Automata and Logic for Concurrent Systems

Benedikt Bollig

Laboratoire Spécification et Vérification

Workshop *Automaten und Logik*
Theorietag *Automaten und Formale Sprachen*
25.-27. September 2013, Ilmenau

What is a concurrent system?

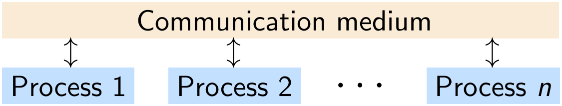
- Collection of autonomous computing entities (processes) connected by some communication medium

What is a concurrent system?

- Collection of autonomous computing entities (processes) connected by some communication medium
- Processes access and update shared resources (e.g., variables, channels, databases, ...)

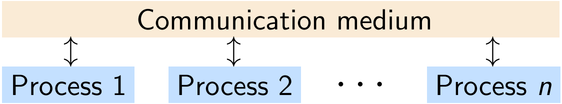
What is a concurrent system?

- Collection of autonomous computing entities (processes) connected by some communication medium
- Processes access and update shared resources (e.g., variables, channels, databases, ...)
- Schematic view:



What is a concurrent system?

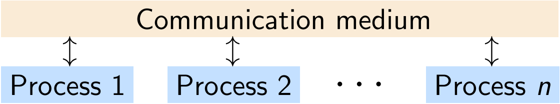
- Collection of autonomous computing entities (processes) connected by some communication medium
- Processes access and update shared resources (e.g., variables, channels, databases, ...)
- Schematic view:



- Purpose:
  - entities collaborate on a task:
    terminating computation with input and output
  - entities model a reactive system:
    focus on behavior, properties of performed action sequence
    (e.g., mutual exclusion)

What is a concurrent system?

- Collection of autonomous computing entities (processes) connected by some communication medium
- Processes access and update shared resources (e.g., variables, channels, databases, ...)
- Schematic view:



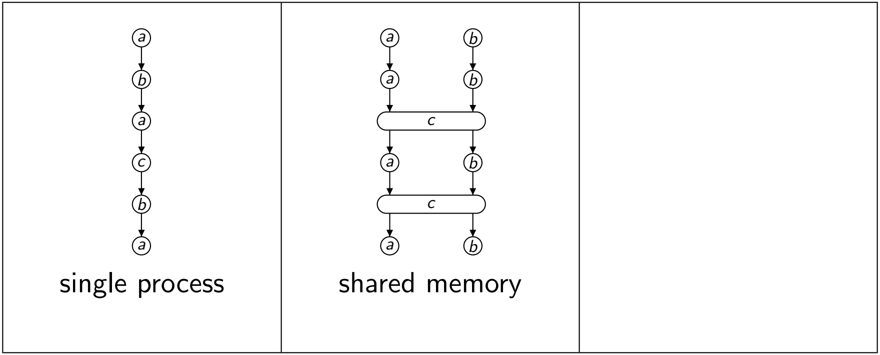
- Purpose:
  - entities collaborate on a task:
    terminating computation with input and output
  - entities model a reactive system:
    focus on behavior, properties of performed action sequence
    (e.g., mutual exclusion)
- In this talk: formal modeling of concurrent reactive systems (in terms of automata) to make them accessible to formal methods

2. Classification

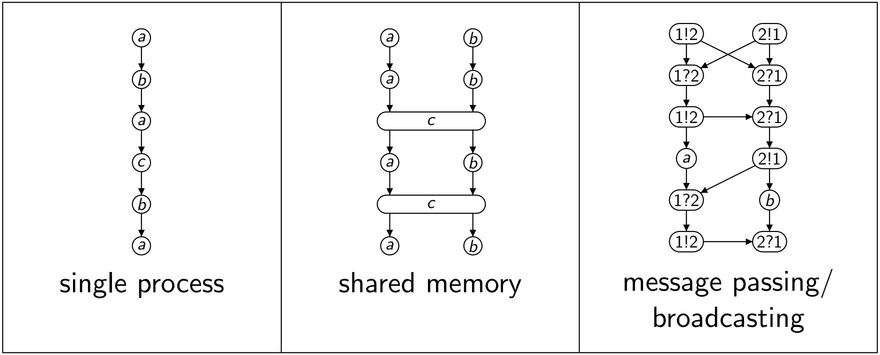
Form of communication

single process

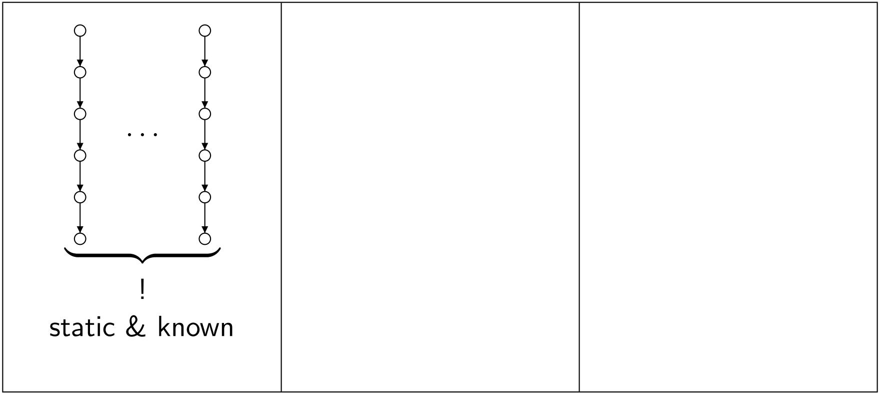
Form of communication



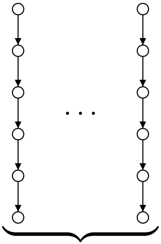
Form of communication



System architecture

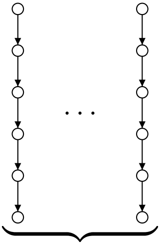


System architecture



!

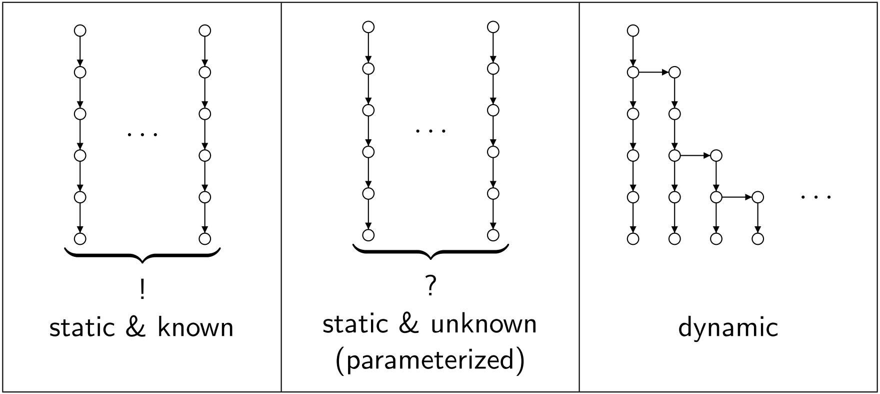
static & known



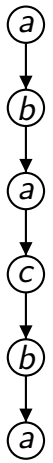
?

static & unknown
(parameterized)

System architecture

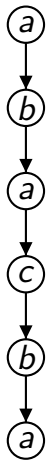


Type of single process

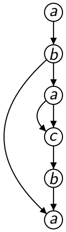


finite-state

Type of single process

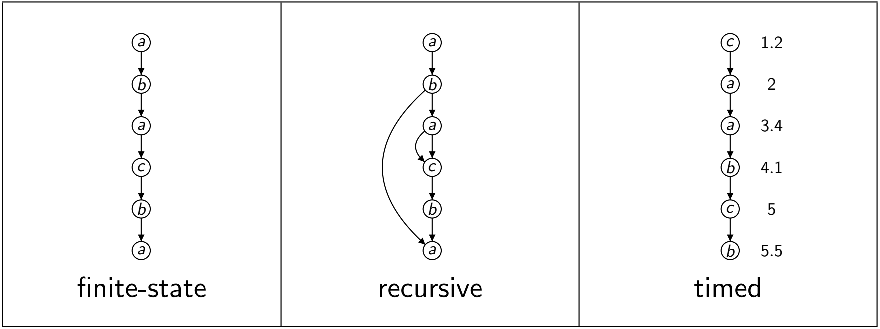


finite-state

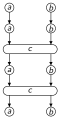
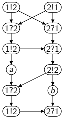
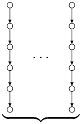
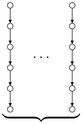
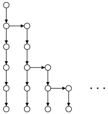
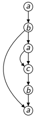
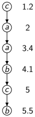


recursive

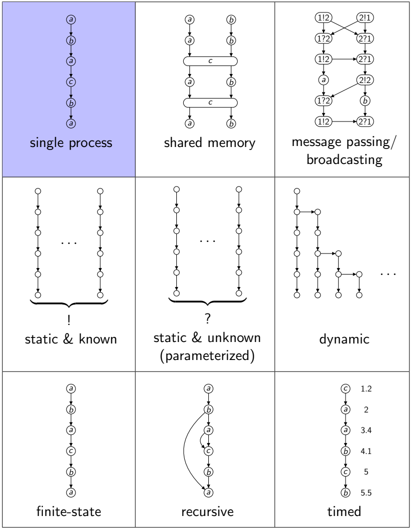
Type of single process



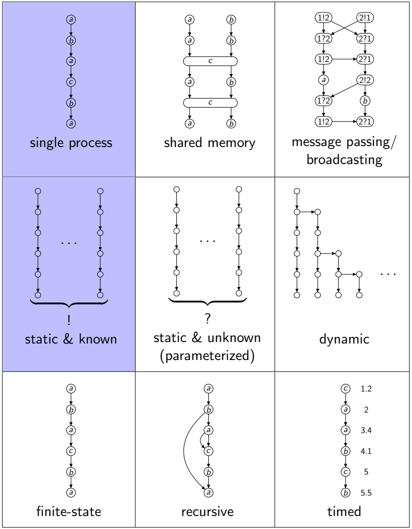
The various settings ...

 single process	 shared memory	 message passing/ broadcasting
 ! static & known	 ? static & unknown (parameterized)	 dynamic
 finite-state	 recursive	 timed

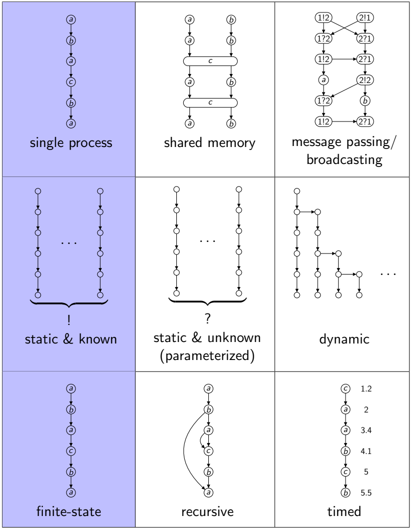
The various settings ...



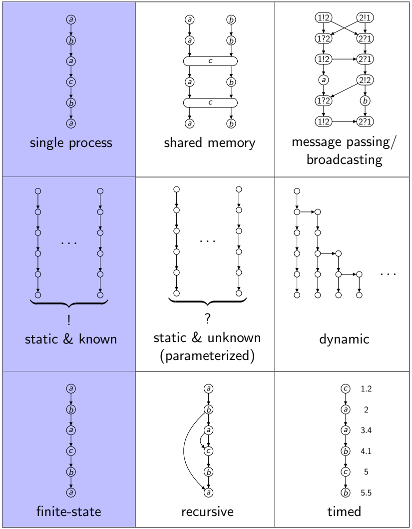
The various settings ...



The various settings ...



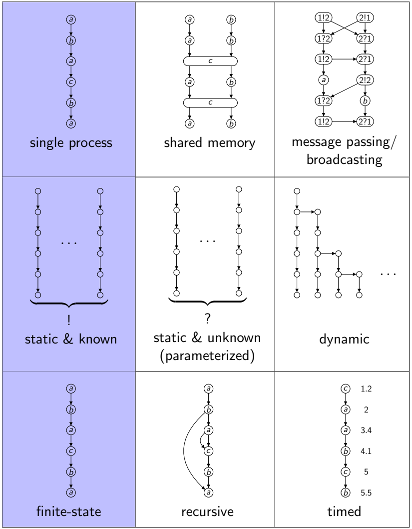
The various settings ...



Behavior

- Words

The various settings ...



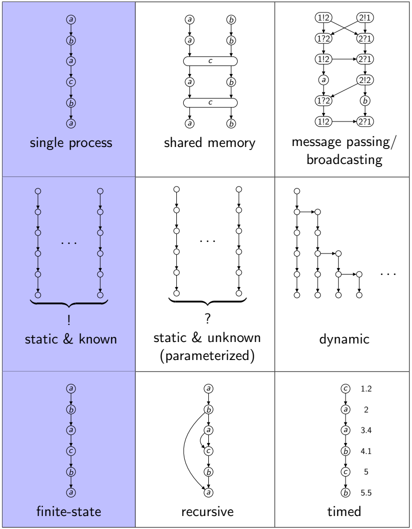
Behavior

- Words

System model

- Finite automata
- Kripke structures

The various settings ...



Behavior

- Words

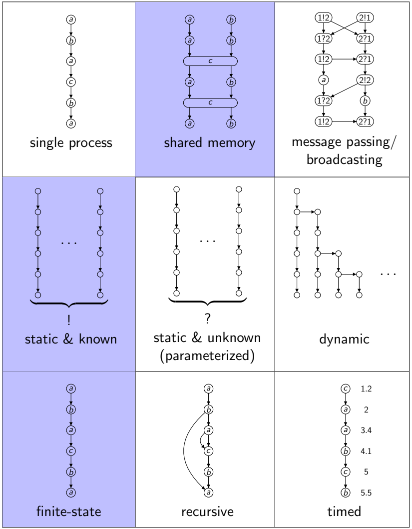
System model

- Finite automata
- Kripke structures

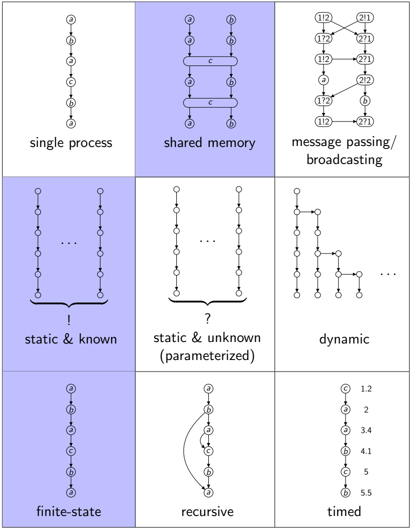
Specification

- Linear-time temporal logic (LTL)
- Monadic second-order logic (MSO)
- Regular expressions

The various settings ...



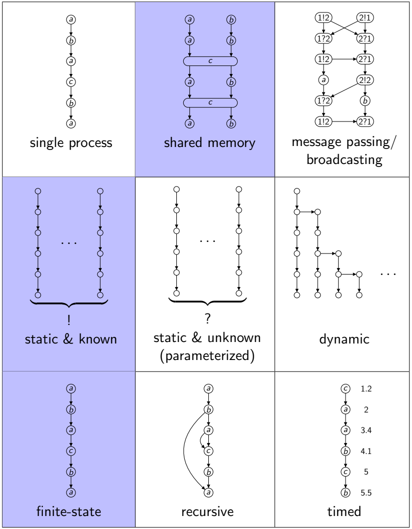
The various settings ...



Behavior

- Mazurkiewicz traces
  [Mazurkiewicz '86]

The various settings ...



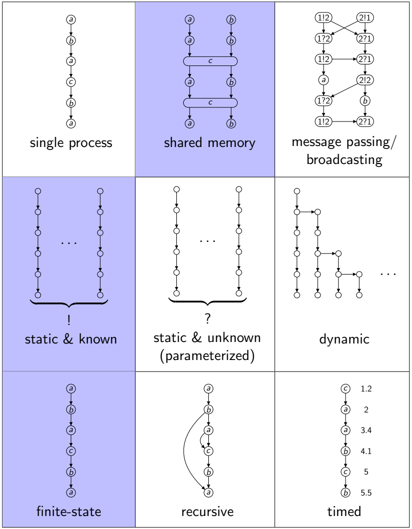
Behavior

- Mazurkiewicz traces
  [Mazurkiewicz '86]

System model

- Asynchronous automata
  [Zielonka '87]
- Asynchronous cellular automata

The various settings ...



Behavior

- Mazurkiewicz traces
  [Mazurkiewicz '86]

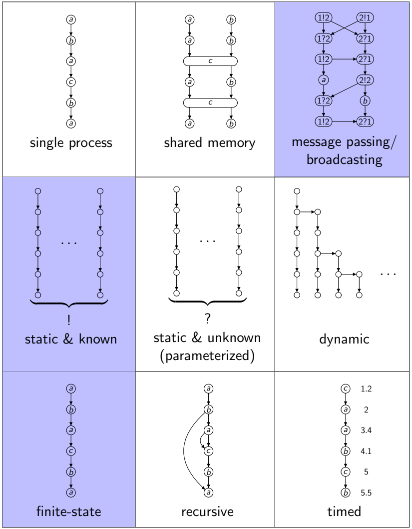
System model

- Asynchronous automata
  [Zielonka '87]
- Asynchronous cellular automata

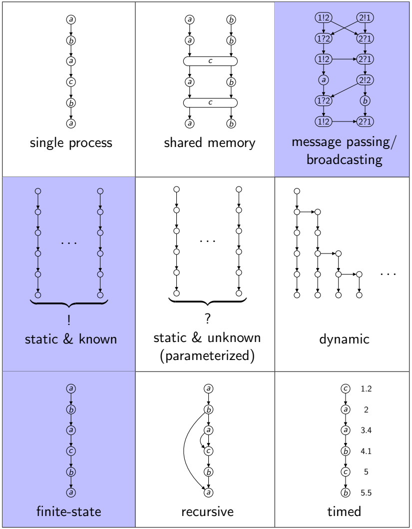
Specification

- Temporal logic (such as LTL)
- Monadic second-order logic (MSO)
- Regular (rational) expressions

The various settings ...



