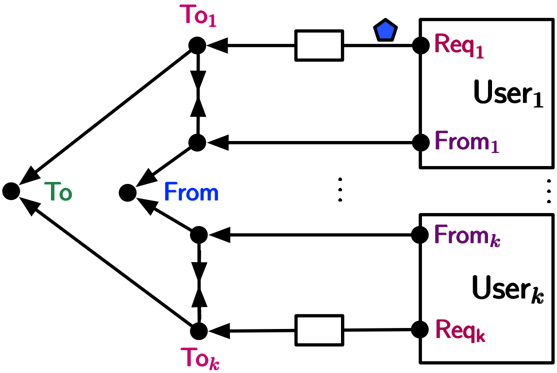


nondeterministic merge at node To between the user-requests submitted via nodes $\mathsf{To}_1, \mathsf{To}_2, \ldots, \mathsf{To}_k$



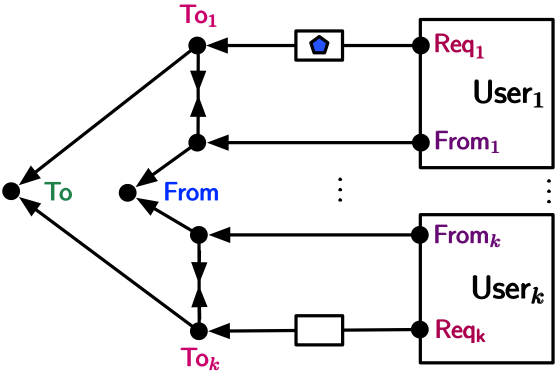
$$\bigwedge_{\substack{1\leqslant i\leqslant k\\ 1\leqslant j\leqslant k}} \Big(\square \big(\langle \mathsf{Req}_i \wedge d_{\mathsf{Req}_i}{=}j \rangle \mathit{true} \longrightarrow \Diamond\big(\text{“elevator at } i\text{”} \wedge \Diamond \text{ “elevator at } j\text{”} \big) \big) \Big) \; ?$$

Each user will be served ?



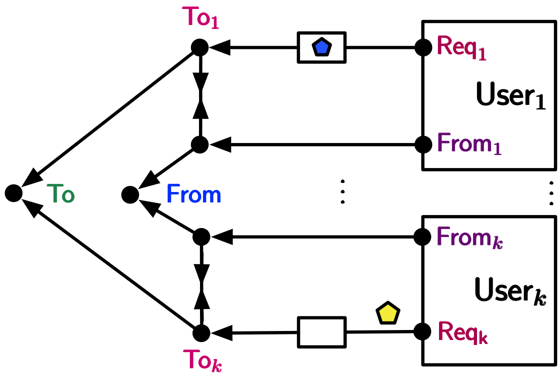
$$\bigwedge_{\substack{1\leqslant i\leqslant k\\ 1\leqslant j\leqslant k}} \Big(\Box\big(\langle \mathrm{Req}_i \wedge d_{\mathrm{Req}_i}{=}j \rangle \mathit{true} \longrightarrow \Diamond(\text{“elevator at } i\text{”} \wedge \Diamond \text{“elevator at } j\text{”}) \big) \Big) \ ?$$

Each user will be served ?

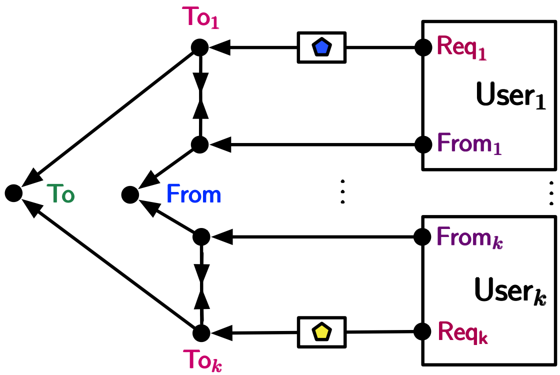


$$\bigwedge_{\substack{1\leqslant i\leqslant k\\1\leqslant j\leqslant k}}\Big(\Box\big(\langle\, \mathsf{Req}_i \wedge d_{\mathsf{Req}_i}{=}j\,\rangle \mathit{true} \longrightarrow \Diamond(\text{ “elevator at } i\text{” } \wedge \ \Diamond \text{ “elevator at } j\text{” })\big)\Big) \ ?$$

Each user will be served ?

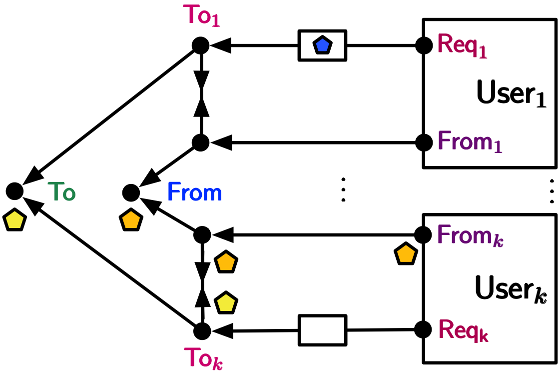


$$\bigwedge_{\substack{1\leqslant i\leqslant k\\1\leqslant j\leqslant k}} \Big(\Box \big(\langle \mathsf{Req}_i \wedge d_{\mathsf{Req}_i}{=}j \rangle \mathit{true} \longrightarrow \Diamond(\text{“elevator at } i\text{”} \wedge \Diamond \text{“elevator at } j\text{”}) \big) \Big) \; ?$$

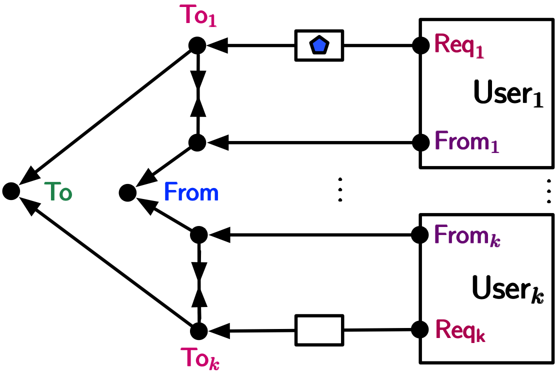


$$\bigwedge_{\substack{1\leqslant i\leqslant k\\ 1\leqslant j\leqslant k}} \Big(\Box\big(\langle \mathsf{Req}_i \wedge d_{\mathsf{Req}_i}{=}j \rangle \textit{true} \longrightarrow \Diamond(\text{ “elevator at } i\text{” } \wedge\ \Diamond \text{ “elevator at } j\text{” })\big)\Big)\ ?$$

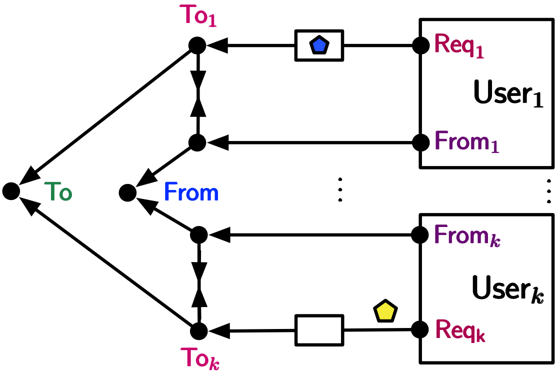
Each user will be served ?



$$\bigwedge_{\substack{1\leqslant i\leqslant k\\ 1\leqslant j\leqslant k}} \Big(\Box\big(\langle \mathsf{Req}_i \wedge d_{\mathsf{Req}_i} = j \rangle \mathit{true} \longrightarrow \Diamond(\text{“elevator at } i\text{”} \wedge \Diamond \text{ “elevator at } j\text{”}) \big) \Big) \; ?$$

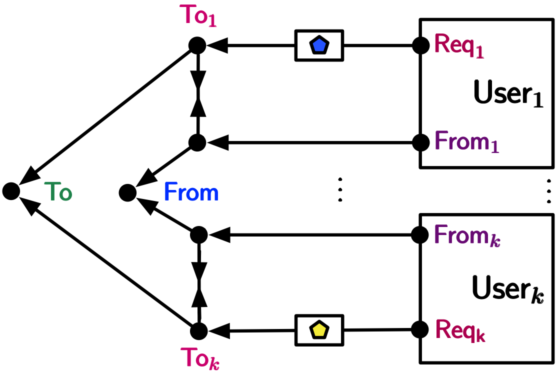


$$\bigwedge_{\substack{1\leqslant i\leqslant k\\ 1\leqslant j\leqslant k}} \Big(\Box \big(\langle\, \mathsf{Req}_i \wedge d_{\mathsf{Req}_i}{=}j \,\rangle \mathit{true} \longrightarrow \Diamond\big(\text{ “elevator at } i\text{” } \wedge \ \Diamond \text{ “elevator at } j\text{” } \big)\big)\Big) \ ?$$

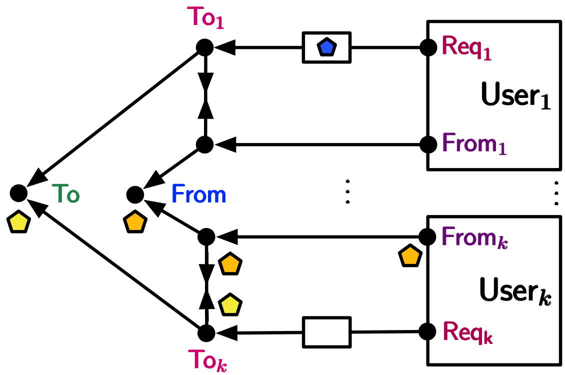


$$\bigwedge_{\substack{1\leqslant i\leqslant k\\1\leqslant j\leqslant k}} \Big(\Box \big(\langle\, \mathsf{Req}_i \wedge d_{\mathsf{Req}_i} = j \,\rangle \mathit{true} \longrightarrow \Diamond(\text{ “elevator at } i\text{” } \wedge \Diamond \text{ “elevator at } j\text{” }) \big) \Big) \text{ ?}$$

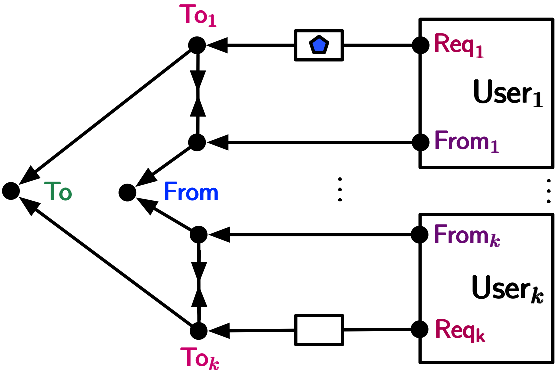
Each user will be served ?



$$\bigwedge_{\substack{1\leqslant i\leqslant k\\ 1\leqslant j\leqslant k}} \Big(\Box \big(\langle \mathrm{Req}_i \wedge d_{\mathrm{Req}_i} = j \rangle \mathit{true} \longrightarrow \Diamond(\text{ “elevator at } i\text{” } \wedge \Diamond \text{ “elevator at } j\text{” }) \big) \Big) \; ?$$



$$\bigwedge_{\substack{1\leqslant i\leqslant k\\ 1\leqslant j\leqslant k}} \Big(\Box \big(\langle \mathsf{Req}_i \wedge d_{\mathsf{Req}_i}{=}j \rangle \mathit{true} \longrightarrow \Diamond\big(\text{“elevator at } i\text{”} \wedge \Diamond \text{“elevator at } j\text{”}\big)\big)\Big) \; ?$$



unfair merge at node To is possible

Liveness properties often cannot be established for nondeterministic system models, although they are expected to hold.