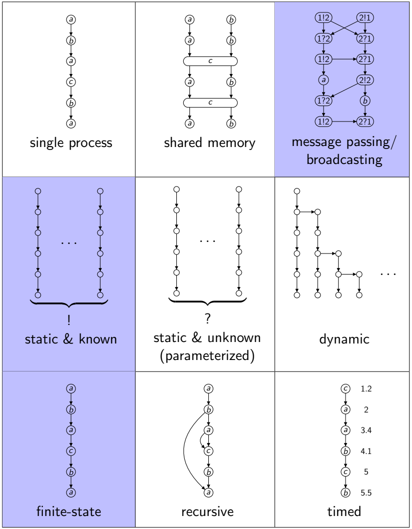
The various settings ...



Behavior

- Message sequence charts

The various settings ...



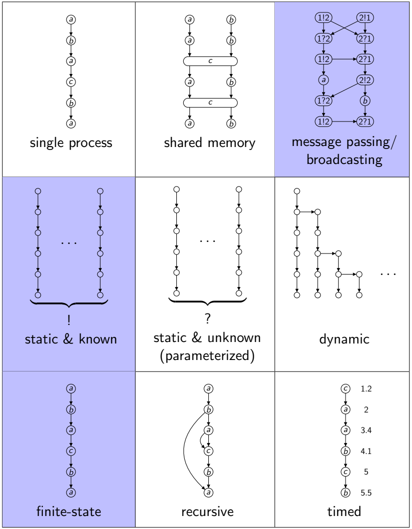
Behavior

- Message sequence charts

System model

- Communicating automata
  [Brand-Zafiropulo '83]
- Lossy channel systems
  [Finkel '87, Abdulla-Jonsson '96]

The various settings ...



Behavior

- Message sequence charts

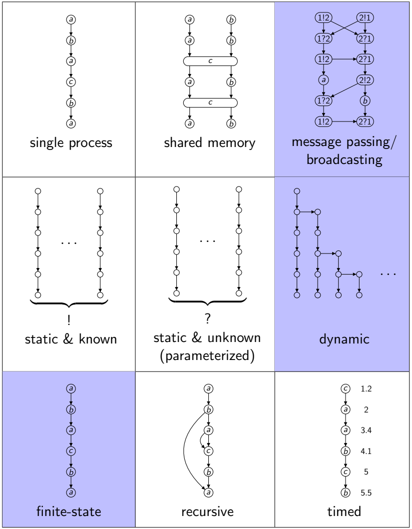
System model

- Communicating automata
  [Brand-Zafiropulo '83]
- Lossy channel systems
  [Finkel '87, Abdulla-Jonsson '96]

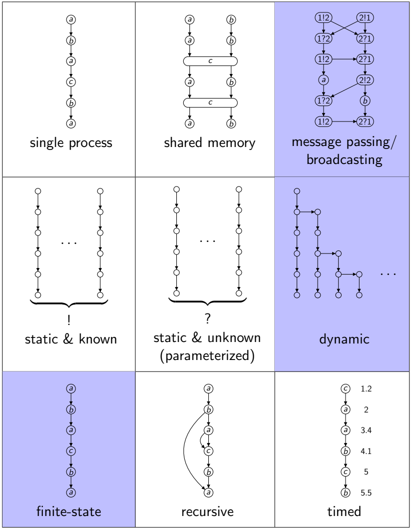
Specification

- Temporal logic
- Monadic second-order logic (MSO)
- High-level expressions

The various settings ...



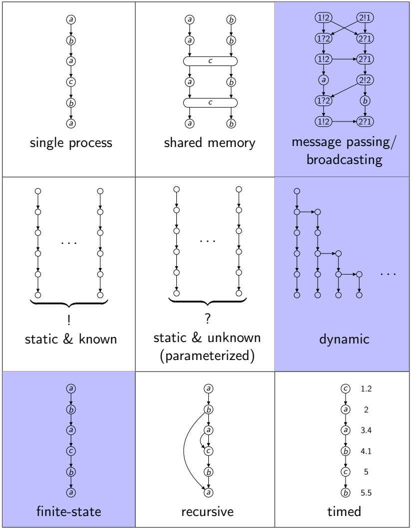
The various settings ...



Behavior

- Dynamic message sequence charts

The various settings ...



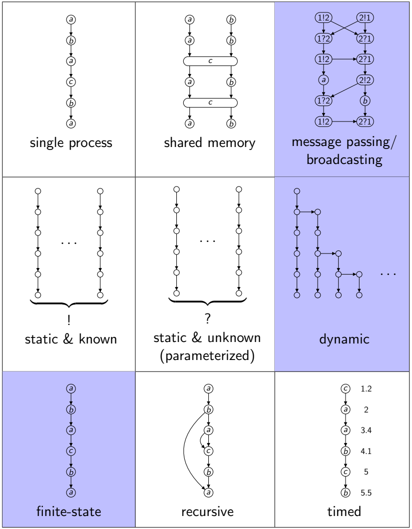
Behavior

- Dynamic message sequence charts

System model

- Dynamic communicating automata
  [B., Cyriac, Hélouët, Kara, Schwentick ’13]

The various settings ...



Behavior

- Dynamic message sequence charts

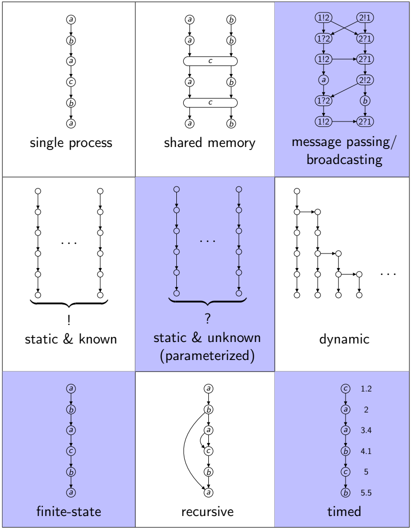
System model

- Dynamic communicating automata
  [B., Cyriac, Hélouët, Kara, Schwentick '13]

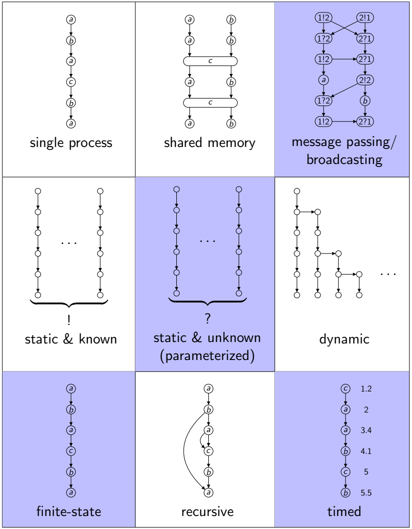
Specification

- High-level expressions with registers

The various settings ...



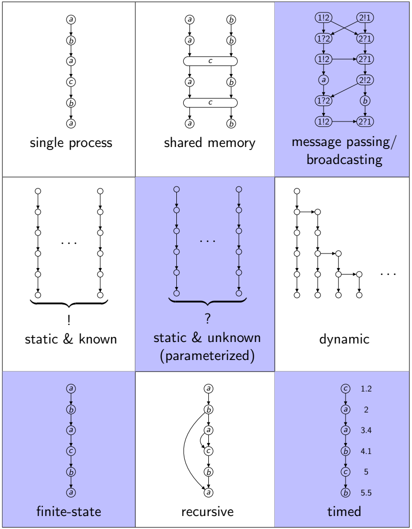
The various settings ...



Behavior

- Words ?

The various settings ...



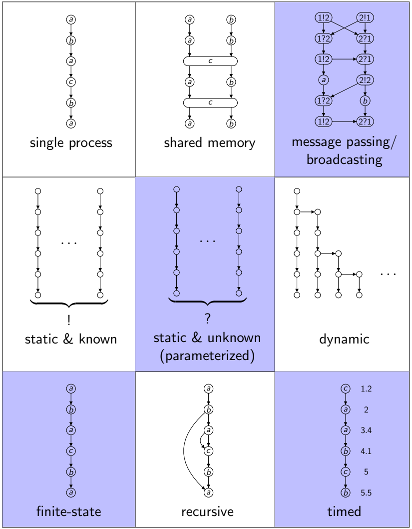
Behavior

- Words ?

System model

- Parametric ad-hoc networks
  [Delzanno-Sangnier et al. '10–'13]

The various settings ...



Behavior

- Words ?

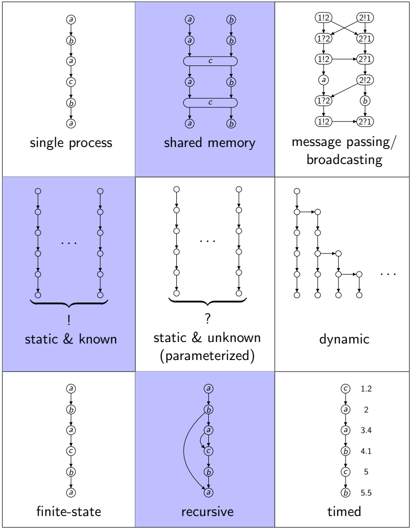
System model

- Parametric ad-hoc networks
  [Delzanno-Sangnier et al. '10–'13]

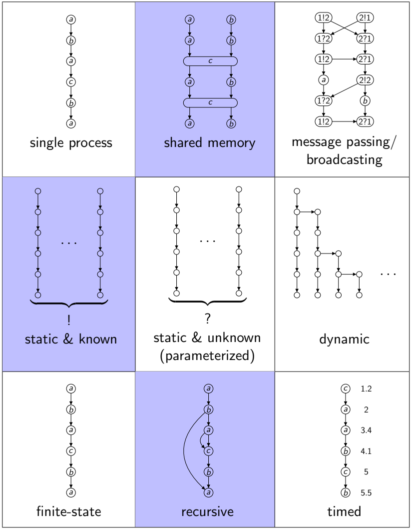
Specification

- Reachability questions

The various settings ...



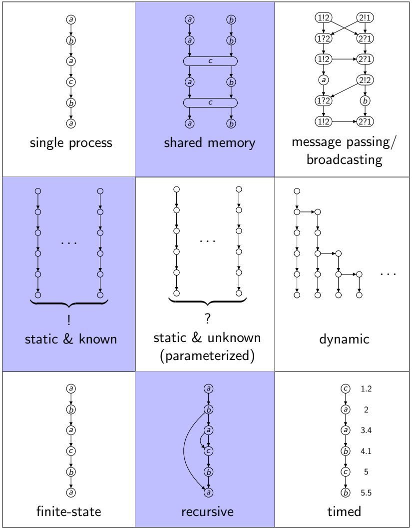
The various settings ...



Behavior

- Nested traces

The various settings ...



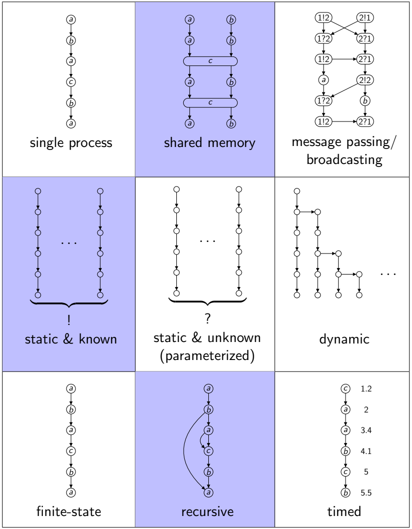
Behavior

- Nested traces

System model

- Multi-stack systems
  [La Torre et al. '07–'13], [Atig et al.]
- Nested-word automata
  [Alur et al. '04]

The various settings ...



Behavior

- Nested traces

System model

- Multi-stack systems
  [La Torre et al. '07–'13], [Atig et al.]
- Nested-word automata
  [Alur et al. '04]

Specification

- Temporal logic (such as LTL)
- Monadic second-order logic (MSO)
- Regular (rational) expressions

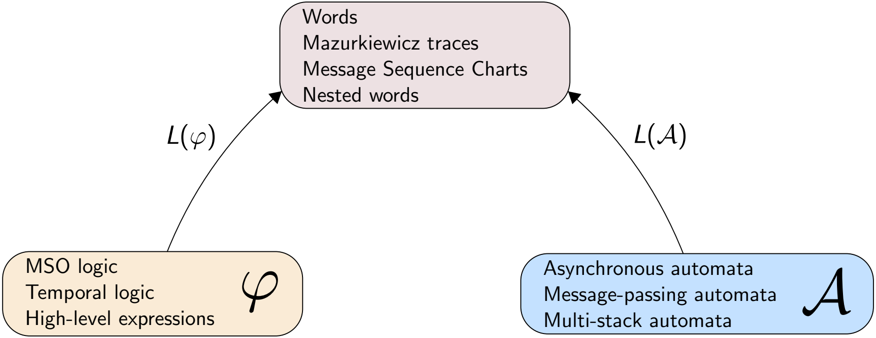
Landscape and Objectives

Words
Mazurkiewicz traces
Message Sequence Charts
Nested words

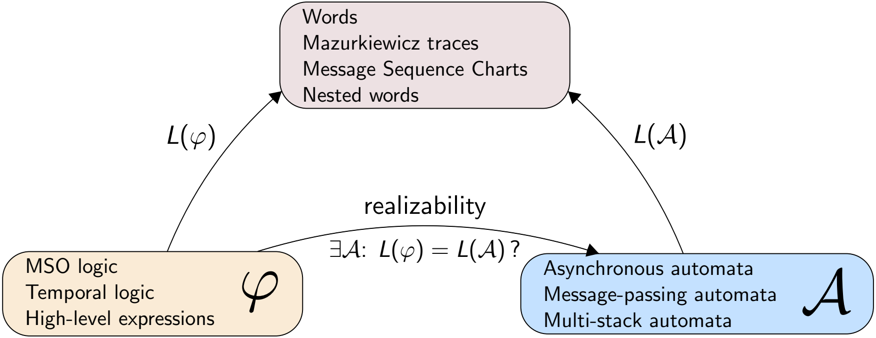
MSO logic
Temporal logic
High-level expressions
φ

Asynchronous automata
Message-passing automata
Multi-stack automata
$\mathcal{A}$

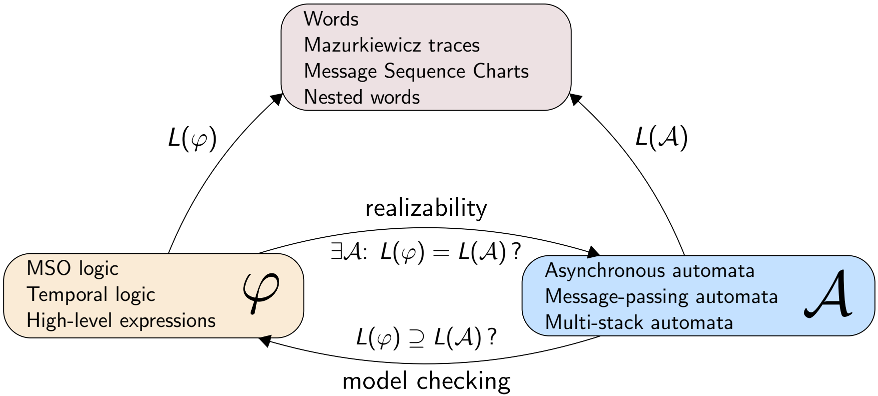
Landscape and Objectives



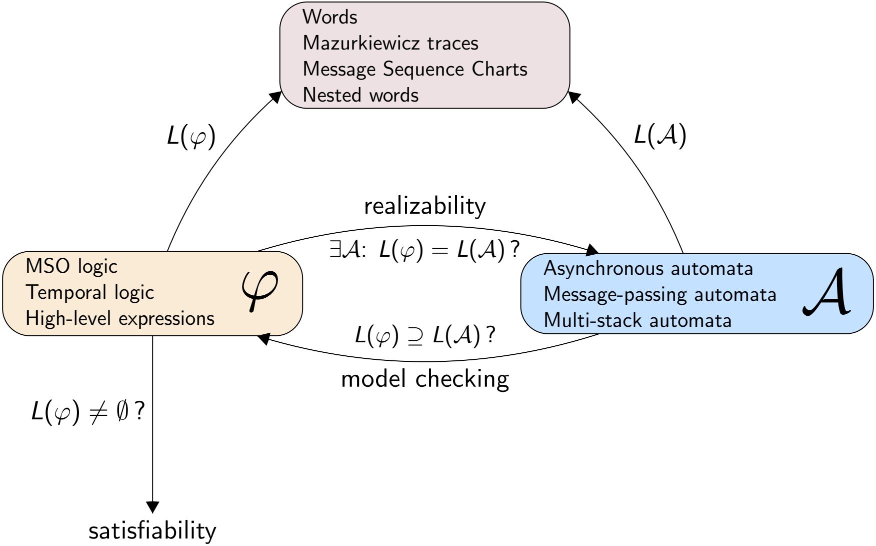
Landscape and Objectives



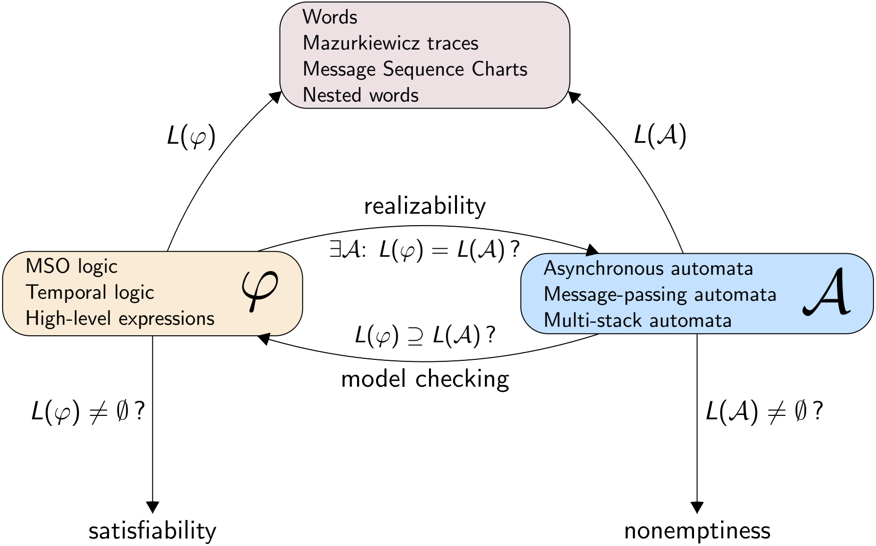
Landscape and Objectives



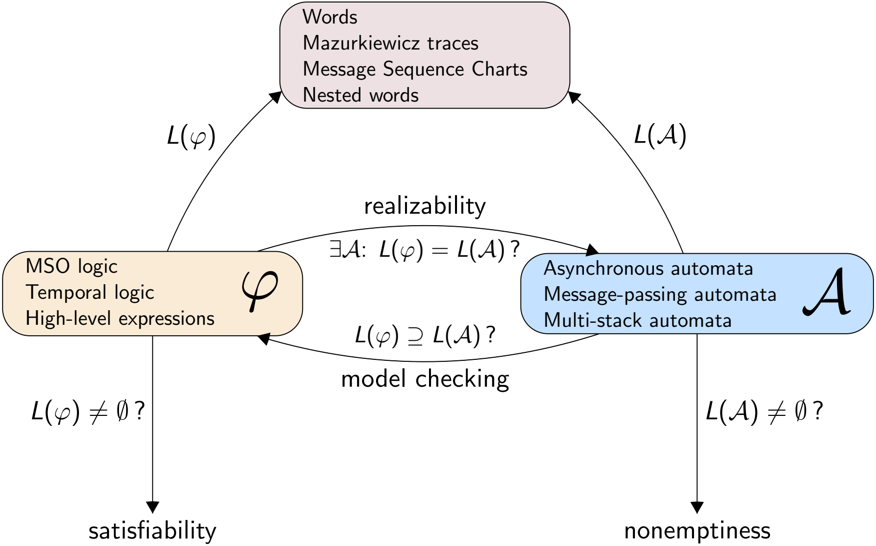
Landscape and Objectives



Landscape and Objectives



Landscape and Objectives: Linear-Time Setting



In this talk:

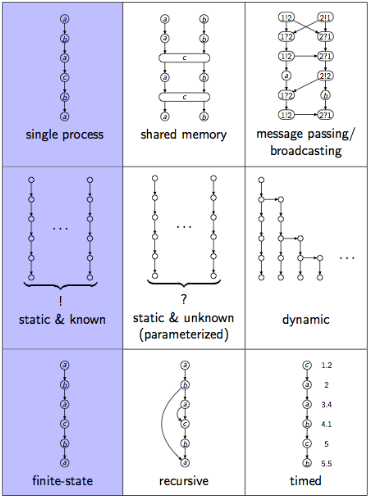
- Finite-State Sequential Systems
- Finite-State Shared-Memory Systems
- Recursive Shared-Memory Systems
- Message-Passing Systems

In this talk:

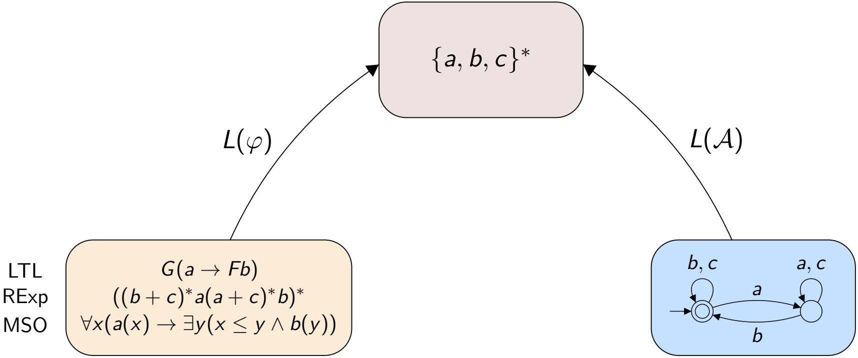
- Finite-State Sequential Systems
- Finite-State Shared-Memory Systems
- Recursive Shared-Memory Systems
- Message-Passing Systems

with static and known system architecture

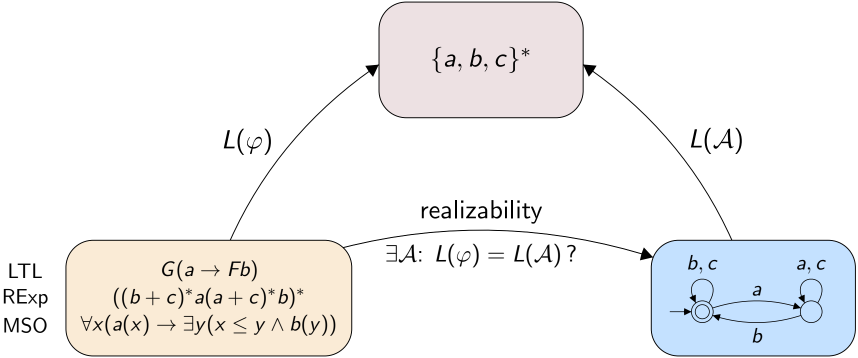
3. Finite-State Sequential Systems



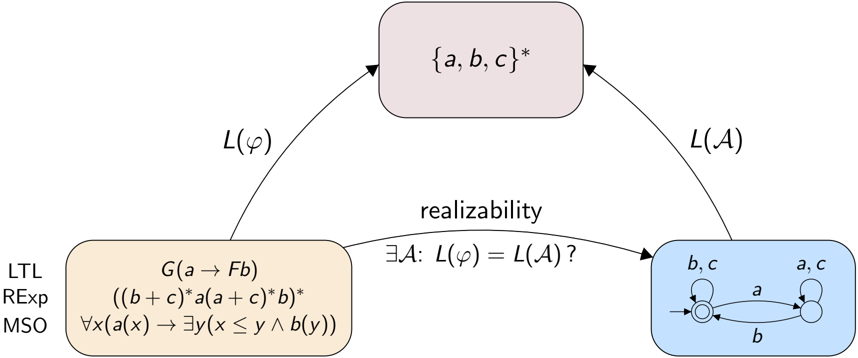
Finite-State Sequential Systems



Finite-State Sequential Systems



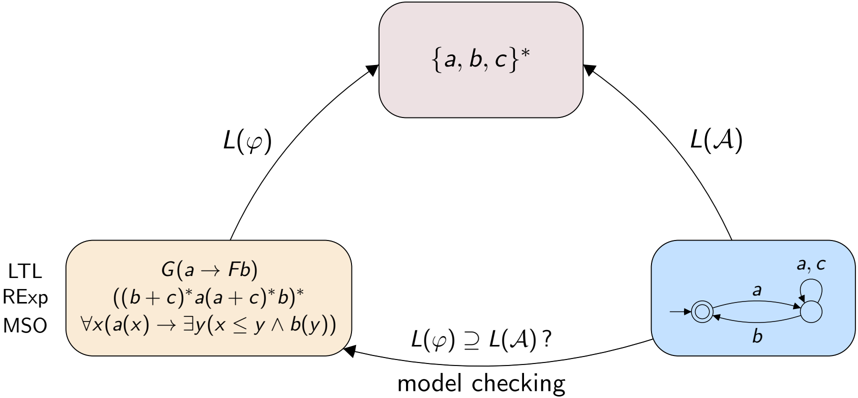
Finite-State Sequential Systems



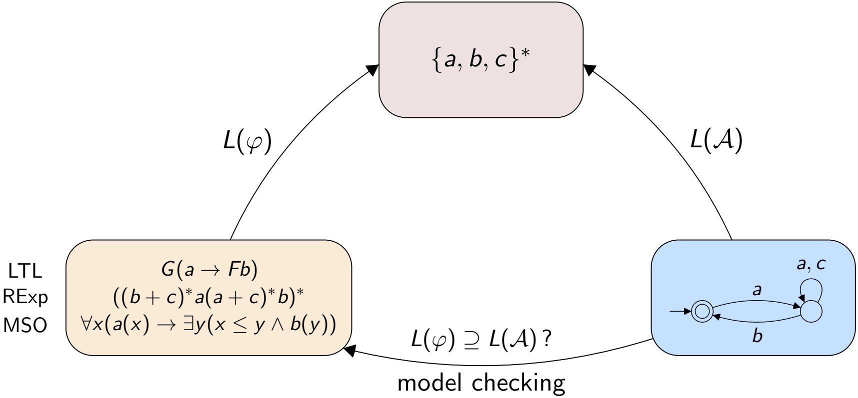
Theorem (Büchi-Elgot-Trakhtenbrot ’60s)

Every MSO formula is equivalent to some (deterministic) finite automaton.

Finite-State Sequential Systems



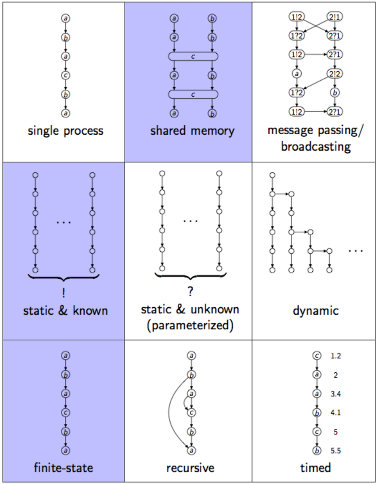
Finite-State Sequential Systems



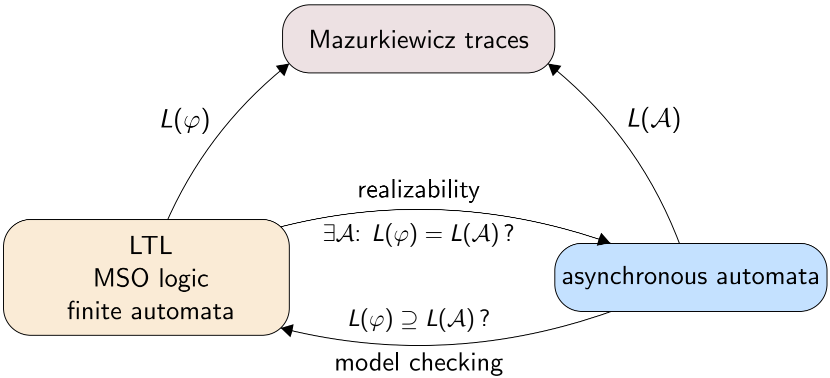
Theorem (Büchi-Elgot-Trakhtenbrot '60s; Sistla-Clarke '85)

Model checking against MSO is decidable, but nonelementary.
Model checking LTL is PSPACE-complete.

4. Finite-State Shared-Memory Systems



Finite-State Shared-Memory Systems



Asynchronous Automata and Mazurkiewicz Traces

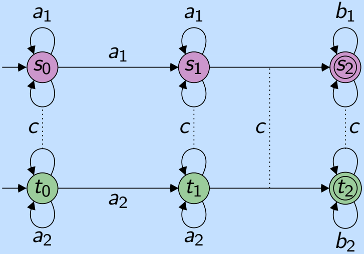
$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton

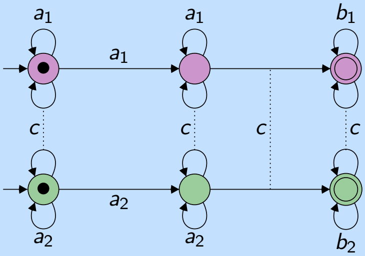
$$(s_0) \xrightarrow{a_1} (s_1)$$
$$(s_0, t_0) \xrightarrow{c} (s_0, t_0)$$
$$(s_1, t_1) \xrightarrow{c} (s_2, t_2)$$
$$(s_0, t_1) \not\xrightarrow{c} (s_0, t_1)$$



Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton

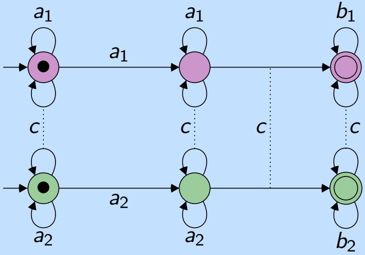


Mazurkiewicz Trace

Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton

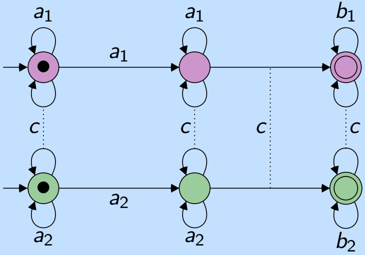


Mazurkiewicz Trace

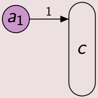
Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton



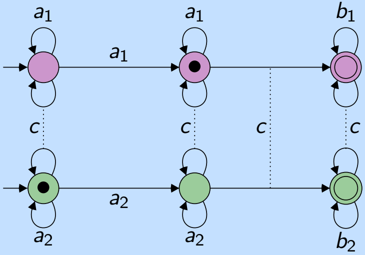
Mazurkiewicz Trace



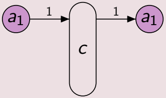
Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton



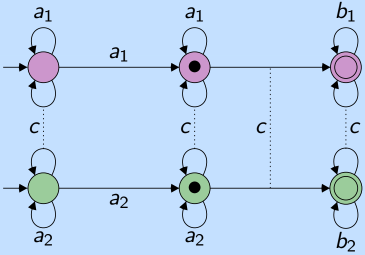
Mazurkiewicz Trace



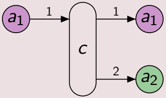
Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton



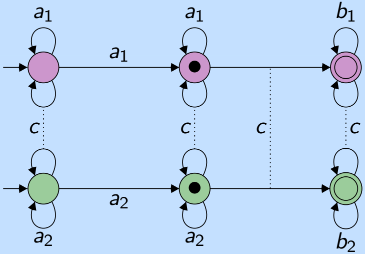
Mazurkiewicz Trace



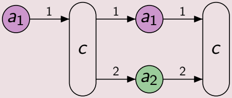
Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton



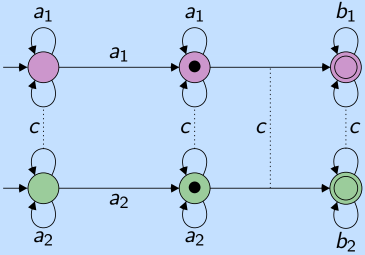
Mazurkiewicz Trace



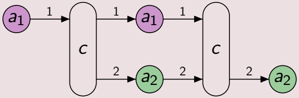
Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton



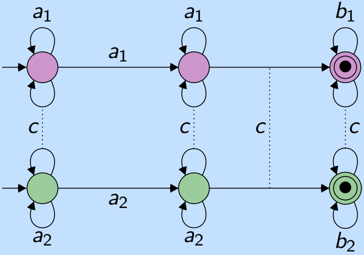
Mazurkiewicz Trace



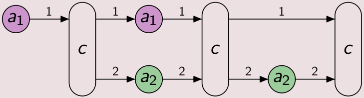
Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton



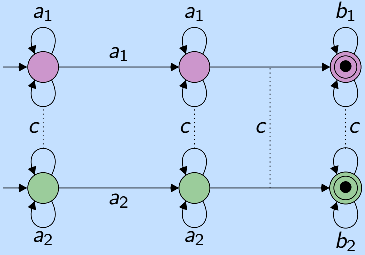
Mazurkiewicz Trace



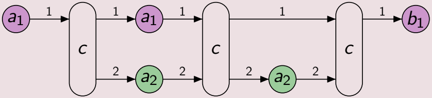
Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton



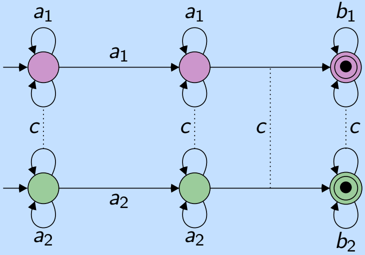
Mazurkiewicz Trace



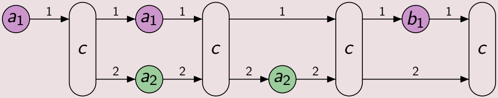
Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton



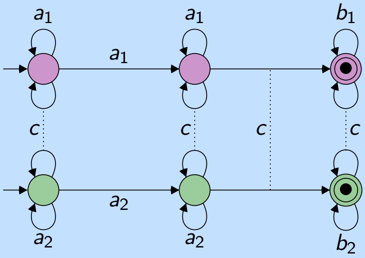
Mazurkiewicz Trace



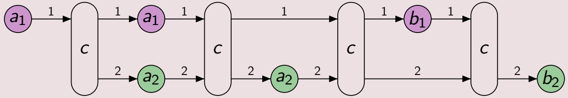
Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton



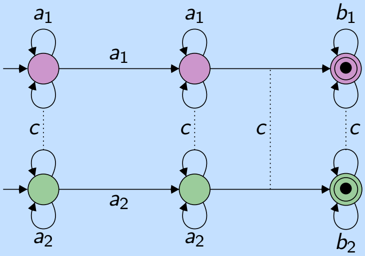
Mazurkiewicz Trace



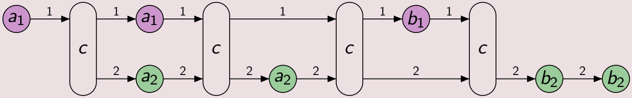
Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton



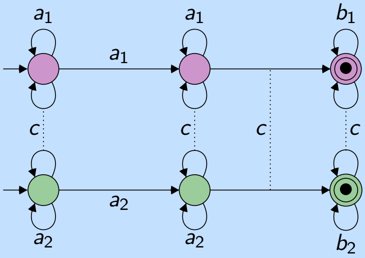
Mazurkiewicz Trace



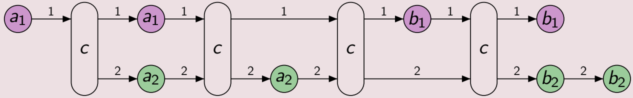
Asynchronous Automata and Mazurkiewicz Traces

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$

Asynchronous Automaton

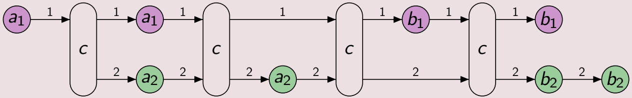


Mazurkiewicz Trace



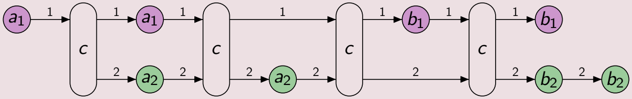
Mazurkiewicz Traces and Their Linearizations