Nondeterministic system models can contain unrealistic behavior, e.g.,

- a certain process is never scheduled
- a request is ignored forever

Liveness properties often cannot be established for nondeterministic system models, although they are expected to hold.

Nondeterministic system models can contain unrealistic behavior, e.g.,

- a certain process is never scheduled
- a request is ignored forever

> specify appropriate fairness assumptions, consider only fair executions for the analysis, and ignore the unfair ones

unconditional fairness:

$$\Box\Diamond\, \mathit{engaged}(P)$$

weak fairness:

$$\Diamond\Box\, \mathit{enabled}(P) \longrightarrow \Box\Diamond\, \mathit{engaged}(P)$$

strong fairness:

$$\Box\Diamond\, \mathit{enabled}(P) \longrightarrow \Box\Diamond\, \mathit{engaged}(P)$$

$\mathit{engaged}(P) \;\widehat{=}\;$ process P is scheduled

$\mathit{enabled}(P) \;\widehat{=}\;$ process P can be scheduled unconditional fairness:

$$\Box\Diamond\, \mathit{engaged}(A)$$

weak fairness:

$$\Diamond\Box\, \mathit{enabled}(A) \longrightarrow \Box\Diamond \mathit{engaged}(A)$$

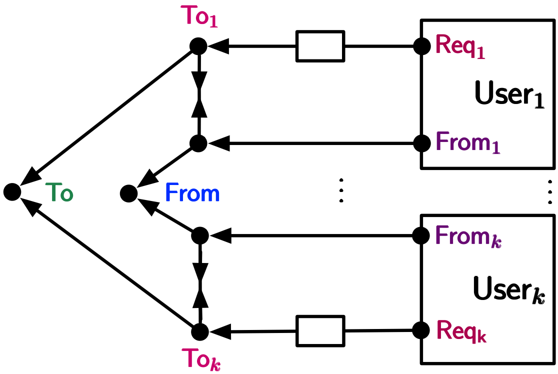
strong fairness:

$$\Box\Diamond\, \mathit{enabled}(A) \longrightarrow \Box\Diamond \mathit{engaged}(A)$$

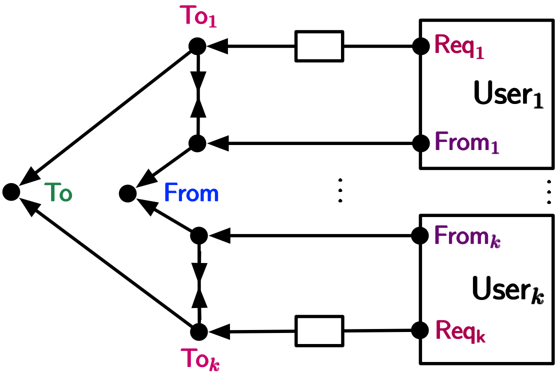
$\mathit{engaged}(A) \;\; \widehat{=} \;\; \langle A \rangle \mathit{true}$

$\mathit{enabled}(A)$ iff there exists a transition of the current state where A is involved

Again: elevator request component



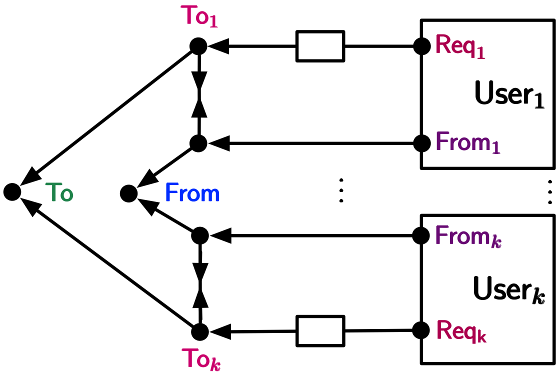
Again: elevator request component



suppose that node To resolves its choices fairly:

$$\psi = \bigwedge_{1 \leqslant i \leqslant k} \Big(\Box\Diamond\ \mathit{enabled}(\mathsf{To}_i) \longrightarrow \Box\Diamond\langle \mathsf{To}_i \rangle \mathit{true} \Big)$$

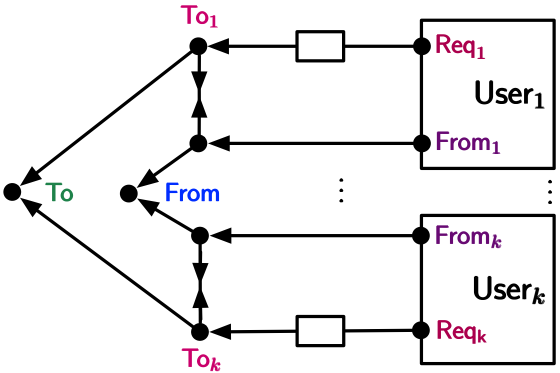
strong fairness for the nodes $\mathsf{To}_1, \ldots, \mathsf{To}_k$



suppose that node To resolves its choices fairly:

$$\psi = \bigwedge_{1 \leqslant i \leqslant k} \Big(\Box\Diamond\, \mathit{enabled}(\mathsf{To}_i) \longrightarrow \Box\Diamond \langle \mathsf{To}_i \rangle \mathit{true} \Big)$$

"there exists a transition where To$_i$ is active"



$$\bigwedge_{\substack{1 \leqslant i \leqslant k \\ 1 \leqslant j \leqslant k}} \Big(\Box \big(\langle\, \mathit{Req}_i \wedge d_{\mathit{Req}_i} = j \,\rangle \mathit{true} \longrightarrow \Diamond(\text{“elevator at } i\text{”} \wedge \Diamond \text{ “elevator at } j\text{”}) \big) \Big)$$

holds under fairness assumption ψ

Model checking with fairness

given: constraint automaton $\mathcal{A}$
LTL$_{\mathrm{IO}}$ formula φ
fairness assumptions $\psi = \bigwedge_i \psi_i$

question: does $\mathcal{A} \models \varphi$ hold under fairness assumption ψ ?

Model checking with fairness

given: constraint automaton $\mathcal{A}$

LTL_{IO} formula φ

fairness assumptions $\psi = \bigwedge_i \psi_i$

question: does $\mathcal{A} \models \varphi$ hold under fairness assumption ψ ?

i.e., for all executions π of $\mathcal{A}$:

if $\pi \models \psi$ then $\pi \models \varphi$

given: constraint automaton $\mathcal{A}$
LTL$_{\text{IO}}$ formula φ
fairness assumptions $\psi = \bigwedge_i \psi_i$

question: does $\mathcal{A} \models \varphi$ hold under fairness assumption ψ ?

i.e., for all executions π of $\mathcal{A}$:

if $\pi \models \psi$ then $\pi \models \varphi$ $\longleftarrow$ $\boxed{\pi \models (\psi \rightarrow \varphi)}$

given: constraint automaton $\mathcal{A}$
LTL$_{IO}$ formula φ
fairness assumptions $\psi = \bigwedge_i \psi_i$

question: does $\mathcal{A} \models \varphi$ hold under fairness assumption ψ ?

i.e., for all executions π of $\mathcal{A}$:

if $\pi \models \psi$ then $\pi \models \varphi$ $\longleftarrow$ $\boxed{\pi \models (\psi \rightarrow \varphi)}$

method: use standard LTL$_{IO}$ model checker
for the formula $\psi \rightarrow \varphi$
(or adapt the model checking procedure)

Overview

1 Modelling components and connectors

constraint automata (CA)

coordination language Reo

2 **Model checking with CA**

Linear Temporal Logic

Alternating Stream Logic $\longleftarrow$

3 Synthesis of connectors

Alternating-time stream logic (ASL)

branching-time logic that combines features of

- computation tree logic (CTL) [Clarke,Emerson '81]
  - temporal operators (safety, liveness)
- modal logic (dynamic logic PDL)
  - regular expressions for specifying data streams and their effect

Alternating-time stream logic (ASL)

branching-time logic that combines features of

- computation tree logic (CTL) [Clarke,Emerson '81]
  - ∗ temporal operators (safety, liveness)
- modal logic (dynamic logic PDL)
  - ∗ regular expressions for specifying data streams and their effect
- alternating-time temporal logic [Alur,Henz.,Kupfer.'97]
  - ∗ CA as multi-player games
  - ∗ reasoning about strategies of components and cooperation facilities of coalitions of components

Syntax of ASL

ASL state formulas: as in CTL + ...

$$\Phi ::= \textit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \ldots$$

atomic proposition

Syntax of ASL

ASL state formulas: as in CTL + ...

$$\Phi ::= \mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \ldots$$

atomic proposition

existential path quantifier

Syntax of ASL

ASL state formulas: as in CTL + ...

$$\Phi ::= \textit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \ldots$$

ASL path formulas: as in CTL + ...

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \mathsf{U} \Phi_2 \mid \ldots$$

ASL state formulas: as in CTL + ATL-like quantifiers

$$\Phi ::= \mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \mathbb{E}_{\mathfrak{C}}\varphi$$

ASL path formulas: as in CTL + ...

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \,\mathsf{U}\, \Phi_2 \mid \ldots$$

$\exists\varphi$ "there exists an execution where φ holds"

$\mathbb{E}_{\mathfrak{C}}\varphi$ "coalition $\mathfrak{C}$ can enforce that φ holds"
no matter how the opponents behave

ASL state formulas: as in CTL + ATL-like quantifiers

$$\Phi ::= \textit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \mathbb{E}_{\mathfrak{C}}\varphi$$

ASL path formulas: as in CTL + ...

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \, \mathsf{U} \, \Phi_2 \mid \ldots$$

$\exists\varphi$ "there exists an execution where φ holds"

$\mathbb{E}_{\mathfrak{C}}\varphi$ "coalition $\mathfrak{C}$ can enforce that φ holds"
no matter how the opponents behave

where $\mathfrak{C}$ is a coalition, i.e., set of I/O-ports

Syntax of ASL

ASL state formulas: as in CTL + ATL-like quantifiers

$$\Phi ::= \mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \mathbb{E}_{\mathfrak{C}}\varphi \mid \mathbb{A}_{\mathfrak{C}}\varphi$$

ASL path formulas: as in CTL + ...

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \,\mathsf{U}\, \Phi_2 \mid \ldots$$

$\exists\varphi$	"there exists an execution where φ holds"
$\mathbb{E}_{\mathfrak{C}}\varphi$	"coalition $\mathfrak{C}$ can enforce that φ holds"
$\mathbb{A}_{\mathfrak{C}}\varphi$	"coalition $\mathfrak{C}$ cannot avoid that φ holds"

where $\mathfrak{C}$ is a coalition, i.e., set of I/O-ports

ASL state formulas: as in CTL + ATL-like quantifiers

$$\Phi ::= \mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \mathbb{E}_{\mathfrak{C}}\varphi \mid \mathbb{A}_{\mathfrak{C}}\varphi$$

ASL path formulas: as in CTL + ...