Mazurkiewicz Trace $\quad t = (E, \rightarrow_1, \rightarrow_2, \lambda) \quad \lambda : E \rightarrow \Sigma \stackrel{\text{def}}{=} \Sigma_1 \cup \Sigma_2$

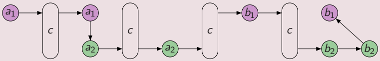


Mazurkiewicz Traces and Their Linearizations

Mazurkiewicz Trace $\quad t = (E, \rightarrow_1, \rightarrow_2, \lambda) \quad \lambda : E \rightarrow \Sigma \stackrel{\text{def}}{=} \Sigma_1 \cup \Sigma_2$

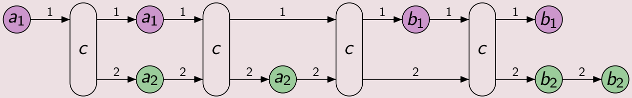


Linearizations $\quad w \in Lin(t) \subseteq \Sigma^* \quad \rightsquigarrow \quad trace(w) = t$

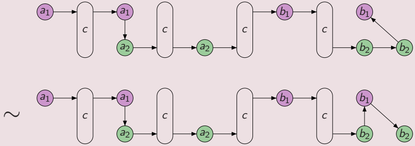


Mazurkiewicz Traces and Their Linearizations

Mazurkiewicz Trace $\quad t = (E, \rightarrow_1, \rightarrow_2, \lambda) \quad \lambda : E \rightarrow \Sigma \stackrel{\text{def}}{=} \Sigma_1 \cup \Sigma_2$

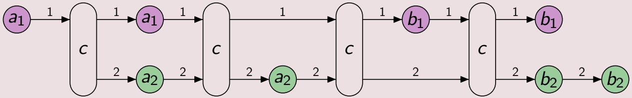


Linearizations $\quad w \in Lin(t) \subseteq \Sigma^* \quad \rightsquigarrow \quad trace(w) = t$

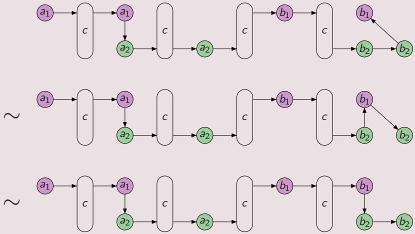


Mazurkiewicz Traces and Their Linearizations

Mazurkiewicz Trace $\quad t = (E, \rightarrow_1, \rightarrow_2, \lambda) \quad \lambda : E \rightarrow \Sigma \stackrel{\text{def}}{=} \Sigma_1 \cup \Sigma_2$

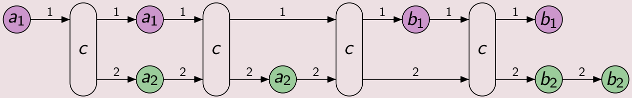


Linearizations $\quad w \in Lin(t) \subseteq \Sigma^* \quad \rightsquigarrow \quad trace(w) = t$

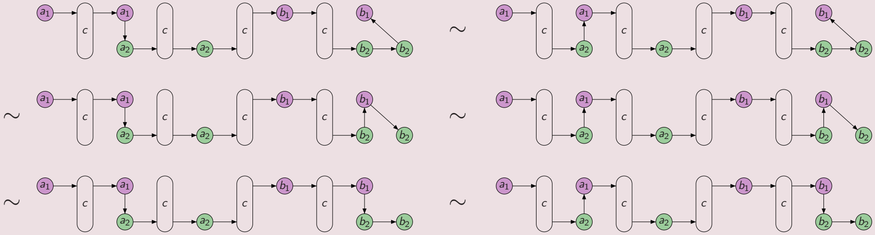


Mazurkiewicz Traces and Their Linearizations

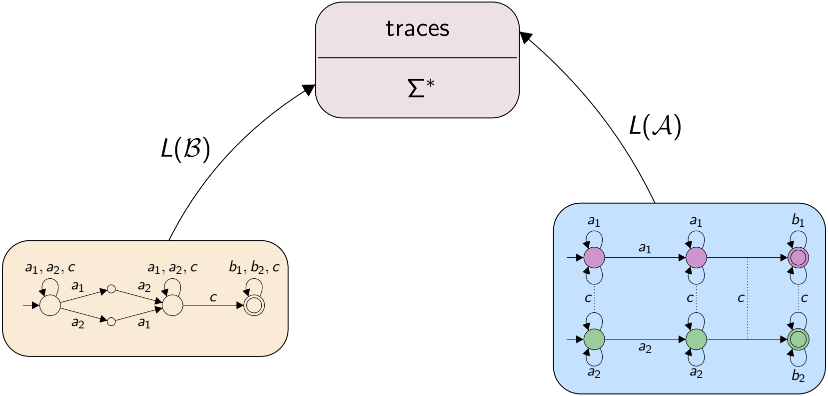
Mazurkiewicz Trace $\quad t = (E, \to_1, \to_2, \lambda) \quad \lambda : E \to \Sigma \stackrel{\text{def}}{=} \Sigma_1 \cup \Sigma_2$



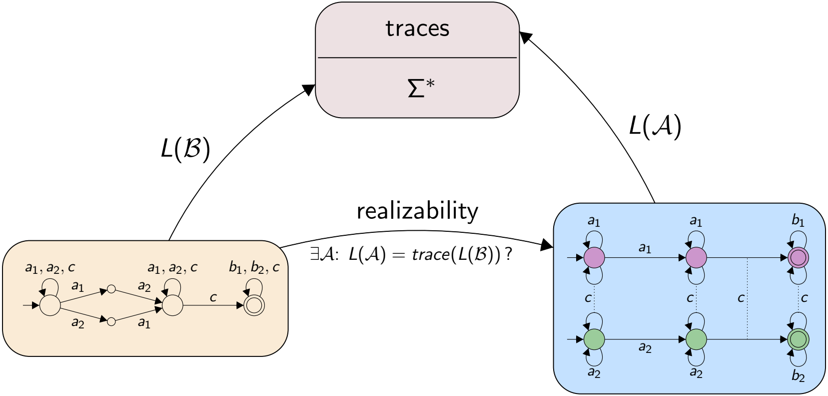
Linearizations $\quad w \in Lin(t) \subseteq \Sigma^* \quad \rightsquigarrow \quad trace(w) = t$



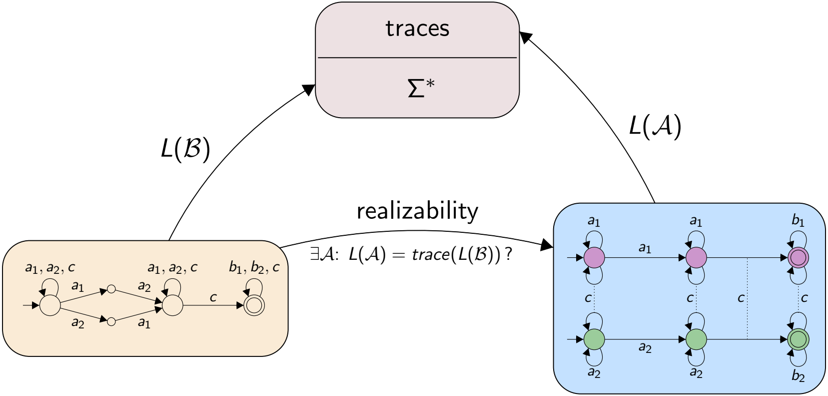
Finite-State Shared-Memory Systems



Finite-State Shared-Memory Systems



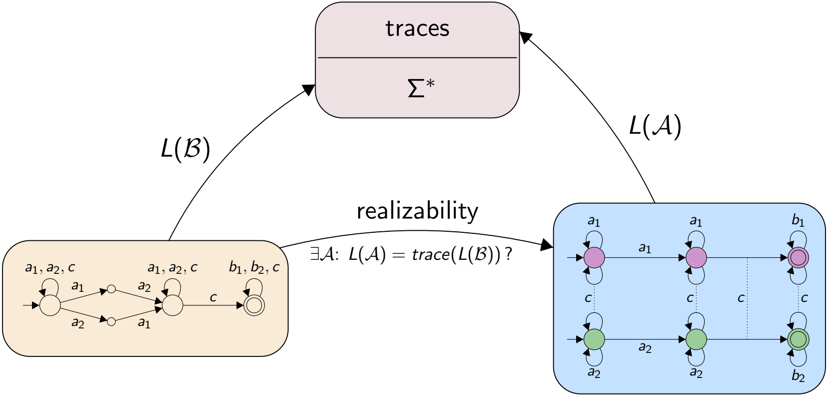
Finite-State Shared-Memory Systems



Theorem (Sakarovitch '92)

Realizability for regular specifications is undecidable.

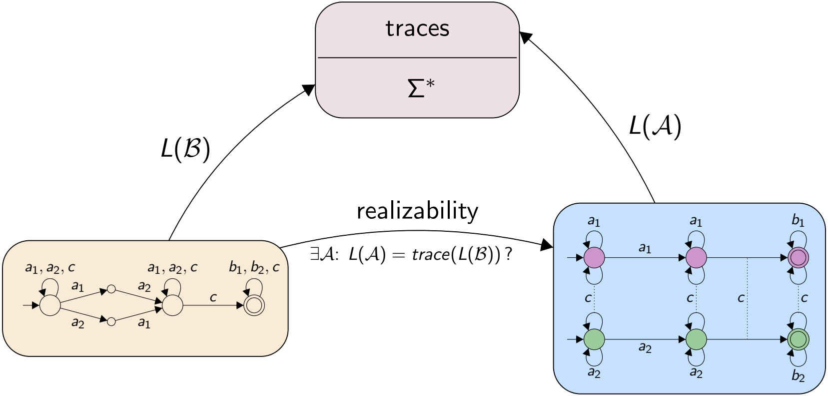
Finite-State Shared-Memory Systems



Theorem (Zielonka '87)

Let $L \subseteq \Sigma^*$ be a $\sim$-closed regular language. There is a (deterministic) asynchronous automaton $\mathcal{A}$ such that $L(\mathcal{A}) = trace(L)$.

Finite-State Shared-Memory Systems



Theorem (Muscholl '94, Peled-Wilke-Wolper '98)

It is decidable (PSPACE-complete) if the language of a finite automaton is $\sim$-closed (PTIME for deterministic automata).

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

- $x \rightarrow_p y$ — x and y are successive events on process $p \in Proc$

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

- $x \rightarrow_p y$ — x and y are successive events on process $p \in Proc$
- $a(x)$ — event x is labeled with $a \in \Sigma$

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

- $x \rightarrow_p y$ — x and y are successive events on process $p \in Proc$
- $a(x)$ — event x is labeled with $a \in \Sigma$
- $x = y$

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

- $x \rightarrow_p y$ — x and y are successive events on process $p \in Proc$
- $a(x)$ — event x is labeled with $a \in \Sigma$
- $x = y$
- $x \in X$ — event x is contained in set of events X

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

▸ $x \rightarrow_p y$	x and y are successive events on process $p \in Proc$
▸ $a(x)$	event x is labeled with $a \in \Sigma$
▸ $x = y$	
▸ $x \in X$	event x is contained in set of events X
▸ $\exists x \varphi$	there is event x such that φ

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

- $x \rightarrow_p y$ — x and y are successive events on process $p \in \mathit{Proc}$
- $a(x)$ — event x is labeled with $a \in \Sigma$
- $x = y$
- $x \in X$ — event x is contained in set of events X
- $\exists x \varphi$ — there is event x such that φ
- $\exists X \varphi$ — there is set of event X such that φ

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

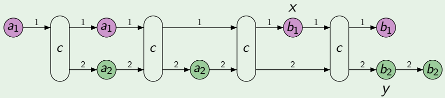
- $x \rightarrow_p y$ — x and y are successive events on process $p \in Proc$
- $a(x)$ — event x is labeled with $a \in \Sigma$
- $x = y$
- $x \in X$ — event x is contained in set of events X
- $\exists x \varphi$ — there is event x such that φ
- $\exists X \varphi$ — there is set of event X such that φ
- $\neg\varphi \qquad \varphi \vee \psi$

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

- $x \rightarrow_p y$ — x and y are successive events on process $p \in \mathit{Proc}$
- $a(x)$ — event x is labeled with $a \in \Sigma$
- $x = y$
- $x \in X$ — event x is contained in set of events X
- $\exists x \varphi$ — there is event x such that φ
- $\exists X \varphi$ — there is set of event X such that φ
- $\neg\varphi \qquad \varphi \vee \psi$

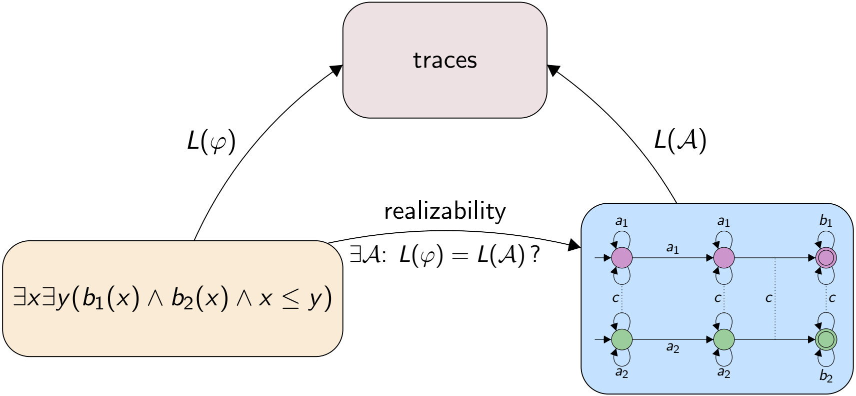
Example



$\models \; \exists x \exists y (b_1(x) \wedge b_2(x) \wedge x \leq y)$

where $\leq = (\rightarrow_1 \cup \rightarrow_2)^*$

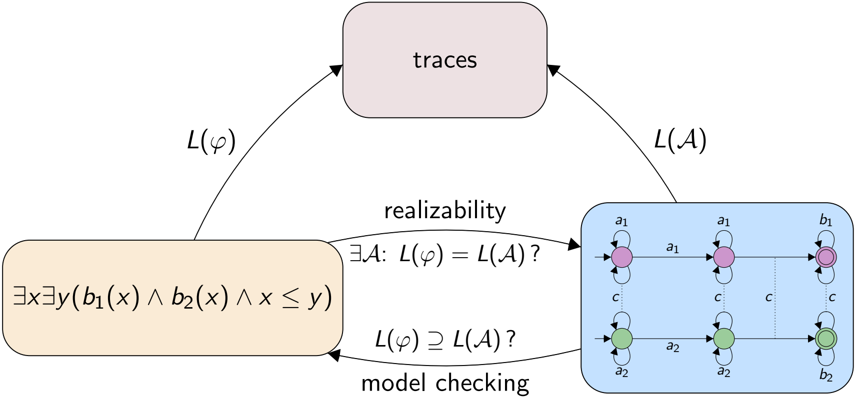
Finite-State Shared-Memory Systems



Theorem (Thomas '90)

MSO logic and asynchronous automata are expressively equivalent.

Finite-State Shared-Memory Systems



Theorem (Thomas '90)

MSO logic and asynchronous automata are expressively equivalent.

⇨ MSO model checking is decidable.

Global Temporal Logic

Global Temporal Logic

$\mathsf{LTrL}_{\forall} \qquad \varphi ::= \mathtt{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \, \mathsf{U}_{\forall} \, \varphi_2 \mid \neg \varphi \mid \varphi_1 \vee \varphi_2 \qquad a \in \Sigma$

Global Temporal Logic

Global Temporal Logic

$\mathrm{LTrL}_\forall$	$\varphi ::= \mathtt{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \, \mathsf{U}_\forall \, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$	$a \in \Sigma$
$\mathrm{LTrL}_\exists$	$\varphi ::= \qquad \mathsf{U}_\exists$	

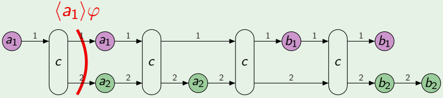
Global Temporal Logic

Global Temporal Logic

$\mathrm{LTrL}_\forall \quad \varphi ::= \mathtt{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \, \mathsf{U}_\forall \, \varphi_2 \mid \neg \varphi \mid \varphi_1 \vee \varphi_2 \qquad a \in \Sigma$

$\mathrm{LTrL}_\exists \quad \varphi ::= \qquad \mathsf{U}_\exists$

Semantics



$\models \ \langle a_1 \rangle \varphi$

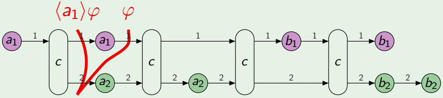
Global Temporal Logic

Global Temporal Logic

$\mathrm{LTrL}_\forall \quad \varphi ::= \mathtt{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \mathsf{U}_\forall \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \qquad a \in \Sigma$

$\mathrm{LTrL}_\exists \quad \varphi ::= \qquad \mathsf{U}_\exists$

Semantics



$\models \langle a_1 \rangle \varphi$

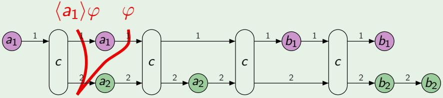
Global Temporal Logic

Global Temporal Logic

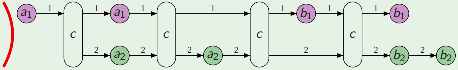
$\mathrm{LTrL}_\forall \qquad \varphi ::= \mathtt{tt} \mid \langle a\rangle\varphi \mid \varphi_1 \,\mathsf{U}_\forall\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \qquad a \in \Sigma$

$\mathrm{LTrL}_\exists \qquad \varphi ::= \qquad \mathsf{U}_\exists$

Semantics



$\models \;\; \langle a_1\rangle\varphi$



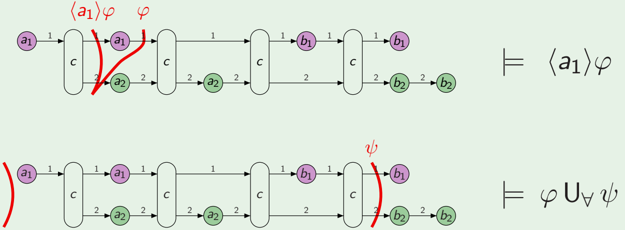
$\models \;\; \varphi \,\mathsf{U}_\forall\, \psi$

Global Temporal Logic

Global Temporal Logic

LTrL$_\forall$	$\varphi ::= \mathtt{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \, \mathsf{U}_\forall \, \varphi_2 \mid \neg \varphi \mid \varphi_1 \vee \varphi_2$	$a \in \Sigma$
LTrL$_\exists$	$\varphi ::= \qquad\qquad \mathsf{U}_\exists$	

Semantics



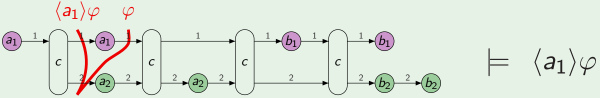
Global Temporal Logic

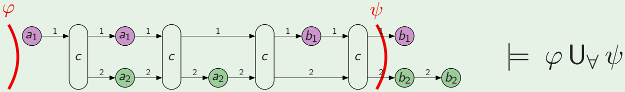
Global Temporal Logic

$\mathsf{LTrL}_\forall$ $\quad \varphi ::= \mathtt{tt} \mid \langle a\rangle\varphi \mid \varphi_1 \,\mathsf{U}_\forall\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \qquad a \in \Sigma$

$\mathsf{LTrL}_\exists$ $\quad \varphi ::= \qquad \mathsf{U}_\exists$

Semantics



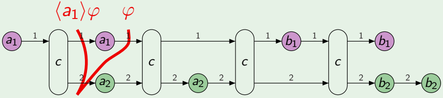


Global Temporal Logic

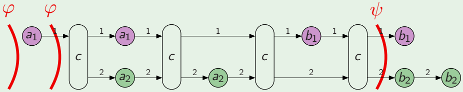
Global Temporal Logic

$\mathsf{LTrL}_\forall$	$\varphi ::= \mathtt{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \, \mathsf{U}_\forall \, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$	$a \in \Sigma$
$\mathsf{LTrL}_\exists$	$\varphi ::= \qquad \mathsf{U}_\exists$	

Semantics



$\models \ \langle a_1 \rangle \varphi$



$\models \ \varphi \, \mathsf{U}_\forall \, \psi$

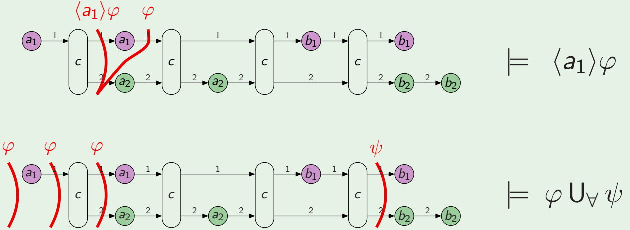
Global Temporal Logic

Global Temporal Logic

$$\text{LTrL}_\forall \qquad \varphi ::= \mathtt{tt} \mid \langle a\rangle\varphi \mid \varphi_1 \,\mathsf{U}_\forall\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \qquad a \in \Sigma$$

$$\text{LTrL}_\exists \qquad \varphi ::= \qquad \mathsf{U}_\exists$$

Semantics

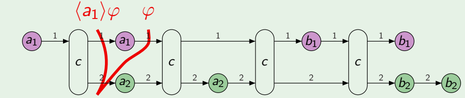


Global Temporal Logic

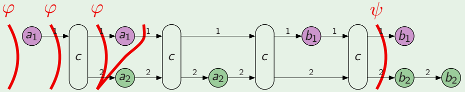
Global Temporal Logic

$\mathsf{LTrL}_\forall$	$\varphi ::= \mathtt{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \, \mathsf{U}_\forall \, \varphi_2 \mid \neg \varphi \mid \varphi_1 \vee \varphi_2$	$a \in \Sigma$
$\mathsf{LTrL}_\exists$	$\varphi ::= \qquad\qquad \mathsf{U}_\exists$	

Semantics



$\models \ \langle a_1 \rangle \varphi$



$\models \ \varphi \, \mathsf{U}_\forall \, \psi$

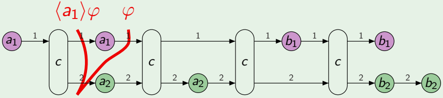
Global Temporal Logic

Global Temporal Logic

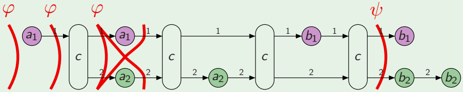
$\mathrm{LTrL}_\forall \quad \varphi ::= \mathtt{tt} \mid \langle a\rangle\varphi \mid \varphi_1 \mathsf{U}_\forall\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \qquad a \in \Sigma$

$\mathrm{LTrL}_\exists \quad \varphi ::= \qquad \mathsf{U}_\exists$

Semantics



$\models \ \langle a_1\rangle\varphi$



$\models \ \varphi\, \mathsf{U}_\forall\, \psi$

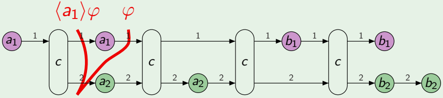
Global Temporal Logic

Global Temporal Logic

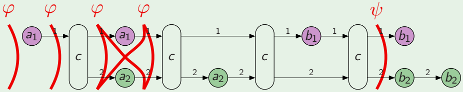
$\mathrm{LTrL}_\forall \quad \varphi ::= \mathtt{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \, \mathsf{U}_\forall \, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \qquad a \in \Sigma$

$\mathrm{LTrL}_\exists \quad \varphi ::= \qquad \mathsf{U}_\exists$

Semantics



$\models \langle a_1 \rangle \varphi$



$\models \varphi \, \mathsf{U}_\forall \, \psi$

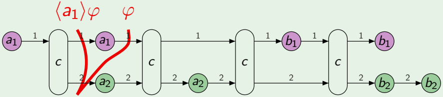
Global Temporal Logic

Global Temporal Logic

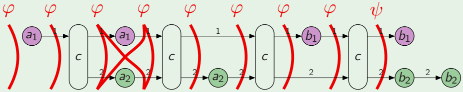
$$\mathrm{LTrL}_\forall \qquad \varphi ::= \mathtt{tt} \mid \langle a\rangle\varphi \mid \varphi_1 \mathsf{U}_\forall \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \qquad a \in \Sigma$$

$$\mathrm{LTrL}_\exists \qquad \varphi ::= \qquad \mathsf{U}_\exists$$

Semantics



$\models \ \langle a_1\rangle\varphi$



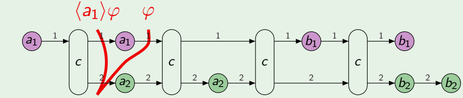
$\models \ \varphi \mathsf{U}_\forall \psi$

Global Temporal Logic

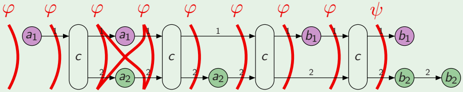
Global Temporal Logic

LTrL$_\forall$	$\varphi ::= \texttt{tt} \mid \langle a\rangle\varphi \mid \varphi_1 \,\mathsf{U}_\forall\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$	$a \in \Sigma$
LTrL$_\exists$	$\varphi ::= \quad \mathsf{U}_\exists$	

Semantics



$\models \quad \langle a_1\rangle\varphi$



$\models \quad \texttt{tt}\,\mathsf{U}_\forall\,\langle b_1\rangle\langle b_2\rangle\texttt{tt}$

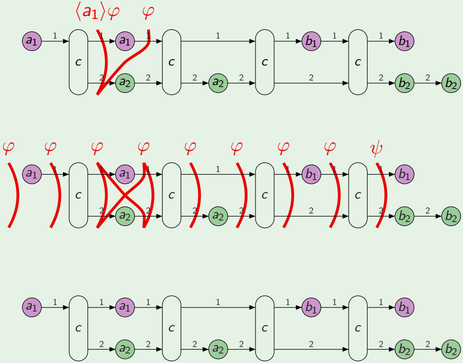
Global Temporal Logic

Global Temporal Logic

$\mathrm{LTrL}_\forall \qquad \varphi ::= \mathtt{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \,\mathsf{U}_\forall\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \qquad a \in \Sigma$

$\mathrm{LTrL}_\exists \qquad \varphi ::= \qquad \mathsf{U}_\exists$

Semantics



$\models \quad \langle a_1 \rangle \varphi$

$\models \quad \mathtt{tt} \,\mathsf{U}_\forall\, \langle b_1 \rangle \langle b_2 \rangle \mathtt{tt}$

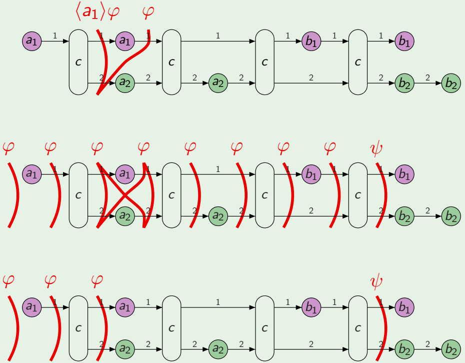
$\models \quad \varphi \,\mathsf{U}_\exists\, \psi$

Global Temporal Logic

Global Temporal Logic

$\mathrm{LTrL}_\forall$	$\varphi ::= \mathtt{tt} \mid \langle a\rangle\varphi \mid \varphi_1 \,\mathsf{U}_\forall\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$	$a \in \Sigma$
$\mathrm{LTrL}_\exists$	$\varphi ::= \qquad \mathsf{U}_\exists$	

Semantics



$\models \langle a_1\rangle\varphi$

$\models \mathtt{tt}\,\mathsf{U}_\forall\,\langle b_1\rangle\langle b_2\rangle\mathtt{tt}$

$\models \varphi\,\mathsf{U}_\exists\,\psi$

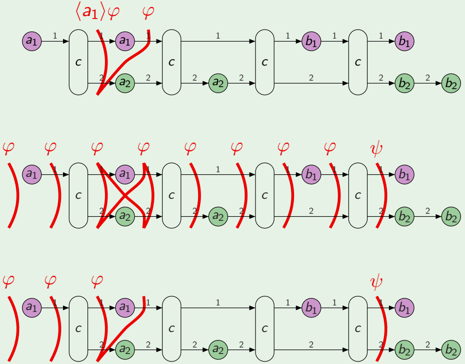
Global Temporal Logic

Global Temporal Logic

$\mathrm{LTrL}_\forall \quad \varphi ::= \mathtt{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \, \mathsf{U}_\forall \, \varphi_2 \mid \neg \varphi \mid \varphi_1 \vee \varphi_2 \qquad a \in \Sigma$

$\mathrm{LTrL}_\exists \quad \varphi ::= \qquad \mathsf{U}_\exists$

Semantics



$\models \langle a_1 \rangle \varphi$

$\models \mathtt{tt} \, \mathsf{U}_\forall \, \langle b_1 \rangle \langle b_2 \rangle \mathtt{tt}$

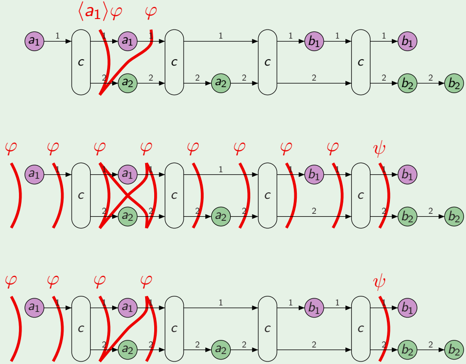
$\models \varphi \, \mathsf{U}_\exists \, \psi$

Global Temporal Logic

Global Temporal Logic

LTrL$_\forall$	$\varphi ::= \mathtt{tt} \mid \langle a\rangle\varphi \mid \varphi_1 \,\mathsf{U}_\forall\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$	$a \in \Sigma$
LTrL$_\exists$	$\varphi ::= \qquad\qquad \mathsf{U}_\exists$	

Semantics



$\models \; \langle a_1\rangle\varphi$

$\models \; \mathtt{tt}\,\mathsf{U}_\forall\,\langle b_1\rangle\langle b_2\rangle\mathtt{tt}$

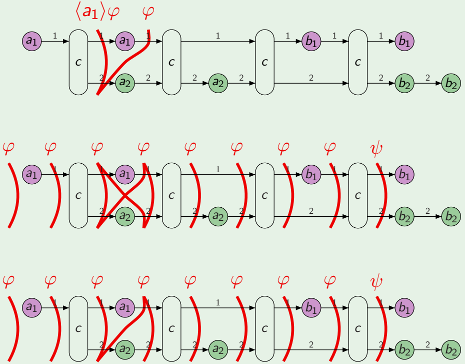
$\models \; \varphi\,\mathsf{U}_\exists\,\psi$

Global Temporal Logic

Global Temporal Logic

$\mathrm{LTrL}_\forall$	$\varphi ::= \mathtt{tt} \mid \langle a\rangle\varphi \mid \varphi_1 \,\mathsf{U}_\forall\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$	$a \in \Sigma$
$\mathrm{LTrL}_\exists$	$\varphi ::= \qquad\qquad \mathsf{U}_\exists$	

Semantics

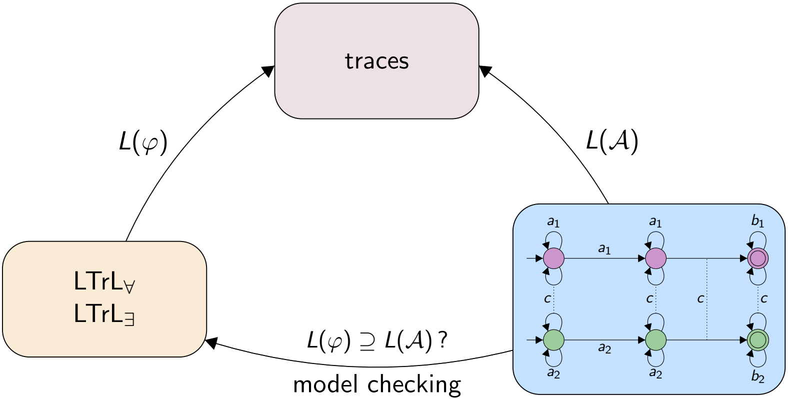


$\models \;\langle a_1\rangle\varphi$

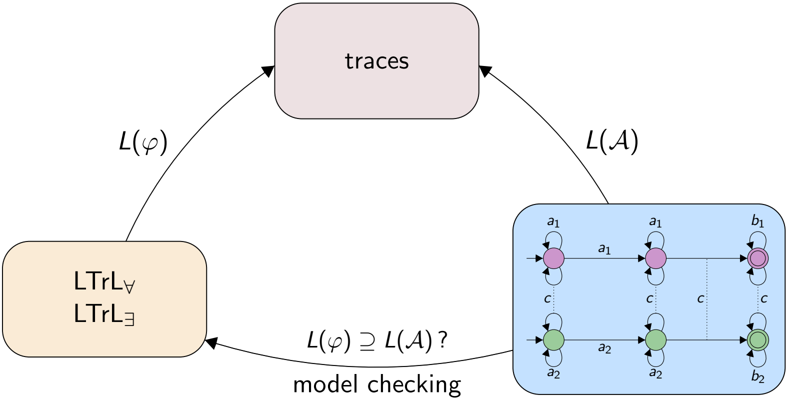
$\models \;\mathtt{tt}\,\mathsf{U}_\forall\,\langle b_1\rangle\langle b_2\rangle\mathtt{tt}$

$\models \;\varphi\,\mathsf{U}_\exists\,\psi$

Finite-State Shared-Memory Systems

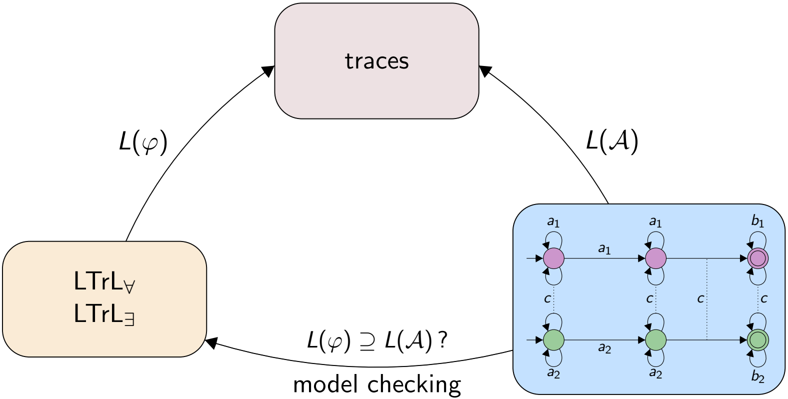


Finite-State Shared-Memory Systems



Theorem (Walukiewicz '98; Alur-McMillan-Peled '98)

- $\mathrm{LTrL}_\forall$ model checking is nonelementary.



Theorem (Walukiewicz '98; Alur-McMillan-Peled '98)

- $\mathrm{LTrL}_\forall$ model checking is nonelementary.
- $\mathrm{LTrL}_\exists$ model checking is undecidable.