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \, \mathsf{U} \, \Phi_2 \mid \ldots$$

$\exists\varphi$ "there exists an execution where φ holds"

$\mathbb{E}_{\mathfrak{C}}\varphi$ "coalition $\mathfrak{C}$ can enforce that φ holds"

$\mathbb{A}_{\mathfrak{C}}\varphi$ "coalition $\mathfrak{C}$ cannot avoid that φ holds"

derived: e.g., $\mathbb{A}_{\mathfrak{C}}\Diamond\Phi = \neg\mathbb{E}_{\mathfrak{C}}\Box\neg\Phi$

ASL state formulas: as in CTL + ATL-like quantifiers

$$\Phi ::= \mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \cancel{\exists\varphi} \mid \mathbb{E}_{\mathfrak{C}}\varphi \mid \mathbb{A}_{\mathfrak{C}}\varphi$$

ASL path formulas: as in CTL + ...

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \,\mathsf{U}\, \Phi_2 \mid \ldots$$

$\exists\varphi$ "there exists an execution where φ holds"

$\mathbb{E}_{\mathfrak{C}}\varphi$ "coalition $\mathfrak{C}$ can enforce that φ holds"

$\mathbb{A}_{\mathfrak{C}}\varphi$ "coalition $\mathfrak{C}$ cannot avoid that φ holds"

derived: e.g., $\mathbb{A}_{\mathfrak{C}}\Diamond\Phi = \neg\mathbb{E}_{\mathfrak{C}}\Box\neg\Phi$, $\exists\varphi = \mathbb{A}_{\varnothing}\varphi$

ASL path formulas

ASL state formulas: as in CTL + ATL-like quantifiers

$$\Phi ::= \textit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \mathbb{E}_{\mathfrak{C}}\varphi \mid \mathbb{A}_{\mathfrak{C}}\varphi$$

ASL path formulas: as in CTL + PDL-like expressions

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \,\mathsf{U}\, \Phi_2 \mid \langle\alpha\rangle\Phi \mid [\alpha]\Phi$$

where α is again a regular expression that specifies a set of finite data streams (as in $\mathbf{LTL_{IO}}$)

ASL state formulas: as in CTL + ATL-like quantifiers

$$\Phi ::= \textit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \mathbb{E}_{\mathfrak{C}}\varphi \mid \mathbb{A}_{\mathfrak{C}}\varphi$$

ASL path formulas: as in CTL + PDL-like expressions

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \,\mathsf{U}\, \Phi_2 \mid \langle\alpha\rangle\Phi \mid [\alpha]\Phi$$

where α is again a regular expression that specifies a set of finite data streams (as in $\mathbf{LTL_{IO}}$)

$\langle\alpha\rangle\Phi$	"there is a α-prefix that ends in a Φ-state"
$[\alpha]\Phi$	"all α-prefixes end in a Φ-state"

Semantics of ASL

Semantics of ASL path formulas

ASL path formulas:

$$\varphi ::= \bigcirc \Phi \;\Big|\; \Phi_1 \, \mathsf{U} \, \Phi_2 \;\Big|\; \langle \alpha \rangle \Phi \;\Big|\; [\alpha] \Phi$$

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ of a CA:

ASL path formulas:

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \,\mathsf{U}\, \Phi_2 \mid \langle\alpha\rangle\Phi \mid [\alpha]\Phi$$

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ of a CA:

$\pi \models \bigcirc\Phi$	iff	$q_1 \models \Phi$
$\pi \models \Phi_1 \,\mathsf{U}\, \Phi_2$	iff	there exists $i \geqslant 0$ s.t. $q_i \models \Phi_2$ and $q_j \models \Phi_1$ for all $0 \leqslant j < i$

ASL path formulas:

$$\varphi ::= \bigcirc \Phi \;\big|\; \Phi_1 \, \mathsf{U} \, \Phi_2 \;\big|\; \langle \alpha \rangle \Phi \;\big|\; [\alpha] \Phi$$

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ of a CA:

$\pi \models \bigcirc \Phi$	iff	$q_1 \models \Phi$
$\pi \models \Phi_1 \, \mathsf{U} \, \Phi_2$	iff	there exists $i \geqslant 0$ s.t. $q_i \models \Phi_2$ and $q_j \models \Phi_1$ for all $0 \leqslant j < i$
$\pi \models \langle \alpha \rangle \Phi$	iff	there exists a finite prefix σ of π s.t. $\mathit{stream}(\sigma) \in \mathcal{L}(\alpha)$ and $\mathit{last}(\sigma) \models \Phi$

ASL path formulas:

$$\varphi ::= \bigcirc \Phi \mid \Phi_1 \mathsf{U} \Phi_2 \mid \langle \alpha \rangle \Phi \mid [\alpha]\Phi$$

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ of a CA:

$\pi \models \bigcirc \Phi$	iff	$q_1 \models \Phi$
$\pi \models \Phi_1 \mathsf{U} \Phi_2$	iff	there exists $i \geqslant 0$ s.t. $q_i \models \Phi_2$ and $q_j \models \Phi_1$ for all $0 \leqslant j < i$
$\pi \models \langle \alpha \rangle \Phi$	iff	there exists a finite prefix σ of π s.t. $\mathit{stream}(\sigma) \in \mathcal{L}(\alpha)$ and $\mathit{last}(\sigma) \models \Phi$
$\pi \models [\alpha]\Phi$	iff	for all finite prefixes σ of π: $\mathit{stream}(\sigma) \in \mathcal{L}(\alpha)$ implies $\mathit{last}(\sigma) \models \Phi$

ASL path formulas:

$$\varphi ::= \bigcirc \Phi \mid \Phi_1 \, \mathsf{U} \, \Phi_2 \mid \langle \alpha \rangle \Phi \mid [\alpha] \Phi$$

given an execution $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ of a CA:

$\pi \models \bigcirc \Phi$	iff	$q_1 \models \Phi$
$\pi \models \Phi_1 \, \mathsf{U} \, \Phi_2$	iff	there exists $i \geqslant 0$ s.t. $q_i \models \Phi_2$ and $q_j \models \Phi_1$ for all $0 \leqslant j < i$
$\pi \models \langle \alpha \rangle \Phi$	iff	there exists $i \geqslant 0$ s.t. $c_1 \, c_2 \ldots c_i \in \mathcal{L}(\alpha)$ and $q_i \models \Phi$
$\pi \models [\alpha] \Phi$	iff	for all $i \geqslant 0$ we have: $c_1 \, c_2 \ldots c_i \in \mathcal{L}(\alpha)$ implies $q_i \models \Phi$

Semantics of ASL state formulas

ASL state formulas:

$$\Phi ::= \underbrace{\mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi}_{\text{as in CTL}} \mid \mathbb{E}_{\mathfrak{C}}\,\varphi \mid \mathbb{A}_{\mathfrak{C}}\,\varphi$$

ASL state formulas:

$$\Phi ::= \underbrace{\mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi}_{\text{as in CTL}} \mid \mathbb{E}_{\mathfrak{C}}\,\varphi \mid \mathbb{A}_{\mathfrak{C}}\,\varphi$$

for state q of a CA:

$q \models \mathit{true}$		
$q \models a$	iff	"a holds in state q"
$q \models \Phi_1 \wedge \Phi_2$	iff	$q \models \Phi_1$ and $q \models \Phi_2$
$q \models \neg\Phi$	iff	$q \not\models \Phi$
$q \models \exists\varphi$	iff	there is an execution π starting in q such that $\pi \models \varphi$

ASL state formulas:

$$\Phi ::= \underbrace{\mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi}_{\text{as in CTL}} \mid \mathbb{E}_{\mathfrak{C}}\,\varphi \mid \mathbb{A}_{\mathfrak{C}}\,\varphi$$

for state q of a CA:

$q \models \mathbb{E}_{\mathfrak{C}}\,\varphi$ iff "coalition $\mathfrak{C}$ can enforce that φ holds"

$q \models \mathbb{A}_{\mathfrak{C}}\,\varphi$ iff "coalition $\mathfrak{C}$ cannot avoid that φ holds"

ASL state formulas:

$$\Phi ::= \underbrace{\textit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi}_{\text{as in CTL}} \mid \mathbb{E}_{\mathfrak{C}}\,\varphi \mid \mathbb{A}_{\mathfrak{C}}\,\varphi$$

for state q of a CA:

$q \models \mathbb{E}_{\mathfrak{C}}\,\varphi$	iff	there exists a strategy $\mathcal{S}$ for the ports $A \in \mathfrak{C}$ s.t. for all $\mathcal{S}$-executions π starting in q: $\pi \models \varphi$

$q \models \mathbb{E}_{\mathfrak{C}}\,\varphi$ iff "coalition $\mathfrak{C}$ can enforce that φ holds"

$q \models \mathbb{A}_{\mathfrak{C}}\,\varphi$ iff "coalition $\mathfrak{C}$ cannot avoid that φ holds"

ASL state formulas:

$$\Phi ::= \underbrace{\mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi}_{\text{as in CTL}} \mid \mathbb{E}_{\mathfrak{C}}\,\varphi \mid \mathbb{A}_{\mathfrak{C}}\,\varphi$$

for state q of a CA:

$q \models \mathbb{E}_{\mathfrak{C}}\,\varphi$ iff there exists a strategy $\mathcal{S}$ for the ports $A \in \mathfrak{C}$ s.t. for all $\mathcal{S}$-executions π starting in q: $\pi \models \varphi$

strategy $\mathcal{S}$ assigns to each finite execution σ a set of concurrent I/O-operations that contains all concurrent I/O-operations where no port in $\mathfrak{C}$ is involved

ASL state formulas:

$$\Phi ::= \underbrace{\mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi}_{\text{as in CTL}} \mid \mathbb{E}_{\mathfrak{C}}\,\varphi \mid \mathbb{A}_{\mathfrak{C}}\,\varphi$$

for state q of a CA:

$q \models \mathbb{E}_{\mathfrak{C}}\,\varphi$	iff	there exists a strategy $\mathcal{S}$ for the ports $A \in \mathfrak{C}$ s.t. for all $\mathcal{S}$-executions π starting in q: $\pi \models \varphi$
$q \models \mathbb{A}_{\mathfrak{C}}\,\varphi$	iff	for all strategies $\mathcal{S}$ for the ports $A \in \mathfrak{C}$ there exists an $\mathcal{S}$-execution π starting in q such that $\pi \models \varphi$

ASL model checking

given: finite constraint automaton $\mathcal{A}$
ASL state formula Φ

question: does $\mathcal{A} \models \Phi$ hold **?**

ASL model checking

given: finite constraint automaton $\mathcal{A}$
ASL state formula Φ

question: does $\mathcal{A} \models \Phi$ hold ?

algorithm: compute recursively the satisfaction sets

$$Sat(\Psi) \; = \; \{ \, q \text{ state in } \mathcal{A} : q \models \Psi \, \}$$

for all state subformulas Ψ of Φ

given: finite constraint automaton $\mathcal{A}$
ASL state formula Φ

question: does $\mathcal{A} \models \Phi$ hold ?

algorithm: compute recursively the satisfaction sets

$$Sat(\Psi) = \{ q \text{ state in } \mathcal{A} : q \models \Psi \}$$

for all state subformulas Ψ of Φ

- treatment of $\mathbb{E}_{\mathfrak{c}} \lozenge \Psi$, or $\mathbb{E}_{\mathfrak{c}} \square \Psi$:
iterative fixed point computation
- treatment of $\mathbb{E}_{\mathfrak{c}} \langle \alpha \rangle \Psi$ or $\mathbb{E}_{\mathfrak{c}} [\alpha] \Psi$:
via reduction to $\mathbb{E}_{\mathfrak{c}} \lozenge$ and or $\mathbb{E}_{\mathfrak{c}} \square$, respectively

Computation of $Sat(\mathbb{E}_{\mathfrak{c}}\langle\alpha\rangle\Psi)$

finite constraint automaton $\mathcal{A}$	ASL state formula $\mathbb{E}_{\mathfrak{c}}\langle\alpha\rangle\Psi$

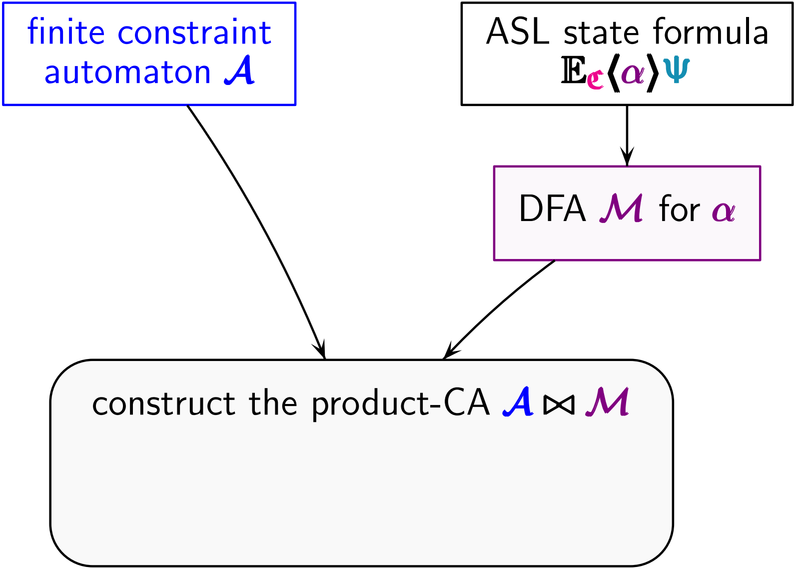
Computation of $Sat(\mathbb{E}_{\mathfrak{c}}\langle\alpha\rangle\Psi)$

finite constraint automaton $\mathcal{A}$

ASL state formula $\mathbb{E}_{\mathfrak{c}}\langle\alpha\rangle\Psi$

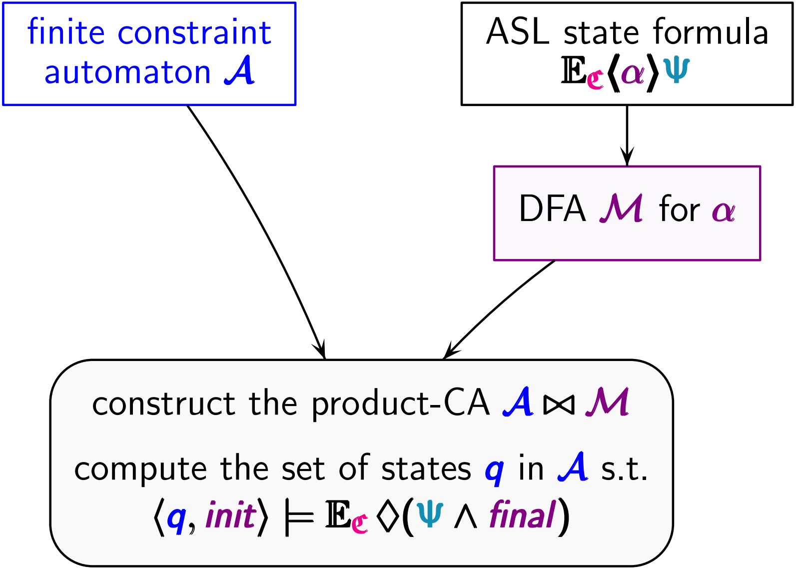
$\downarrow$

DFA $\mathcal{M}$ for α

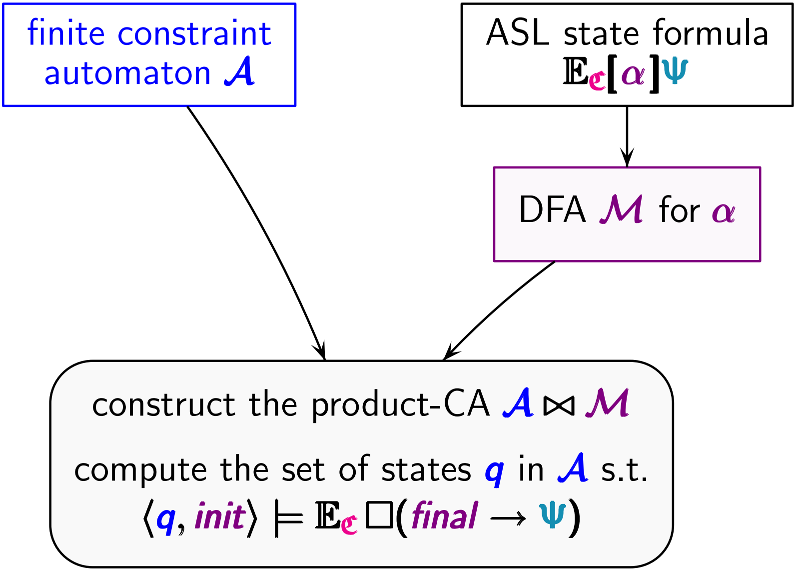


finite constraint automaton $\mathcal{A}$
ASL state formula $\mathbb{E}_{\mathfrak{c}}\langle\alpha\rangle\Psi$
DFA $\mathcal{M}$ for α
construct the product-CA $\mathcal{A} \bowtie \mathcal{M}$

Computation of $Sat(\mathbb{E}_{\mathfrak{c}}\langle\alpha\rangle\Psi)$



Computation of $Sat(\mathbb{E}_{\mathfrak{e}}[\alpha]\Psi)$



given: finite constraint automaton $\mathcal{A}$
ASL state formula Φ

question: does $\mathcal{A} \models \Phi$ hold ?

complexity of the sketched algorithm:

$$\mathcal{O}\big(\mathit{size}(\mathcal{A}) \cdot \mathbf{exp}(k) \cdot |\Phi|\big)$$

where k = maximal length $|\alpha|$ of a stream expression that appears in Φ

given: finite constraint automaton $\mathcal{A}$

ASL state formula Φ

question: does $\mathcal{A} \models \Phi$ hold ?

complexity of the sketched algorithm:

$$\mathcal{O}\big(size(\mathcal{A}) \cdot \mathbf{exp}(k) \cdot |\Phi|\big)$$

where k = maximal length $|\alpha|$ of a stream expression that appears in Φ

exponential blow-up possible as DFA for path subformulas $\langle\alpha\rangle\Psi$ or $[\alpha]\Psi$ are required

Better complexity for fragments of ASL

775

Better complexity for fragments of ASL

complexity of model checking for formulas of

- the ATL-fragment of ASL (without $\langle\!\langle \alpha \rangle\!\rangle$, $[\![\alpha]\!]$):

  $$\mathcal{O}\big(\mathit{size}(\mathcal{A}) \cdot |\Phi|\big) \quad \longleftarrow \text{ as for standard ATL}$$

complexity of model checking for formulas of

- the ATL-fragment of ASL (without $\langle\alpha\rangle$, $[\alpha]$):

  $\mathcal{O}\big(\mathit{size}(\mathcal{A}) \cdot |\Phi|\big) \quad \longleftarrow$ as for standard ATL

- the fragment of ASL consisting of CTL $+\exists\langle\alpha\rangle$:

  $\mathcal{O}\big(\mathit{size}(\mathcal{A}) \cdot |\Phi|\big) \quad \longleftarrow$ NFA are sufficient complexity of model checking for formulas of

- the ATL-fragment of ASL (without $\langle\alpha\rangle$, $[\alpha]$):

  $\mathcal{O}(\mathit{size}(\mathcal{A}) \cdot |\Phi|) \quad \longleftarrow$ as for standard ATL

- the fragment of ASL consisting of CTL $+ \exists\langle\alpha\rangle$:

  $\mathcal{O}(\mathit{size}(\mathcal{A}) \cdot |\Phi|) \quad \longleftarrow$ NFA are sufficient

but: PSPACE-completeness of the model checking problem for ASL formulas of the form $\mathbb{E}_{\mathfrak{C}}\langle\alpha\rangle a$ where α is given by an NFA

ASL and synthesis

779

Cooperation of components

component interfaces
$C_1, C_2, C_3, \ldots, C_n$

connector
Reo network

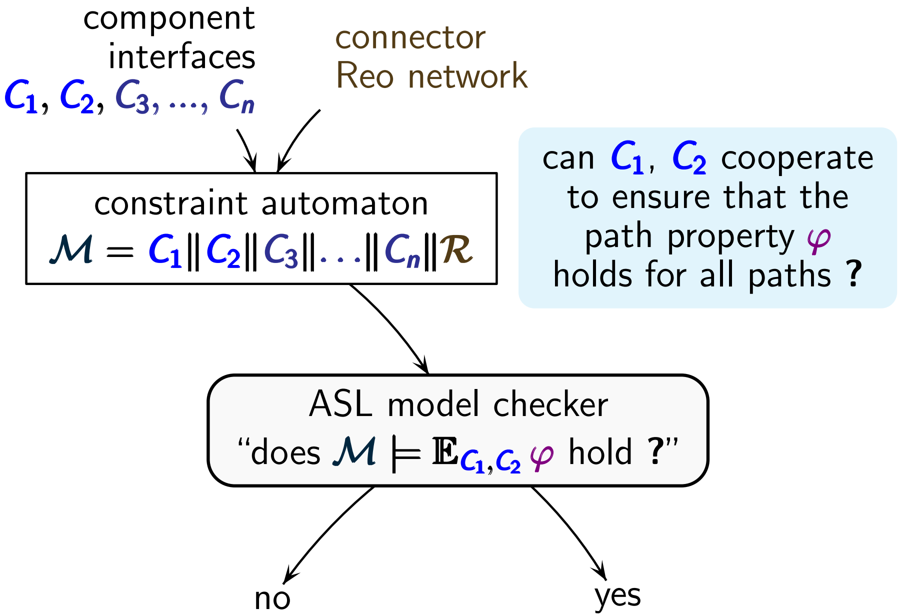
constraint automaton
$\mathcal{M} = C_1 \| C_2 \| C_3 \| \ldots \| C_n \| \mathcal{R}$

ASL model checker
"does $\mathcal{M} \models \mathbb{E}_{C_1, C_2}\, \varphi$ hold **?**"

no

yes

Cooperation of components



Synthesis of controller for C_1 and C_2 ?