Local Temporal Logic

Local Temporal Logic

$$\varphi \quad ::= \quad a \mid \mathsf{EX}\varphi \mid \mathsf{EX}_p\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \mid \varphi_1 \,\mathsf{U}_p\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$$

$$a \in \Sigma, p \in Proc$$

Local Temporal Logic

Local Temporal Logic

$$\varphi \quad ::= \quad a \mid \mathsf{EX}\varphi \mid \mathsf{EX}_p\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \mid \varphi_1 \,\mathsf{U}_p\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$$

$$a \in \Sigma, p \in Proc$$

Semantics (wrt. trace $t = (E, (\rightarrow_p)_{p \in Proc}, \lambda)$ and $e \in E$)

- $t, e \models \mathsf{EX}\varphi$ if there is $f \in E$ such that $e \lessdot f$ and $t, f \models \varphi$

Local Temporal Logic

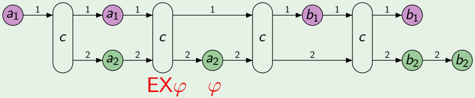
Local Temporal Logic

$$\varphi \quad ::= \quad a \mid \mathsf{EX}\varphi \mid \mathsf{EX}_p\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \mid \varphi_1 \,\mathsf{U}_p\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$$

$$a \in \Sigma, p \in Proc$$

Semantics (wrt. trace $t = (E, (\rightarrow_p)_{p\in Proc}, \lambda)$ and $e \in E$)

- $t, e \models \mathsf{EX}\varphi$ if there is $f \in E$ such that $e \lessdot f$ and $t, f \models \varphi$



Local Temporal Logic

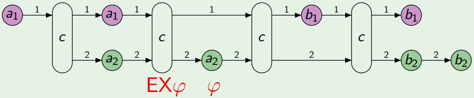
Local Temporal Logic

$$\varphi \quad ::= \quad a \mid \mathsf{EX}\varphi \mid \mathsf{EX}_p\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \mid \varphi_1 \,\mathsf{U}_p\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$$

$$a \in \Sigma, p \in Proc$$

Semantics (wrt. trace $t = (E, (\rightarrow_p)_{p \in Proc}, \lambda)$ and $e \in E$)

- $t, e \models \mathsf{EX}\varphi$ if there is $f \in E$ such that $e \lessdot f$ and $t, f \models \varphi$



- $t, e \models \mathsf{EX}_p\varphi$ if there is $f \in E$ such that $e \leq f$ and $e \rightarrow_p f$ and $t, f \models \varphi$

Local Temporal Logic

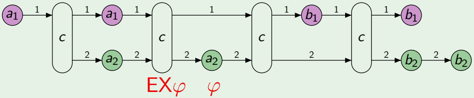
Local Temporal Logic

$$\varphi \quad ::= \quad a \mid \mathrm{EX}\varphi \mid \mathrm{EX}_p\varphi \mid \varphi_1 \,\mathrm{U}\, \varphi_2 \mid \varphi_1 \,\mathrm{U}_p\, \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$$

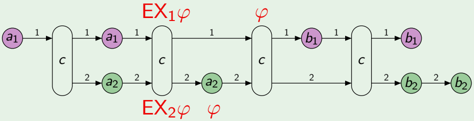
$$a \in \Sigma, p \in Proc$$

Semantics (wrt. trace $t = (E, (\rightarrow_p)_{p \in Proc}, \lambda)$ and $e \in E$)

- $t, e \models \mathrm{EX}\varphi$ if there is $f \in E$ such that $e \lessdot f$ and $t, f \models \varphi$



- $t, e \models \mathrm{EX}_p\varphi$ if there is $f \in E$ such that $e \leq f$ and $e \rightarrow_p f$ and $t, f \models \varphi$



Temporal Logic

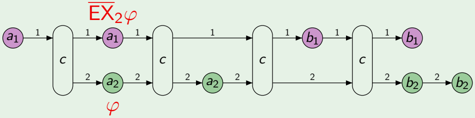
Semantics (wrt. trace $t = (E, (\rightarrow_p)_{p \in Proc}, \lambda)$ and $e \in E$)

- $t, e \models \overline{\mathsf{EX}}_p \varphi$ if there is $f \in E$ such that $\lambda(f) \in \Sigma_p$ and $t, f \models \varphi$ and f is the first p-event not below e wrt. $\leq$

Temporal Logic

Semantics (wrt. trace $t = (E, (\rightarrow_p)_{p \in Proc}, \lambda)$ and $e \in E$)

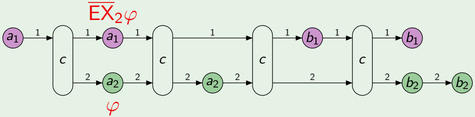
- $t, e \models \overline{\mathsf{EX}}_p \varphi$ if there is $f \in E$ such that $\lambda(f) \in \Sigma_p$ and $t, f \models \varphi$ and f is the first p-event not below e wrt. $\leq$



Temporal Logic

Semantics (wrt. trace $t = (E, (\to_p)_{p \in Proc}, \lambda)$ and $e \in E$)

- $t, e \models \overline{\mathsf{EX}}_p \varphi$ if there is $f \in E$ such that $\lambda(f) \in \Sigma_p$ and $t, f \models \varphi$ and f is the first p-event not below e wrt. $\leq$

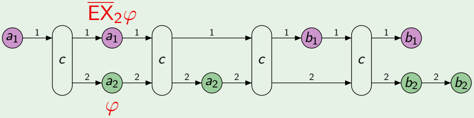


- $t, e \models \varphi \,\mathsf{U}\, \psi$ if there is $f \in E$ such that $t, f \models \psi$ and $t, e' \models \varphi$ for all $e' \in E$ with $e \leq e' < f$

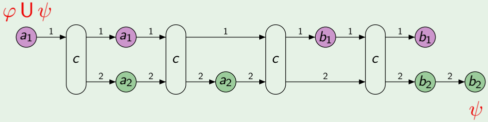
Temporal Logic

Semantics (wrt. trace $t = (E, (\rightarrow_p)_{p \in Proc}, \lambda)$ and $e \in E$)

- $t, e \models \overline{\mathsf{EX}}_p \varphi$ if there is $f \in E$ such that $\lambda(f) \in \Sigma_p$ and $t, f \models \varphi$ and f is the first p-event not below e wrt. $\leq$



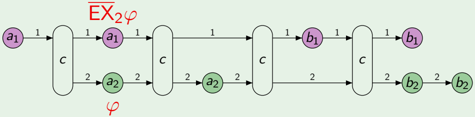
- $t, e \models \varphi \,\mathsf{U}\, \psi$ if there is $f \in E$ such that $t, f \models \psi$ and $t, e' \models \varphi$ for all $e' \in E$ with $e \leq e' < f$



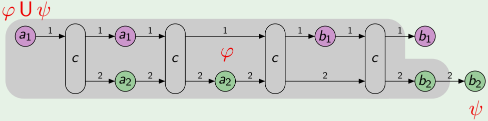
Temporal Logic

Semantics (wrt. trace $t = (E, (\to_p)_{p \in Proc}, \lambda)$ and $e \in E$)

- $t, e \models \overline{\mathsf{EX}}_p\varphi$ if there is $f \in E$ such that $\lambda(f) \in \Sigma_p$ and $t, f \models \varphi$ and f is the first p-event not below e wrt. $\leq$



- $t, e \models \varphi \,\mathsf{U}\, \psi$ if there is $f \in E$ such that $t, f \models \psi$ and $t, e' \models \varphi$ for all $e' \in E$ with $e \leq e' < f$



Temporal Logic

Observation (Gastin-Kuske '03)

All these modalities are MSO-definable!

Temporal Logic

Observation (Gastin-Kuske '03)

All these modalities are MSO-definable!

Semantics

(wrt. trace $t = (E, (\to_p)_{p \in Proc}, \lambda)$ and $e \in E$)

- $t, e \models \mathsf{EX}\varphi$ — if there is $f \in E$ such that $e \lessdot f$ and $t, f \models \varphi$
- $t, e \models \mathsf{EX}_p\varphi$ — if there is $f \in E$ such that $e \to_p f$ and $t, f \models \varphi$ and f is the first p-event not below e wrt. $\leq$
- $t, e \models \varphi \,\mathsf{U}\, \psi$ — if there is $f \in E$ such that $t, f \models \psi$ and $t, e' \models \varphi$ for all $e' \in E$ with $e \leq e' < f$

Temporal Logic

Observation (Gastin-Kuske '03)

All these modalities are MSO-definable!

Semantics (wrt. trace $t = (E, (\rightarrow_p)_{p \in Proc}, \lambda)$ and $e \in E$)

- $t, e \models \mathsf{EX}\varphi$ if there is $f \in E$ such that $e \lessdot f$ and $t, f \models \varphi$
- $t, e \models \mathsf{EX}_p\varphi$ if there is $f \in E$ such that $e \rightarrow_p f$ and $t, f \models \varphi$ and f is the first p-event not below e wrt. $\leq$
- $t, e \models \varphi \,\mathsf{U}\, \psi$ if there is $f \in E$ such that $t, f \models \psi$ and $t, e' \models \varphi$ for all $e' \in E$ with $e \leq e' < f$

Example

- $\mathrm{MSO}^{\mathsf{EX}}(x, Y) = \exists y\,(y \in Y \wedge x \lessdot y)$

Temporal Logic

Observation (Gastin-Kuske '03)

All these modalities are MSO-definable!

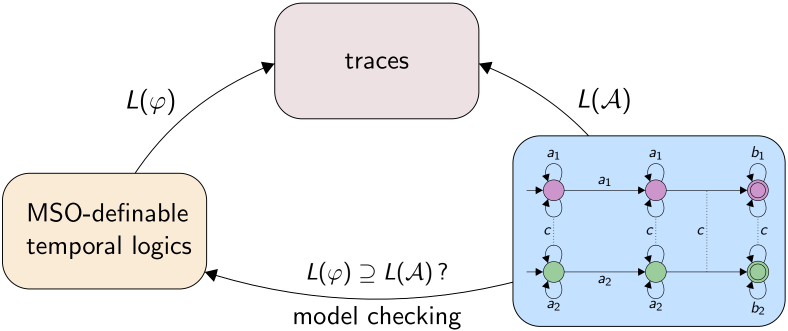
Semantics (wrt. trace $t = (E, (\rightarrow_p)_{p \in Proc}, \lambda)$ and $e \in E$)

- $t, e \models \mathsf{EX}\varphi$ if there is $f \in E$ such that $e \lessdot f$ and $t, f \models \varphi$
- $t, e \models \mathsf{EX}_p\varphi$ if there is $f \in E$ such that $e \rightarrow_p f$ and $t, f \models \varphi$ and f is the first p-event not below e wrt. $\leq$
- $t, e \models \varphi \,\mathsf{U}\, \psi$ if there is $f \in E$ such that $t, f \models \psi$ and $t, e' \models \varphi$ for all $e' \in E$ with $e \leq e' < f$

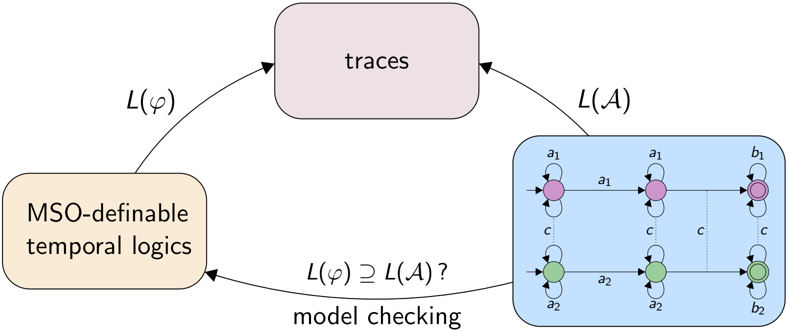
Example

- $\mathrm{MSO}^{\mathsf{EX}}(x, Y) = \exists y\,(y \in Y \wedge x \lessdot y)$
- $\mathrm{MSO}^{\mathsf{U}}(x, X, Y) = \exists y\,(y \in Y \wedge x \leq y \wedge \forall x'(x \leq x' < y \rightarrow x' \in X))$

Finite-State Shared-Memory Systems



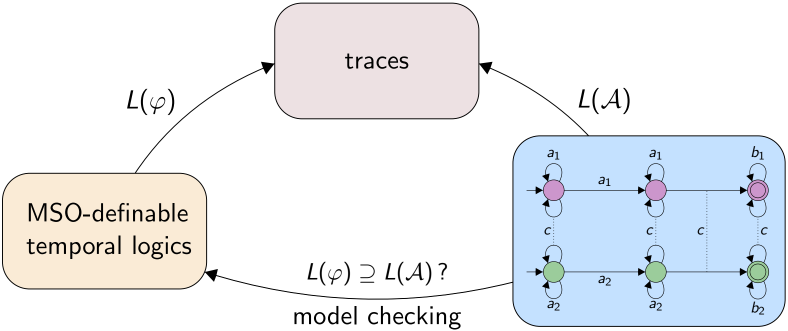
Finite-State Shared-Memory Systems



Theorem (Gastin-Kuske '03)

Model checking for any MSO-definable temporal logic is in PSPACE.

Finite-State Shared-Memory Systems



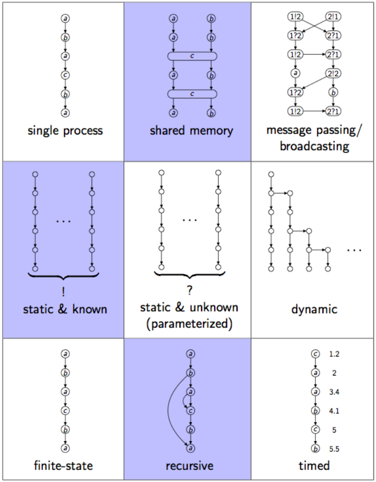
Theorem (Gastin-Kuske '03)

Model checking for any MSO-definable temporal logic is in PSPACE.

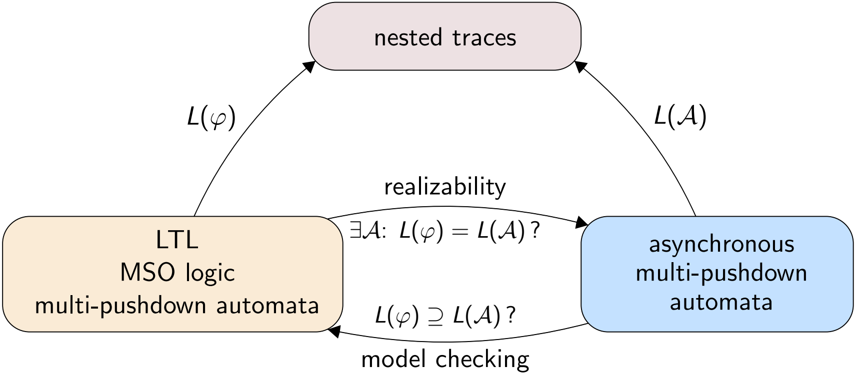
Proof.

Precompile MSO modalities into finite automata. Inductively build finite automaton equivalent to the input formula. □

5. Recursive Shared-Memory Systems



Recursive Shared-Memory Systems



Asynchronous Multi-Pushdown Automata

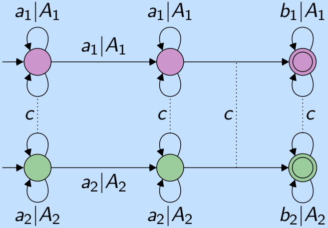
$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{\mathsf{call}} = \{a_1, a_2\}$ $\Sigma_{\mathsf{ret}} = \{b_1, b_2\}$

Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA

Stack 1 Stack 2

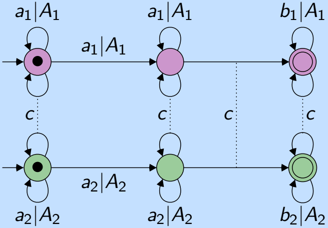


Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA

Stack 1 Stack 2

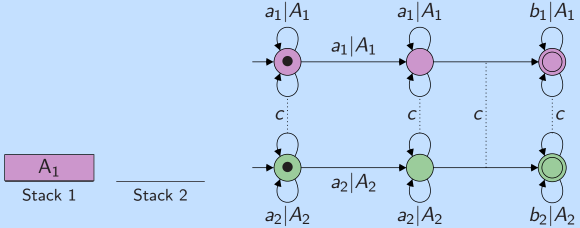


Nested Trace

Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA

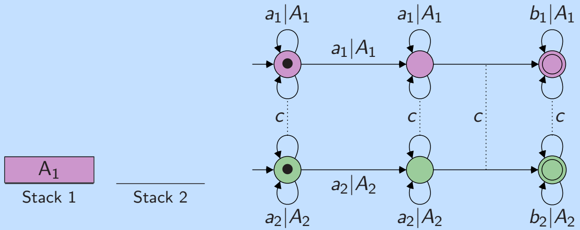


Nested Trace

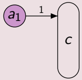
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



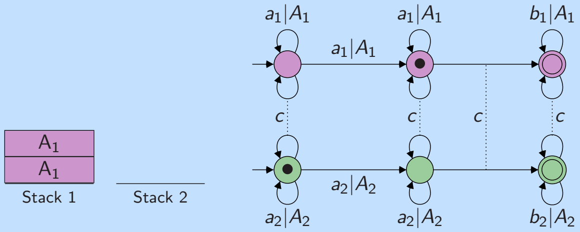
Nested Trace



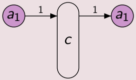
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



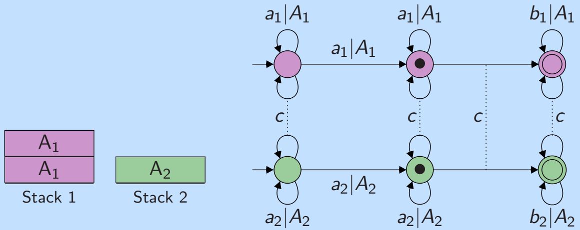
Nested Trace



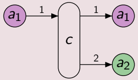
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



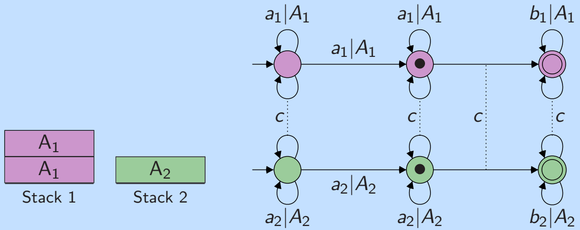
Nested Trace



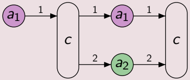
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{\text{call}} = \{a_1, a_2\}$ $\Sigma_{\text{ret}} = \{b_1, b_2\}$

Asynchronous MPA



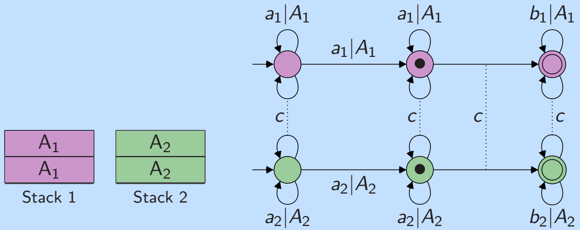
Nested Trace



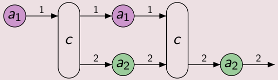
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