780

component interfaces
$C_1, C_2, C_3, \ldots, C_n$

connector
Reo network

constraint automaton
$\mathcal{M} = C_1 \| C_2 \| C_3 \| \ldots \| C_n \| \mathcal{R}$

can C_1, C_2 cooperate to ensure that the path property φ holds for all paths **?**

ASL model checker
"does $\mathcal{M} \models \mathbb{E}_{C_1,C_2}\, \varphi$ hold **?**"

no

?

network for C_1, C_2

Compatibility checking

component interfaces
$C_1, C_2, C_3, \ldots, C_n$

can $C_1, \ldots, C_n$ cooperate such that $C_1 \| \ldots \| C_n$ is never blocked ?

constraint automaton
$\mathcal{M} = C_1 \| C_2 \| C_3 \| \ldots \| C_n$

compatibility
[deAlfaro/Henzinger'03]

ASL model checker
"$\mathcal{M} \models \mathbb{E}_{C_1,\ldots,C_n} \bigcirc \textit{true}$?"

no

yes

Compatibility checking

830

component interfaces
$C_1, C_2, C_3, \ldots, C_n$

do the interfaces of $C_1, \ldots, C_n$ meet each other interface constraints **?**

constraint automaton
$\mathcal{M} = C_1 \| C_2 \| C_3 \| \ldots \| C_n$

ASL$^+$ path formula
$\varphi = \psi_1 \wedge \ldots \wedge \psi_n$

ASL model checker
"$\mathcal{M} \models \mathbb{E}_{C_1, \ldots C_n} \varphi$ **?**"

no

yes

component interfaces $C_1, C_2, C_3, \ldots, C_n$

do the interfaces of $C_1, \ldots, C_n$ meet each other interface constraints **?**

constraint automaton

$\mathcal{M} = C_1 \| C_2 \| C_3 \| \ldots \| C_n$

ASL$^+$ path formula

$\varphi = \psi_1 \wedge \ldots \wedge \psi_n$

ASL model checker

"$\mathcal{M} \models \mathbb{E}_{C_1,\ldots C_n}\, \varphi$ **?**"

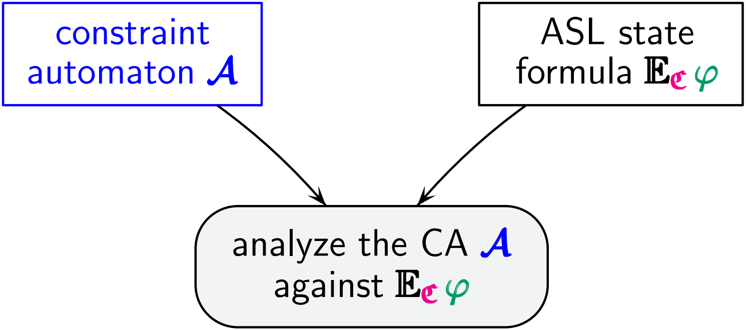
?

no

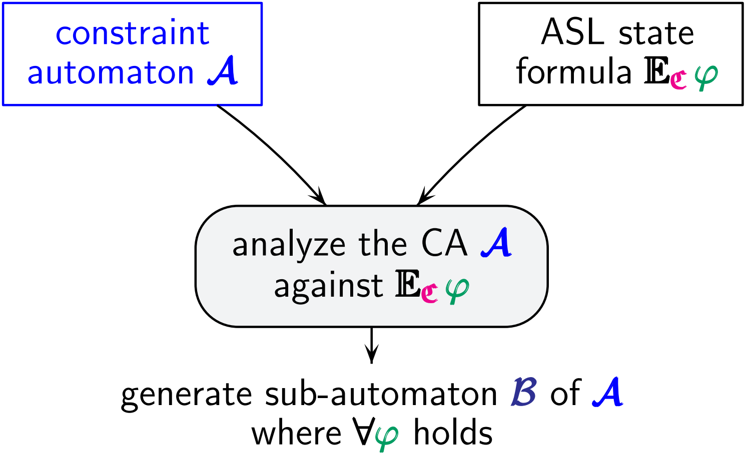
yes + **Reo network**

Generation of Reo network

820



Generation of Reo network



Generation of Reo network

constraint automaton $\mathcal{A}$

ASL state formula $\mathbb{E}_{\mathfrak{C}}\,\varphi$

analyze the CA $\mathcal{A}$ against $\mathbb{E}_{\mathfrak{C}}\,\varphi$

generate sub-automaton $\mathcal{B}$ of $\mathcal{A}$ where $\forall\varphi$ holds

derive a Reo network $\mathcal{R}$ that coordinates $\mathfrak{C}$ from the automaton $\mathcal{B}$

Overview

1 Modelling components and connectors

constraint automata (CA)

coordination language Reo

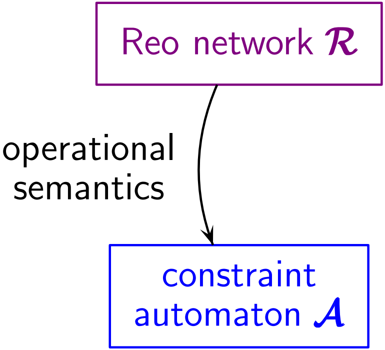
2 Model checking with CA

Linear Temporal Logic

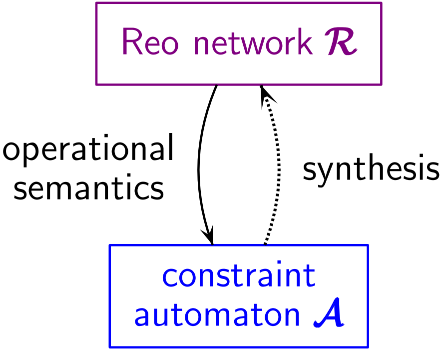
Alternating Stream Logic

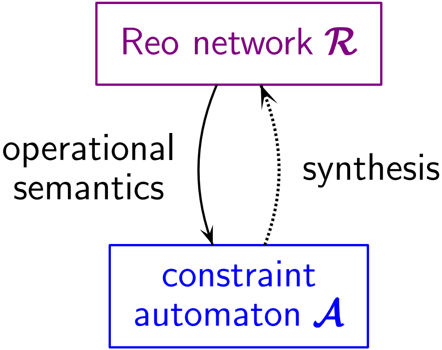
3 **Synthesis of connectors**

Reo and CA


Reo network $\mathcal{R}$
operational
semantics
constraint
automaton $\mathcal{A}$

Reo and CA


Reo network $\mathcal{R}$
operational semantics
synthesis
constraint automaton $\mathcal{A}$



For each finite CA $\mathcal{A}$, a Reo network $\mathcal{R}$ of size $\mathcal{O}(\mathit{size}(\mathcal{A}))$ can be constructed such that

$$\mathcal{A} = \text{operational semantics of } \mathcal{R}$$

(up to some notion of behavioral equivalence)

given: constraint automaton $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, \{q_0\})$

where $\mathcal{N} = \mathcal{N}_{\mathbf{in}} \cup \mathcal{N}_{\mathbf{out}}$ (no internal ports)

task: synthesize an equivalent Reo network $\mathcal{R}$

given: constraint automaton $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, \{q_0\})$

where $\mathcal{N} = \mathcal{N}_{\mathbf{in}} \cup \mathcal{N}_{\mathbf{out}}$ (no internal ports)

task: synthesize an equivalent Reo network $\mathcal{R}$

idea:

states	$\rightsquigarrow$	FIFO buffers
transitions	$\rightsquigarrow$	data flow in $\mathcal{R}$

given: constraint automaton $\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, \{q_0\})$

where $\mathcal{N} = \mathcal{N}_{\mathbf{in}} \cup \mathcal{N}_{\mathbf{out}}$ (no internal ports)

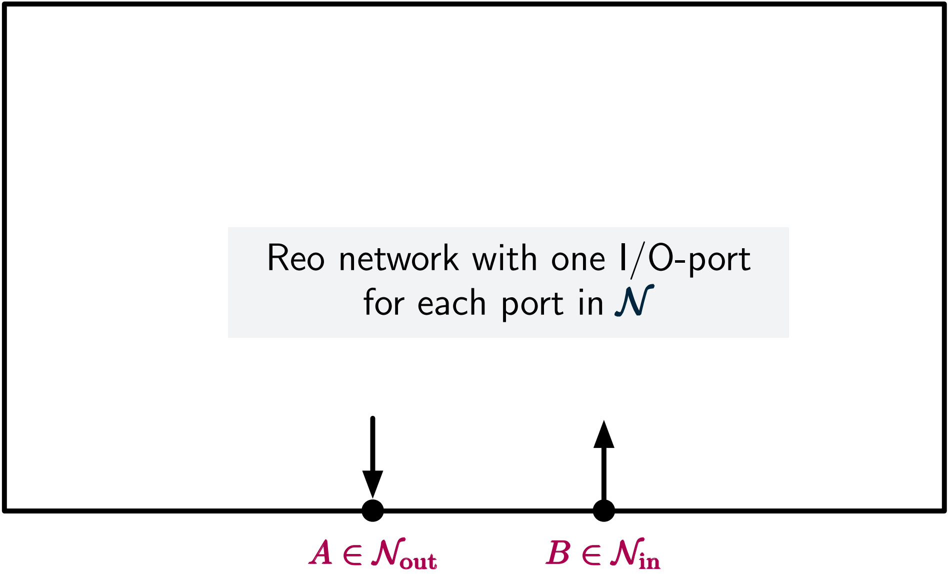
task: synthesize an equivalent Reo network $\mathcal{R}$

idea: states $\rightsquigarrow$ FIFO buffers

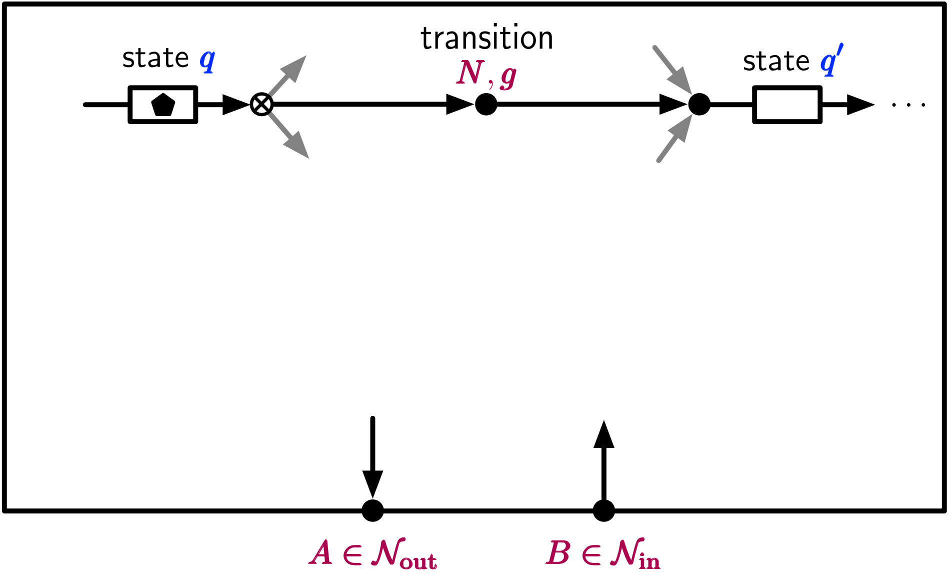
transitions $\rightsquigarrow$ data flow in $\mathcal{R}$