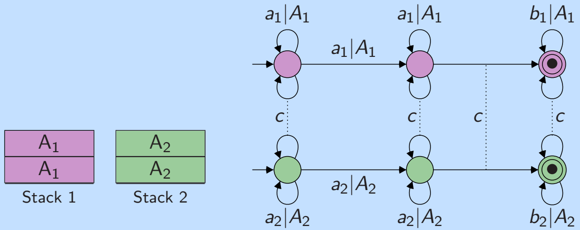
Nested Trace



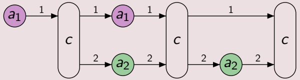
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



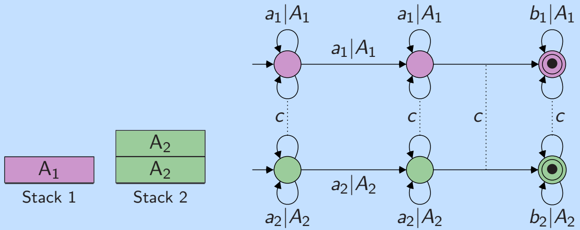
Nested Trace



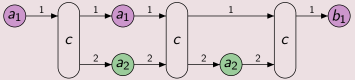
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



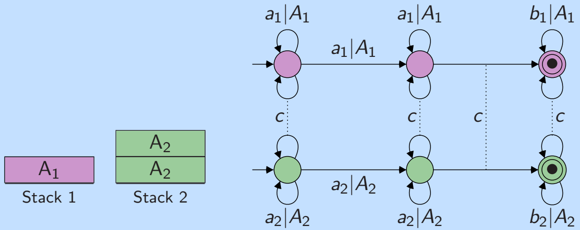
Nested Trace



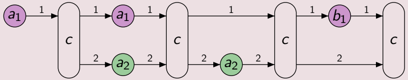
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



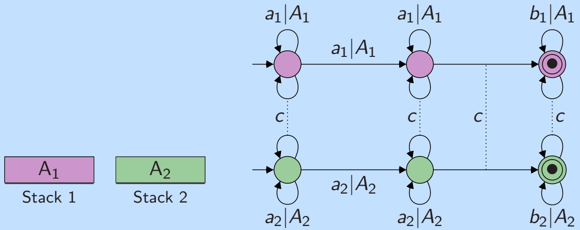
Nested Trace



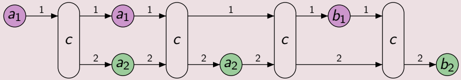
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



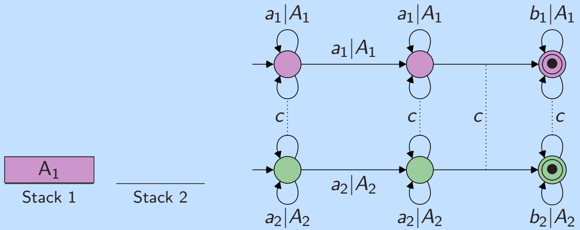
Nested Trace



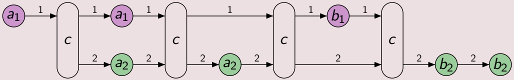
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



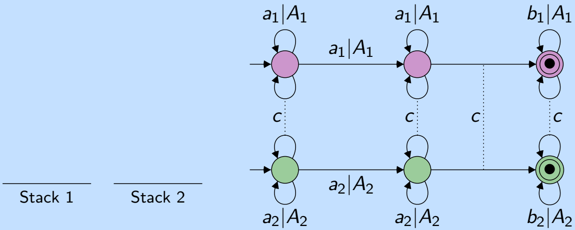
Nested Trace



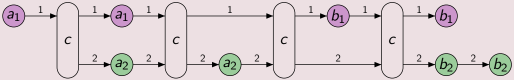
Asynchronous Multi-Pushdown Automata

$Proc = \{1,2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



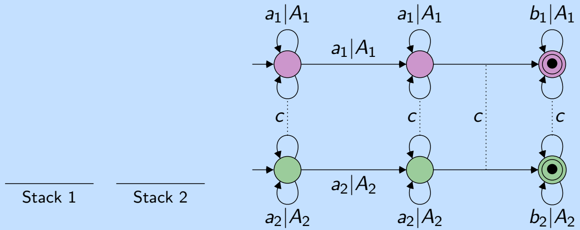
Nested Trace



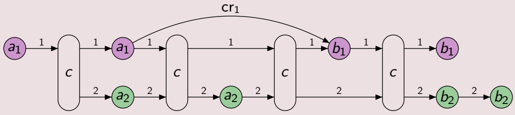
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



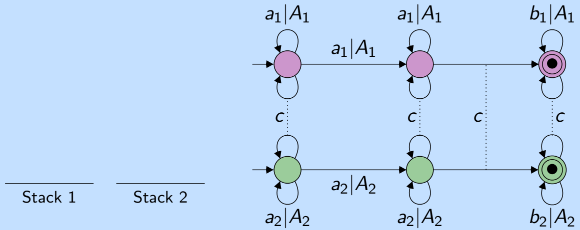
Nested Trace



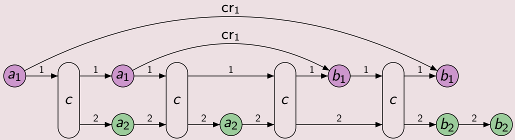
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\quad \Sigma_1 = \{a_1, b_1, c\}$ $\quad \Sigma_2 = \{a_2, b_2, c\}$ $\quad \Sigma_{call} = \{a_1, a_2\}$ $\quad \Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



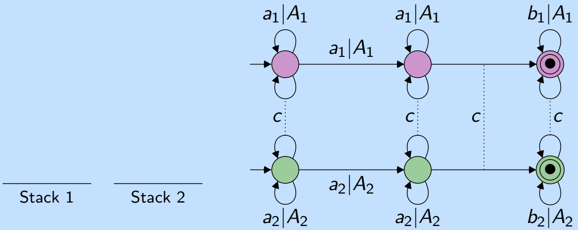
Nested Trace



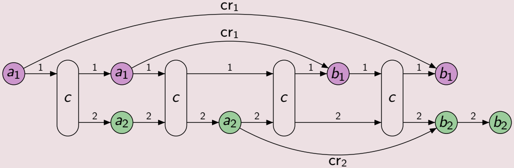
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\quad \Sigma_1 = \{a_1, b_1, c\}$ $\quad \Sigma_2 = \{a_2, b_2, c\}$ $\quad \Sigma_{\text{call}} = \{a_1, a_2\}$ $\quad \Sigma_{\text{ret}} = \{b_1, b_2\}$

Asynchronous MPA



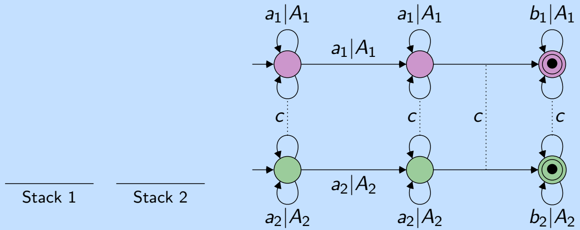
Nested Trace



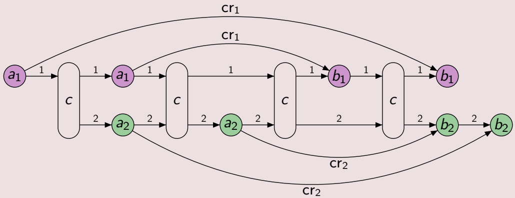
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

Asynchronous MPA



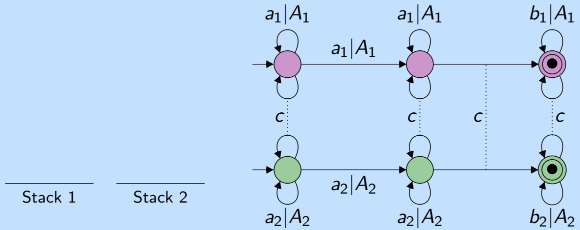
Nested Trace



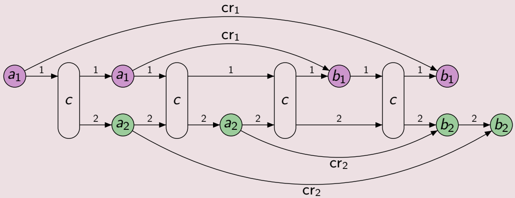
Asynchronous Multi-Pushdown Automata

$Proc = \{1, 2\}$ $\Sigma_1 = \{a_1, b_1, c\}$ $\Sigma_2 = \{a_2, b_2, c\}$ $\Sigma_{call} = \{a_1, a_2\}$ $\Sigma_{ret} = \{b_1, b_2\}$

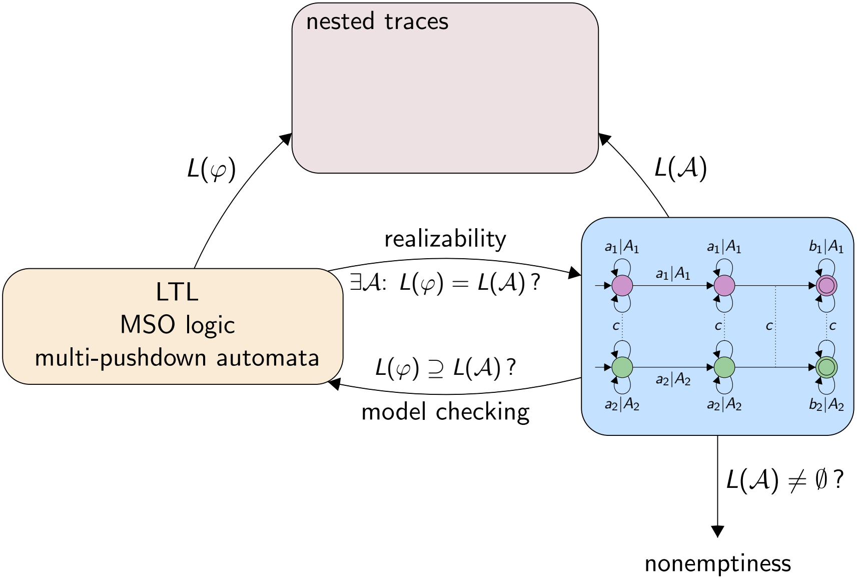
Asynchronous MPA



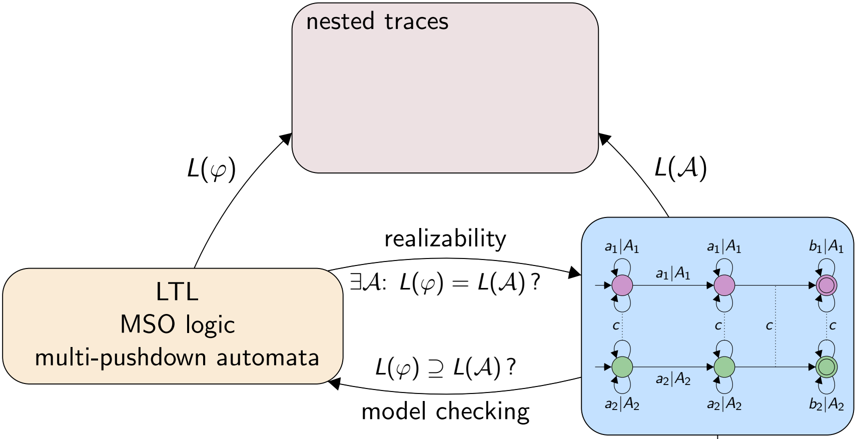
Nested Trace $t = (E, \rightarrow_1, \rightarrow_2, \curvearrowright_1, \curvearrowright_2, \lambda)$



Recursive Shared-Memory Systems



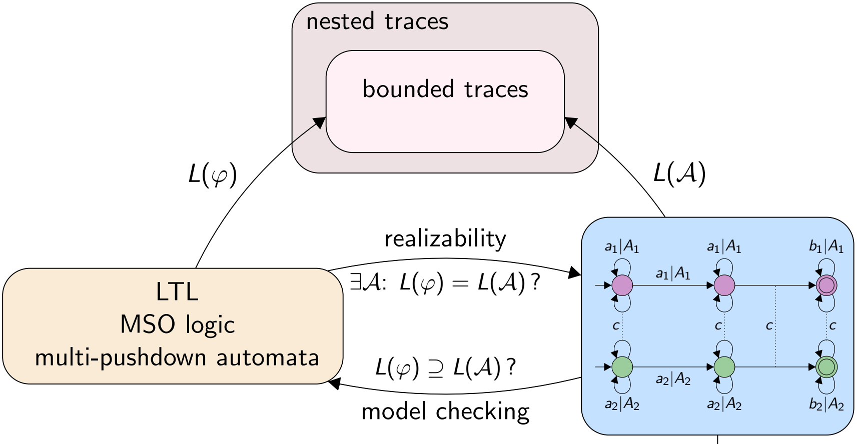
Recursive Shared-Memory Systems



Theorem

Emptiness for (asynchronous) MPA is undecidable.

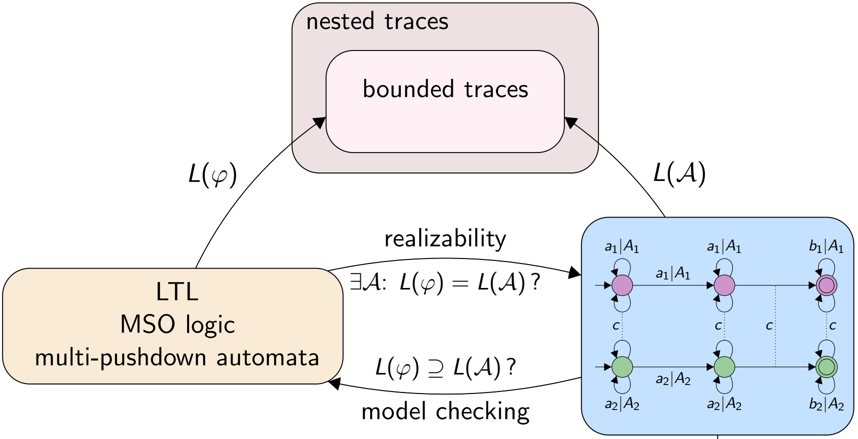
Recursive Shared-Memory Systems



Theorem

Emptiness for (asynchronous) MPA is undecidable.

Recursive Shared-Memory Systems

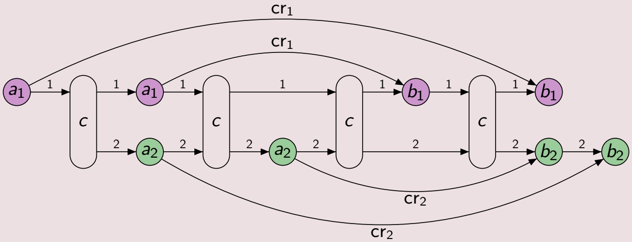


Theorem

Bounded nonemptiness, satisfiability, model checking, and realizability are decidable.

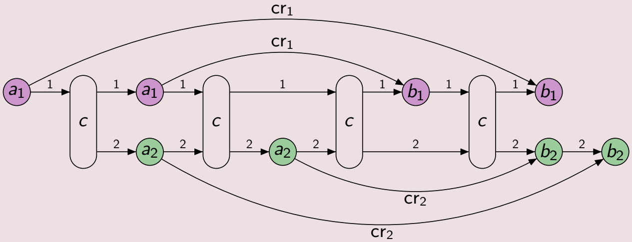
Nested Traces and Their Linearizations

Nested Trace $t = (E, \rightarrow_1, \rightarrow_2, \curvearrowright_1, \curvearrowright_2, \lambda)$

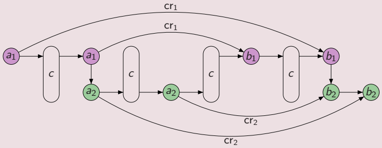


Nested Traces and Their Linearizations

Nested Trace $\quad t = (E, \rightarrow_1, \rightarrow_2, \curvearrowright_1, \curvearrowright_2, \lambda)$

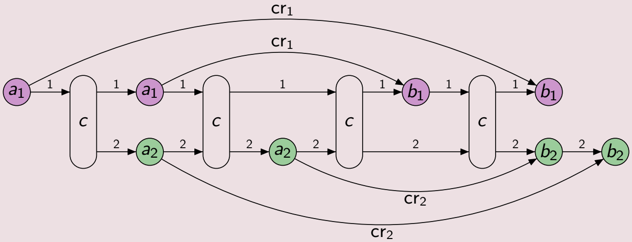


Linearizations $\quad w \in Lin(t) \quad \rightsquigarrow \quad trace(w) = t$

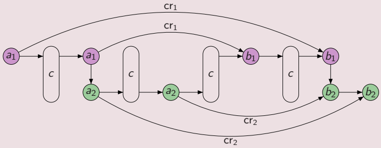


Nested Traces and Their Linearizations

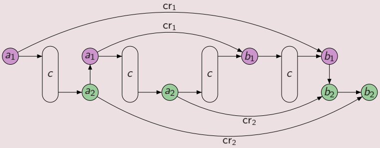
Nested Trace $\quad t = (E, \rightarrow_1, \rightarrow_2, \curvearrowright_1, \curvearrowright_2, \lambda)$



Linearizations $\quad w \in Lin(t) \quad \rightsquigarrow \quad trace(w) = t$



$\sim_0$



Bounded Nested Words

Definition

- In a context, only one process modifies its stack.

Bounded Nested Words

Definition

- In a <u>context</u>, only one process modifies its stack.
- In a <u>phase</u>, only one process pops from its stack.

Bounded Nested Words

Definition

- In a context, only one process modifies its stack.
- In a phase, only one process pops from its stack.

A nested word is

- k-scope bounded if each call-return lies within k contexts.

Bounded Nested Words

Definition

- In a context, only one process modifies its stack.
- In a phase, only one process pops from its stack.

A nested word is

- *k*-scope bounded if each call-return lies within k contexts.
- ordered if a pop is performed only on the first nonempty stack.

Bounded Nested Words

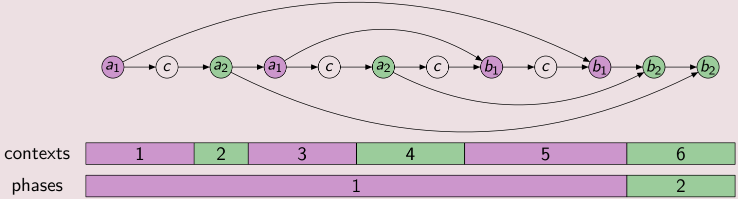
Definition

- In a context, only one process modifies its stack.
- In a phase, only one process pops from its stack.