- current state q of $\mathcal{A}$ is represented by a token in the FIFO buffer for q
- firing a transition in $\mathcal{A}$ corresponds to passing the token from one FIFO to another one

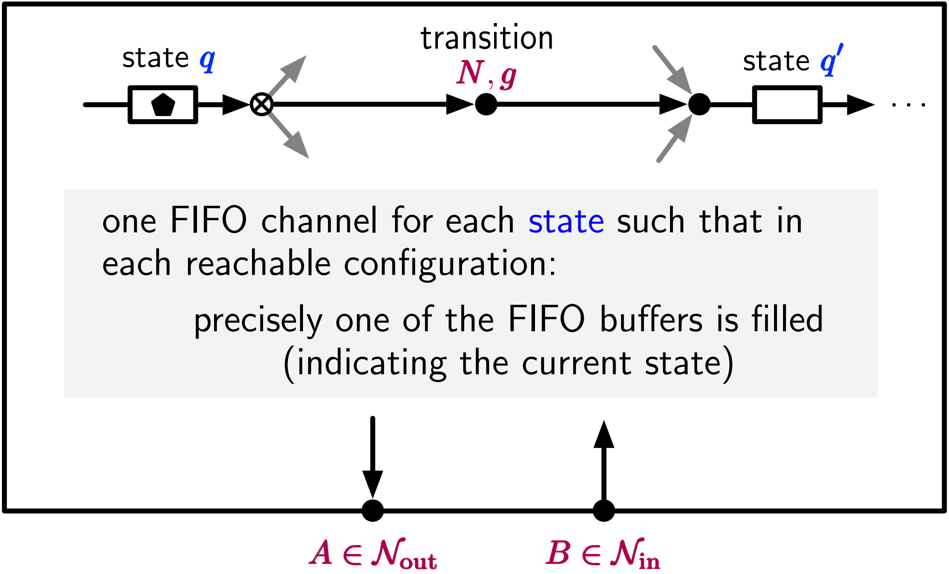
representation of transition $q \xrightarrow{N, g} q'$ in the network representation of transition $q \xrightarrow{N, g} q'$ in the network



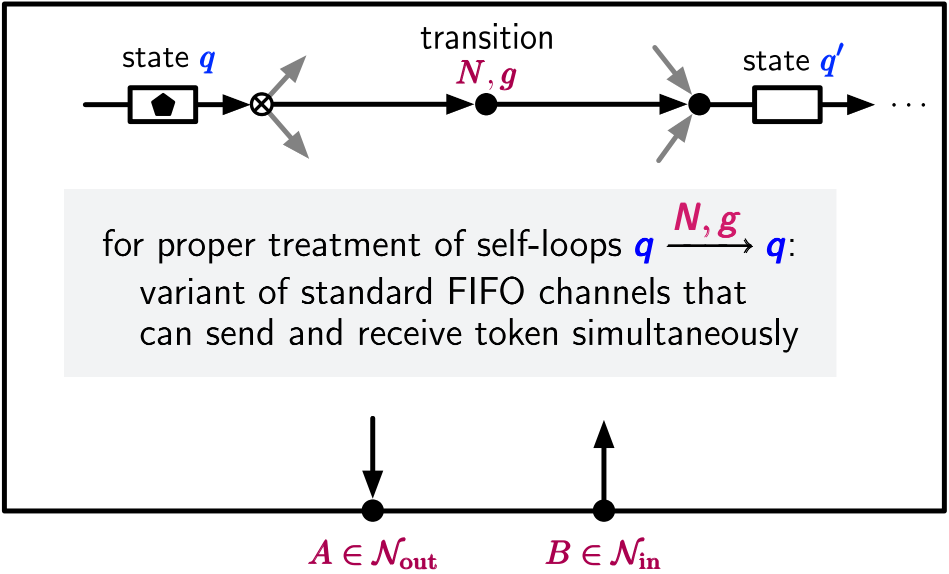
representation of transition $q \xrightarrow{N,g} q'$ in the network



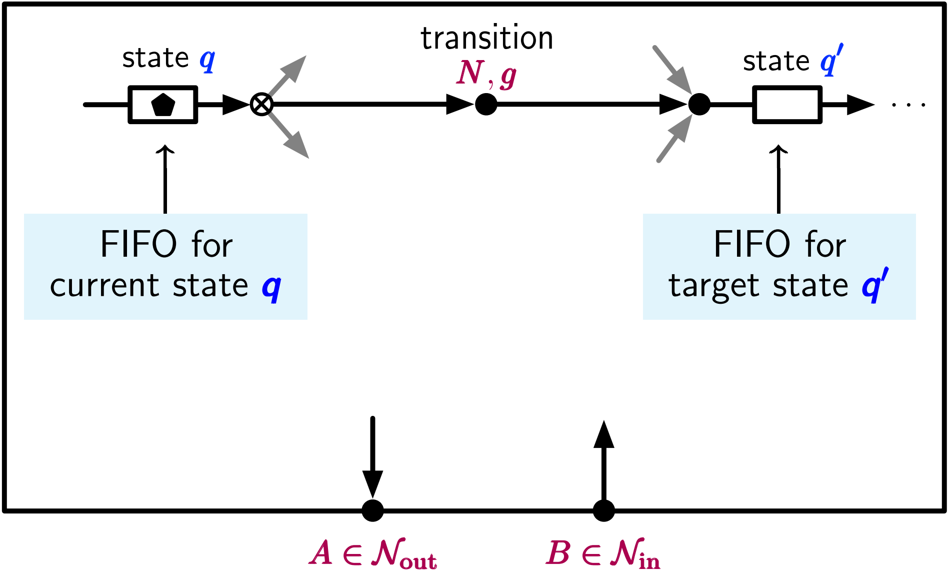
representation of transition $q \xrightarrow{N,g} q'$ in the network



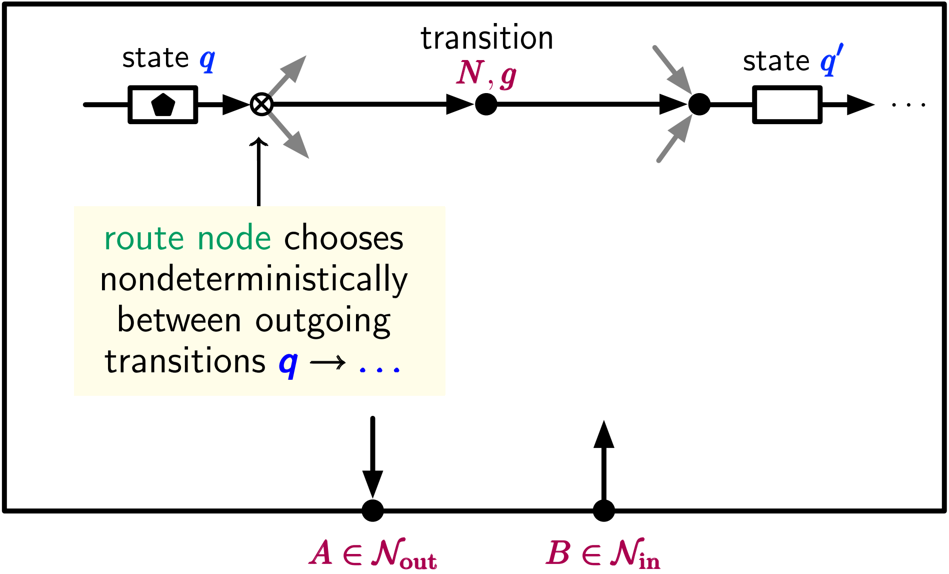
representation of transition $q \xrightarrow{N,g} q'$ in the network



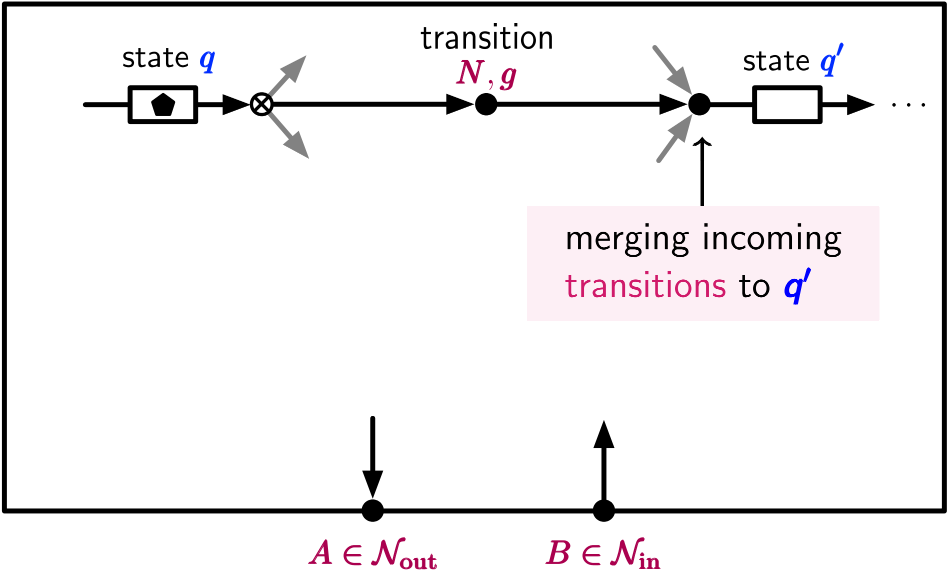
representation of transition $q \xrightarrow{N,g} q'$ in the network



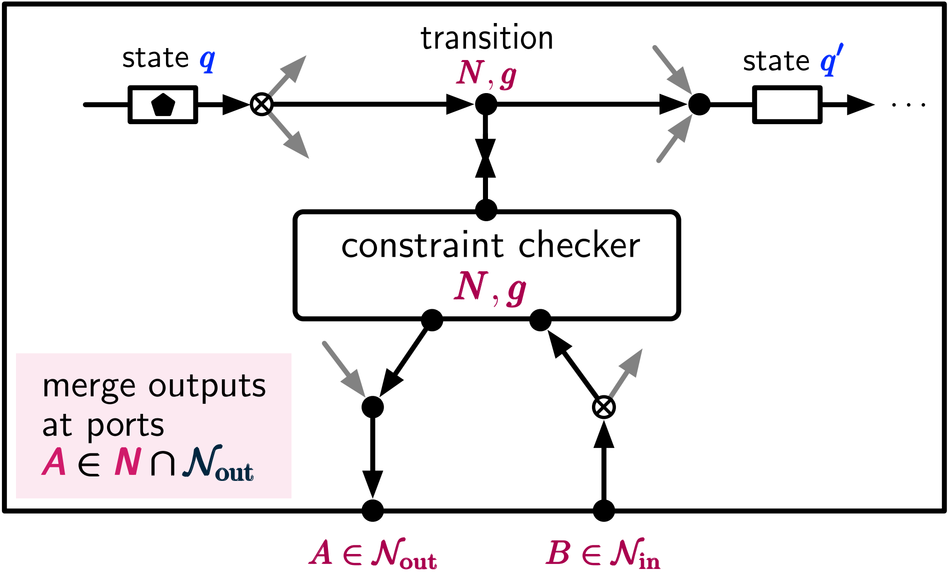
representation of transition $q \xrightarrow{N,\,g} q'$ in the network



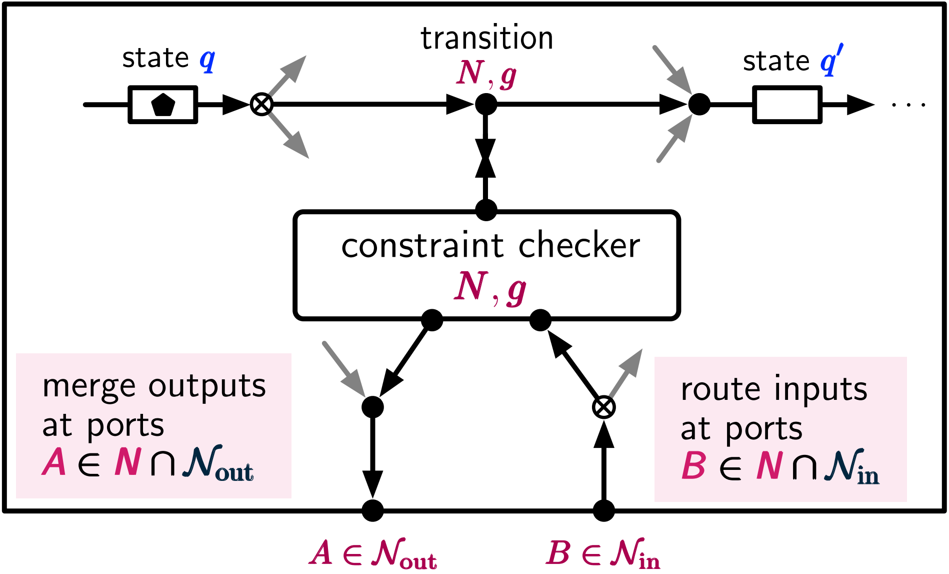
representation of transition $q \xrightarrow{N,g} q'$ in the network



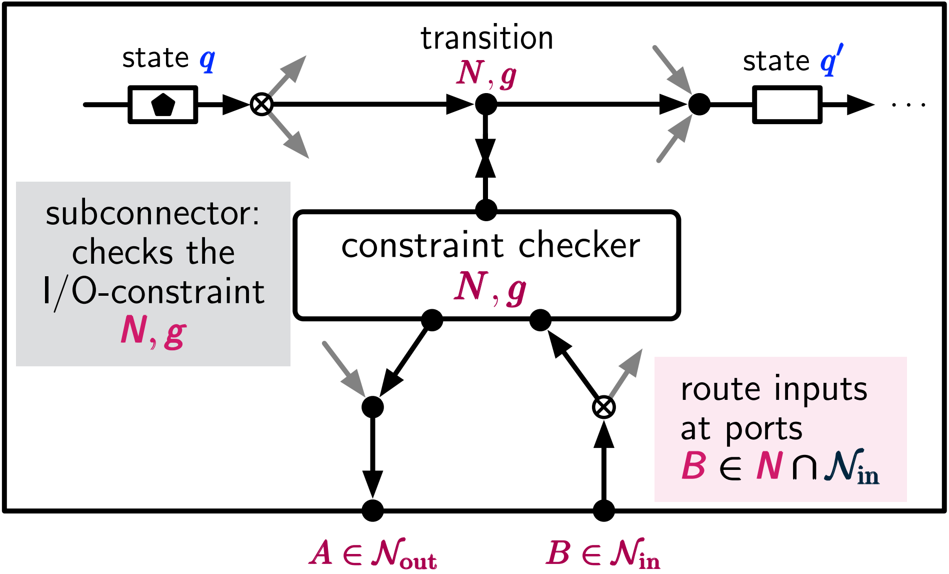
representation of transition $q \xrightarrow{N,g} q'$ in the network



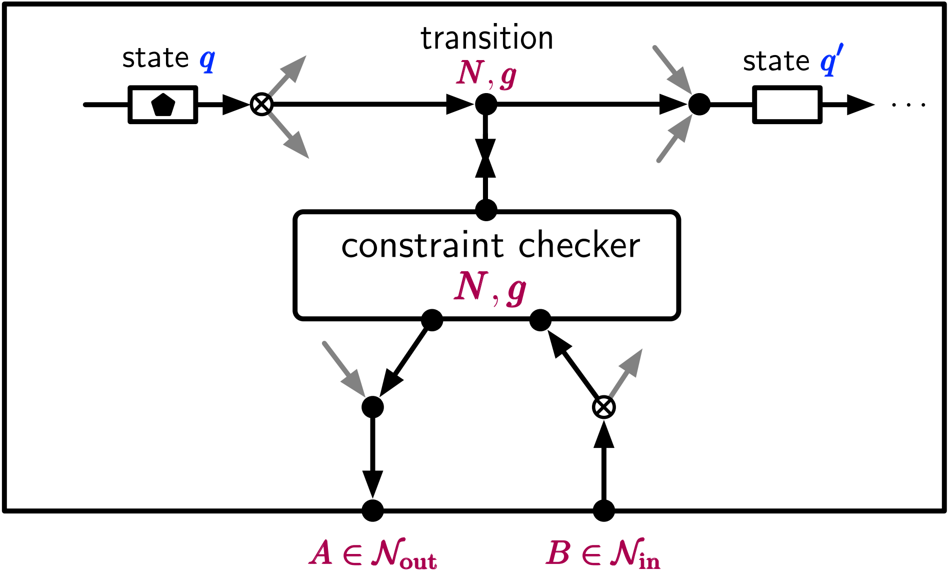
representation of transition $q \xrightarrow{N,g} q'$ in the network



representation of transition $q \xrightarrow{N,g} q'$ in the network

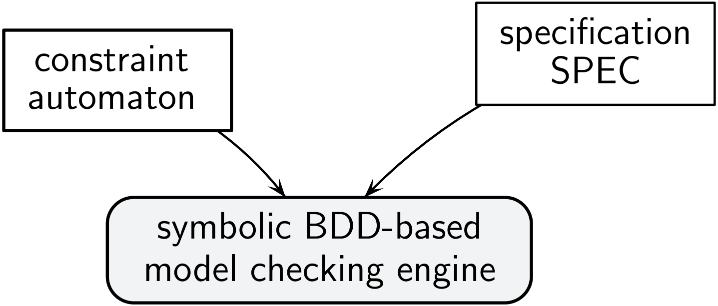


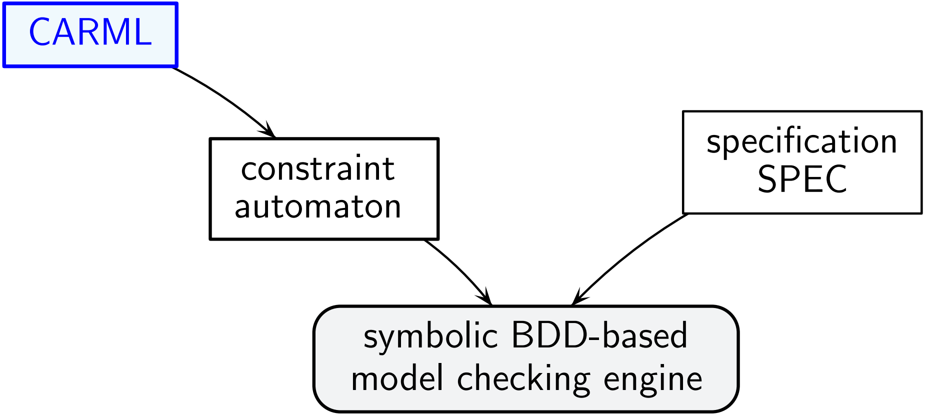
representation of transition $q \xrightarrow{N,g} q'$ in the network



Vereofy

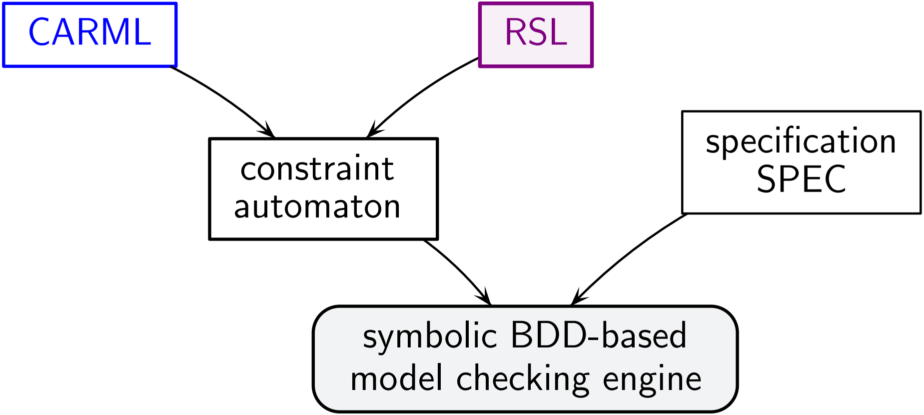
modeling and verification toolset for CA & Reo


constraint automaton
specification SPEC
symbolic BDD-based model checking engine



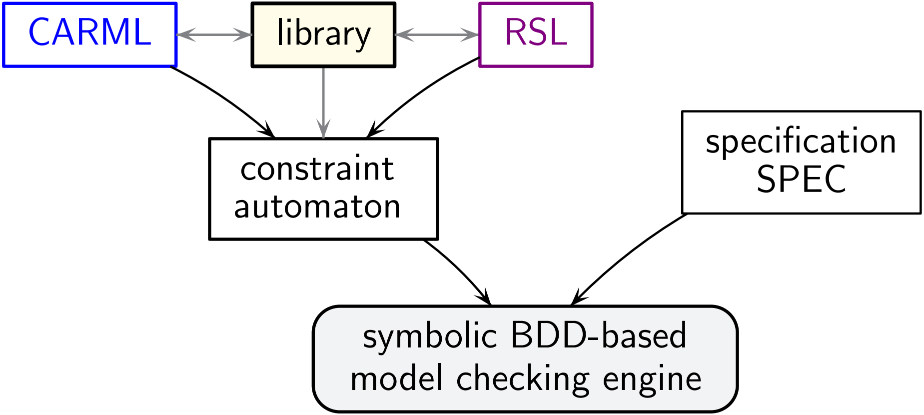
relies on a hybrid modeling approach

CARML guarded command language for specifying CA (component interfaces)



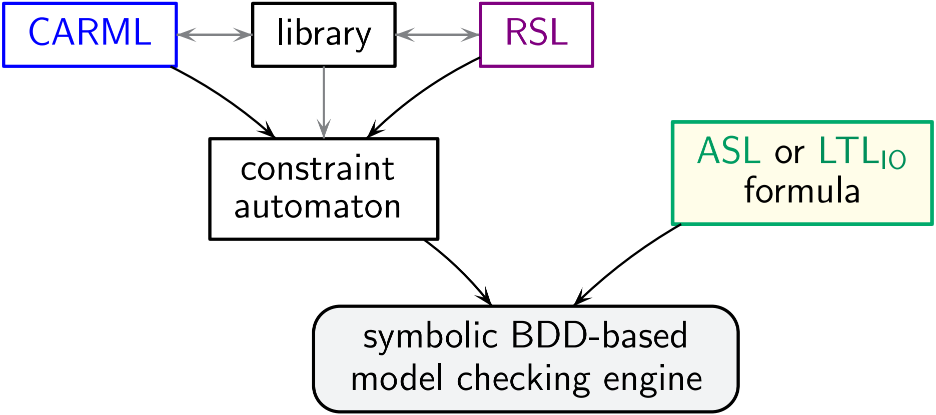
relies on a hybrid modeling approach

CARML	guarded command language for specifying CA
RSL	Reo scripting language



relies on a hybrid modeling approach

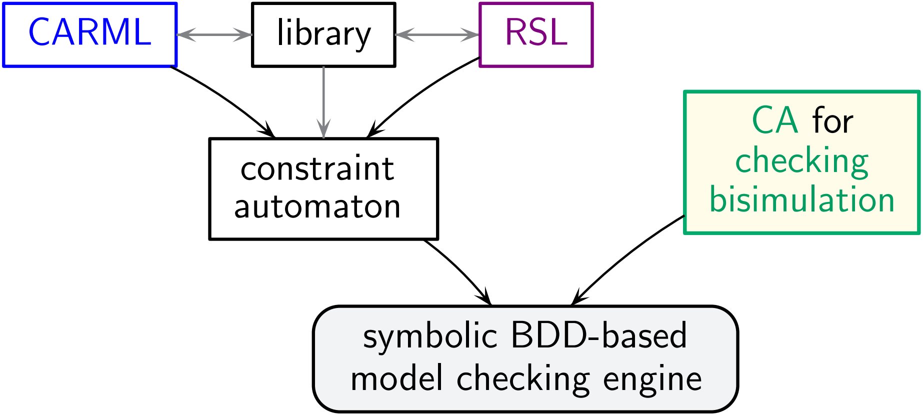
CARML	guarded command language for specifying CA
RSL	Reo scripting language



model checking engine:

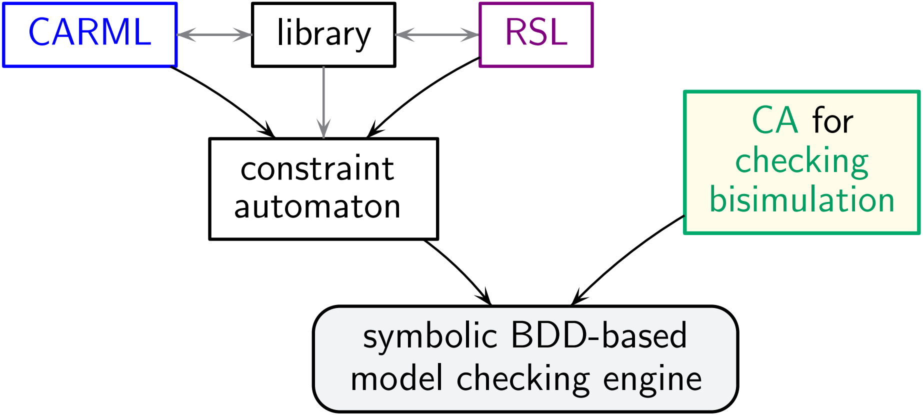
- temporal logics ASL and (fragment of) LTL_{IO}

Vereofy



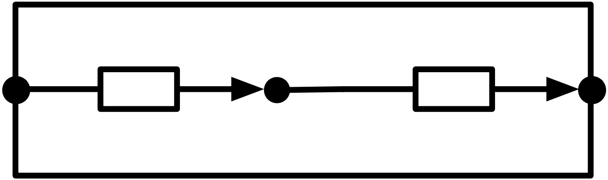
model checking engine:

- temporal logics ASL and (fragment of) $\mathrm{LTL}_{\mathrm{IO}}$
- bisimulation checking



integrated in the Extensible Coordination Tools GUI
graphical Reo network editor, RSL import/export,
counterexample/witness animations,...

Example: RSL script for FIFO with k buffer cells



Example: RSL script for FIFO with *k* buffer cells

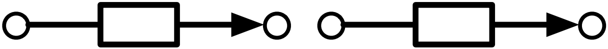
```
CIRCUIT FIFO<k> {
 for (i=0;i<k;i=i+1) {
  f[i] = new FIFO1;
  if (i>1) {
    join(f[i-1].sink, f[i].source); }
 }
 source = f[0].source;
 sink = f[k-1].sink;
}
```

Example: RSL script for FIFO with k buffer cells

```
CIRCUIT FIFO<k> {                ⟵ parametrization
 for (i=0;i<k;i=i+1) {
  f[i] = new FIFO1;
  if (i>1) {
    join(f[i-1].sink, f[i].source); }
 }
 source = f[0].source;
 sink = f[k-1].sink;
}
```

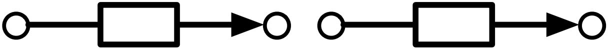
Example: RSL script for FIFO with k buffer cells

```
CIRCUIT FIFO<k> {                        ⟵ parametrization
  for (i=0;i<k;i=i+1) {
   f[i] = new FIFO1;                     ⟵ instantiation
   if (i>1) {
     join(f[i-1].sink, f[i].source); }
  }
  source = f[0].source;
  sink = f[k-1].sink;
}
```



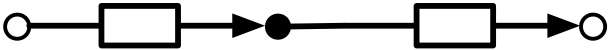
```
CIRCUIT FIFO<k> {                      ⟵ parametrization
 for (i=0;i<k;i=i+1) {
  f[i] = new FIFO1;                    ⟵ instantiation
  if (i>1) {
    join(f[i-1].sink, f[i].source); }
 }
 source = f[0].source;
 sink = f[k-1].sink;
}
```

combine
sink end of FIFO $\boldsymbol{i-1}$
with source end of FIFO $\boldsymbol{i}$
in a node



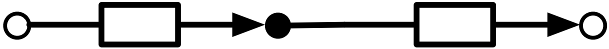
```
CIRCUIT FIFO<k> {                    ⟵ parametrization
  for (i=0;i<k;i=i+1) {
   f[i] = new FIFO1;                 ⟵ instantiation
   if (i>1) {
     join(f[i-1].sink, f[i].source); }
  }
  source = f[0].source;
  sink = f[k-1].sink;
}
```

combine
sink end of FIFO $\boldsymbol{i-1}$
with source end of FIFO $\boldsymbol{i}$
in a node



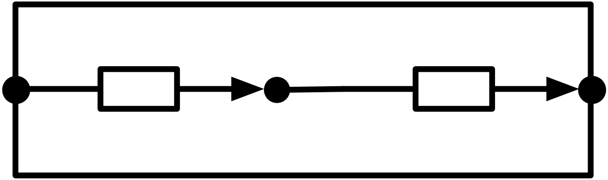
Example: RSL script for FIFO with *k* buffer cells

```
CIRCUIT FIFO<k> {                        ⟵ parametrization
 for (i=0;i<k;i=i+1) {
  f[i] = new FIFO1;                      ⟵ instantiation
  if (i>1) {
    join(f[i-1].sink, f[i].source); }
 }
 source = f[0].source;
 sink = f[k-1].sink;                     ⟵ interface declaration
}
```



Example: RSL script for FIFO with k buffer cells


```
CIRCUIT FIFO<k> {                          ←— parametrization
  for (i=0;i<k;i=i+1) {
   f[i] = new FIFO1;                       ←— instantiation
   if (i>1) {
     join(f[i-1].sink, f[i].source); }
  }
  source = f[0].source;                    ←— interface declaration
  sink = f[k-1].sink;
}
```



Conclusions

1000

constraint automata ... uniform operational model for

- component interfaces $C_1, \ldots, C_n$
- glue code $\mathcal{R}$
- composite system $C_1 \| \ldots \| C_n \| \mathcal{R}$

Conclusions

constraint automata ... uniform operational model for

- component interfaces $C_1, \ldots, C_n$
- glue code $\mathcal{R}$ $\longleftarrow$ Reo network
- composite system $C_1 \| \ldots \| C_n \| \mathcal{R}$

Reo ... elegant, declaractive way to specify connectors

- compositional CA-semantics
- well suited for hierarchical design
- very expressive

Conclusions

constraint automata ... uniform operational model for

- component interfaces $C_1, \ldots, C_n$
- glue code $\mathcal{R}$ $\longleftarrow$ Reo network
- composite system $C_1 \| \ldots \| C_n \| \mathcal{R}$

Reo ... elegant, declaractive way to specify connectors

- compositional CA-semantics
- well suited for hierarchical design
- very expressive

embeddings of various other formalisms into Reo

Petri nets	CCS-like process calculi
UML sequence diagrams	name-passing calculi

constraint automata ... uniform operational model for

- component interfaces $C_1, \ldots, C_n$
- glue code $\mathcal{R}$ $\longleftarrow$ Reo network
- composite system $C_1 \| \ldots \| C_n \| \mathcal{R}$

Reo ... elegant, declaractive way to specify connectors

- compositional CA-semantics
- well suited for hierarchical design
- very expressive

various applications, e.g.,

coordination protocols
web-services
electronic auction protocol
biological systems
sensor networks
⋮

constraint automata ... uniform operational model for

- component interfaces $C_1, \ldots, C_n$
- glue code $\mathcal{R}$ $\longleftarrow$ Reo network
- composite system $C_1 \| \ldots \| C_n \| \mathcal{R}$

Reo ... elegant, declaractive way to specify connectors

- compositional CA-semantics
- well suited for hierarchical design
- very expressive

just mild adaptions of known verification techniques

- temporal logics: linear- and branching-time
- bisimulation equivalence checking
- synthesis, compatibility, alternating-time logics

Current developments, future work

- quantitative analysis, QoS
  - probabilistic extensions of CA
  - timed CA
  - CA with other cost functions

Current developments, future work

- quantitative analysis, QoS
  - probabilistic extensions of CA
  - timed CA
  - CA with other cost functions
- variants of CA to model context-sensitive behaviors and dynamic reconfigurations

Current developments, future work

- quantitative analysis, QoS
  - probabilistic extensions of CA
  - timed CA
  - CA with other cost functions
- variants of CA to model context-sensitive behaviors and dynamic reconfigurations
- better algorithms for compatibility and compositional connector synthesis
- extensions of ASL and LTL_{IO} (towards ASL*)
- partial-information game semantics for CA, proper game logic