A nested word is

- *k*-scope bounded if each call-return lies within k contexts.
- ordered if a pop is performed only on the first nonempty stack.

Bounded Nested Words



6-context bounded / 2-phase bounded / 5-scope bounded / ordered

Bounded Nested Traces

Definition

A nested trace if k-context bounded / k-phase bounded / k-scope bounded / ordered if at least one linearization is so.

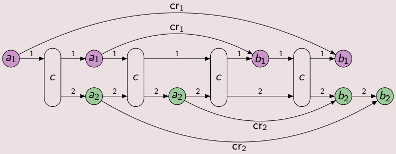
Bounded Nested Traces

Definition

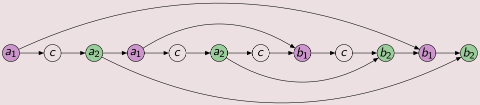
A nested trace if k-context bounded / k-phase bounded / k-scope bounded / ordered if at least one linearization is so.

Bounded Nested Traces

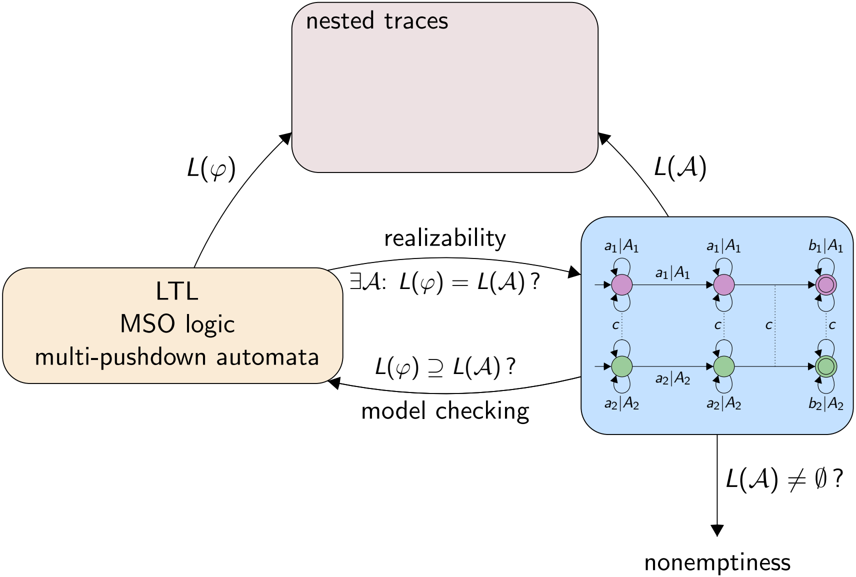
2-phase bounded



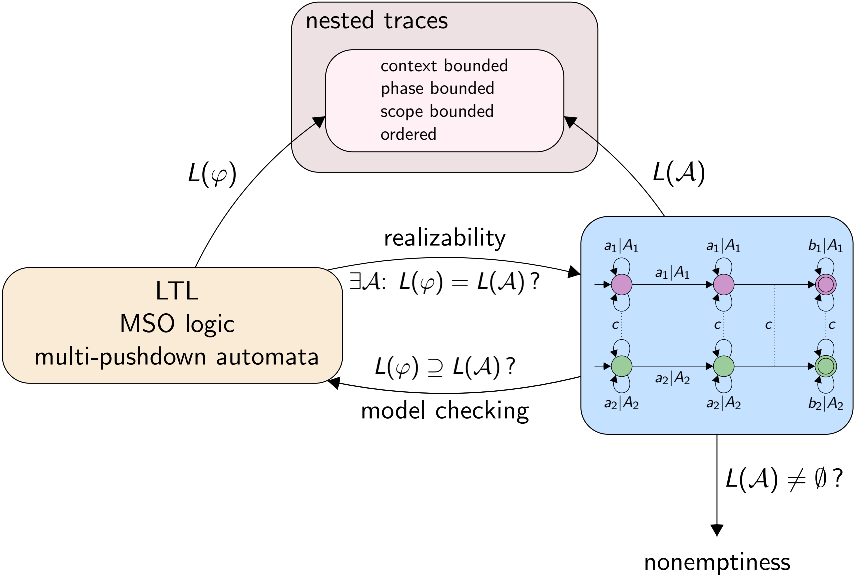
not 2-phase bounded



Recursive Shared-Memory Systems



Recursive Shared-Memory Systems



Recursive Shared-Memory Systems

nested traces

context bounded

Theorem

Bounded nonemptiness for sequential MPA is

context	NP-complete	[Qadeer-Rehof '05]
scope	PSPACE-complete	[La Torre-Napoli '11]
phase	2EXPTIME-complete	[La Torre-Madhusudan-Parlato '07]
ordered	2EXPTIME-complete	[Atig-B.-Habermehl '08]

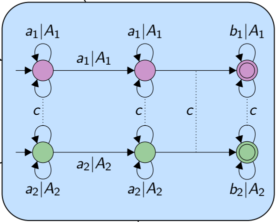
$L(\mathcal{A})$

$\exists\mathcal{A}$: $L(\varphi) = L(\mathcal{A})$?

LTL
MSO logic
multi-pushdown automata

$L(\varphi) \supseteq L(\mathcal{A})$?

model checking



$L(\mathcal{A}) \neq \emptyset$?

nonemptiness

Recursive Shared-Memory Systems

nested traces

context bounded

Theorem

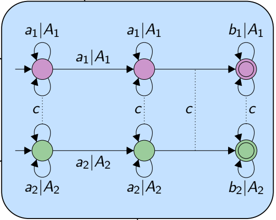
Bounded nonemptiness for sequential MPA is

- context NP-complete [Qadeer-Rehof '05]
- scope PSPACE-complete [La Torre-Napoli '11]
- phase 2EXPTIME-complete [La Torre-Madhusudan-Parlato '07]
- ordered 2EXPTIME-complete [Atig-B.-Habermehl '08]

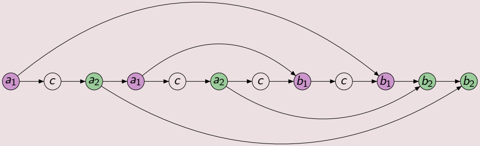
LTL

$\exists\mathcal{A}$: $L(\varphi) = L(\mathcal{A})$?

$L(\mathcal{A})$



Proof for phases: binary-tree encoding



$L(\mathcal{A}) \neq \emptyset$?

nonemptiness

Recursive Shared-Memory Systems

nested traces

context bounded

$L(\mathcal{A})$

Theorem

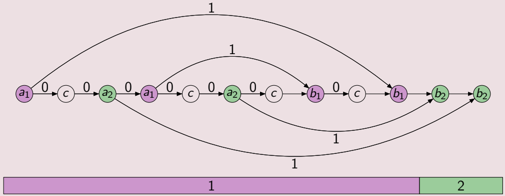
Bounded nonemptiness for sequential MPA is

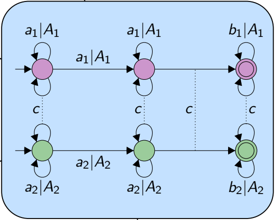
- context NP-complete [Qadeer-Rehof '05]
- scope PSPACE-complete [La Torre-Napoli '11]
- phase 2EXPTIME-complete [La Torre-Madhusudan-Parlato '07]
- ordered 2EXPTIME-complete [Atig-B.-Habermehl '08]

LTL

$\exists\mathcal{A}$: $L(\varphi) = L(\mathcal{A})$?

Proof for phases: binary-tree encoding





$L(\mathcal{A}) \neq \emptyset$?

nonemptiness

Recursive Shared-Memory Systems

nested traces

context bounded

Theorem

Bounded nonemptiness for sequential MPA is

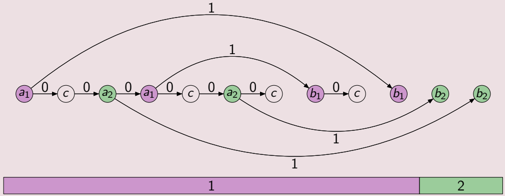
- context NP-complete [Qadeer-Rehof '05]
- scope PSPACE-complete [La Torre-Napoli '11]
- phase 2EXPTIME-complete [La Torre-Madhusudan-Parlato '07]
- ordered 2EXPTIME-complete [Atig-B.-Habermehl '08]

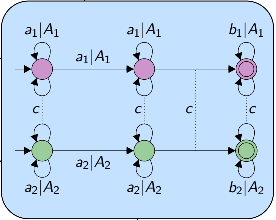
LTL

$\exists\mathcal{A}\colon L(\varphi) = L(\mathcal{A})$?

$L(\mathcal{A})$

Proof for phases: binary-tree encoding





$L(\mathcal{A}) \neq \emptyset$?

nonemptiness

Recursive Shared-Memory Systems

nested traces

context bounded

Theorem

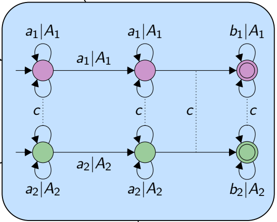
Bounded nonemptiness for sequential MPA is

- context NP-complete [Qadeer-Rehof '05]
- scope PSPACE-complete [La Torre-Napoli '11]
- phase 2EXPTIME-complete [La Torre-Madhusudan-Parlato '07]
- ordered 2EXPTIME-complete [Atig-B.-Habermehl '08]

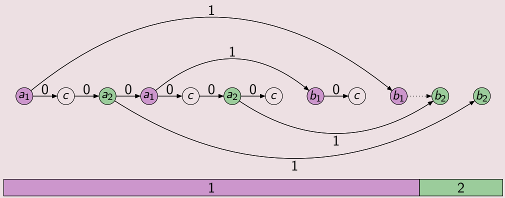
LTL

$\exists \mathcal{A}$: $L(\varphi) = L(\mathcal{A})$?

$L(\mathcal{A})$



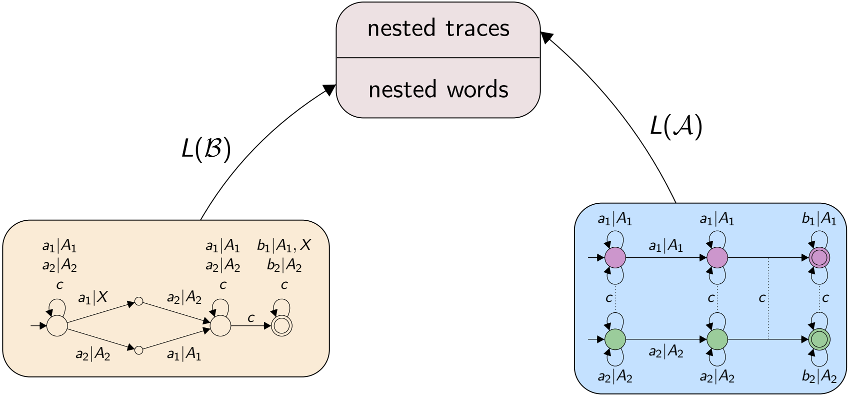
Proof for phases: binary-tree encoding



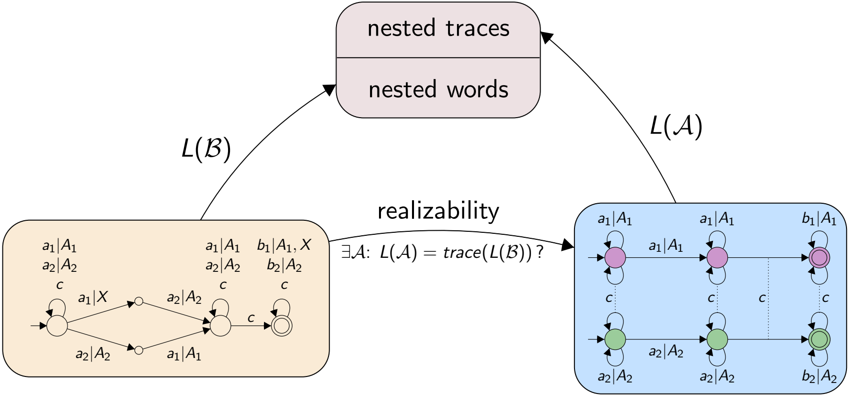
$L(\mathcal{A}) \neq \emptyset$?

nonemptiness

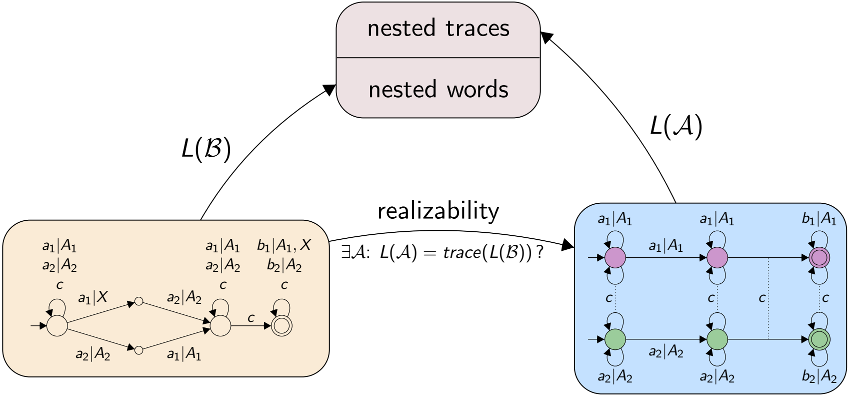
Recursive Shared-Memory Systems



Recursive Shared-Memory Systems



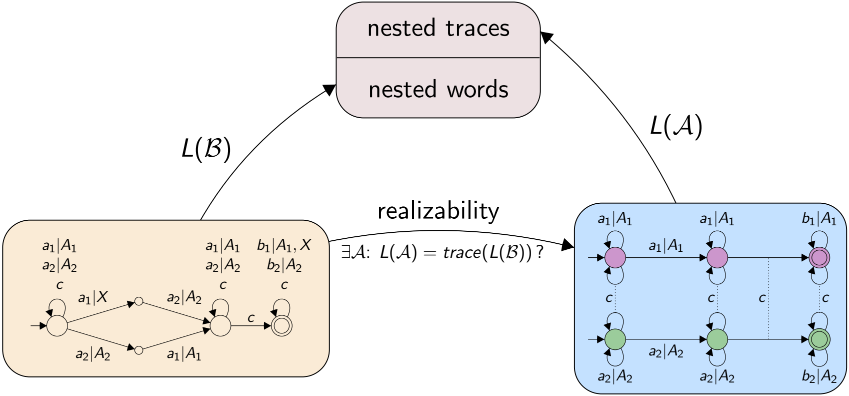
Recursive Shared-Memory Systems



Theorem (B.-Grindei-Habermehl '09)

Let L be a $\sim$-closed language recognized by some sequential MPA. There is an asynchronous MPA $\mathcal{A}$ such that $L(\mathcal{A}) = trace(L)$.

Recursive Shared-Memory Systems



Theorem

It is undecidable if the language of a sequential MPA is $\sim$-closed.

Representations

Let $\theta \in \{k\text{-context}, k\text{-scope}, k\text{-phase}, \text{ordered} \mid k \in \mathbb{N}\}$.

Definition

A set L of θ-nested words is a θ-representation if, for all θ-nested words w, w' with $w \sim_0 w'$, we have $w \in L$ iff $w' \in L$.

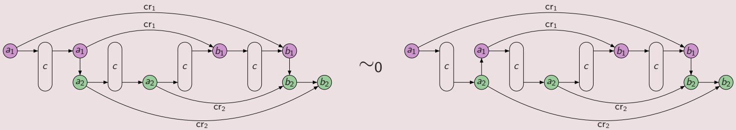
Representations

Let $\theta \in \{k\text{-context}, k\text{-scope}, k\text{-phase}, \text{ordered} \mid k \in \mathbb{N}\}$.

Definition

A set L of θ-nested words is a $\underline{\theta\text{-representation}}$ if, for all θ-nested words w, w' with $w \sim_0 w'$, we have $w \in L$ iff $w' \in L$.

2-phase representation



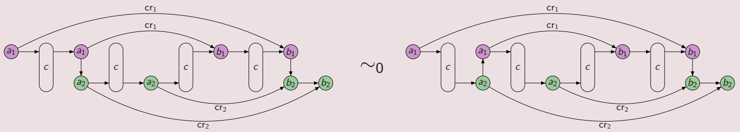
Representations

Let $\theta \in \{k\text{-context}, k\text{-scope}, k\text{-phase}, \text{ordered} \mid k \in \mathbb{N}\}$.

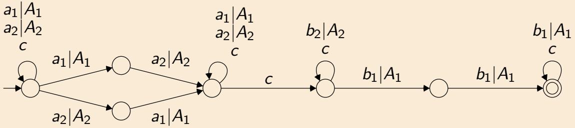
Definition

A set L of θ-nested words is a $\underline{\theta\text{-representation}}$ if, for all θ-nested words w, w' with $w \sim_0 w'$, we have $w \in L$ iff $w' \in L$.

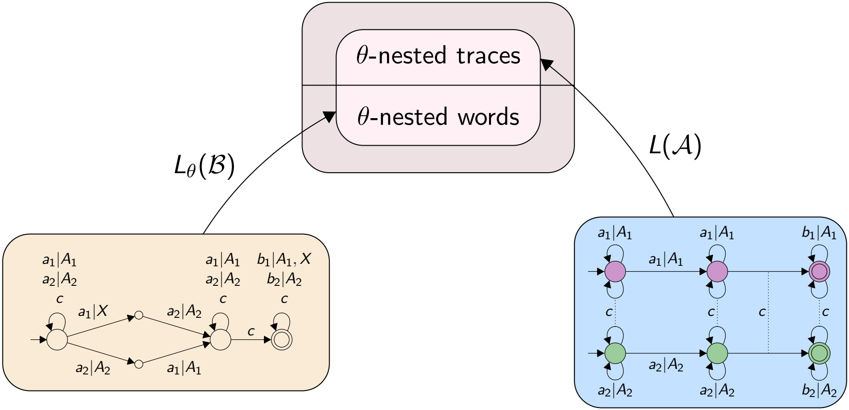
2-phase representation



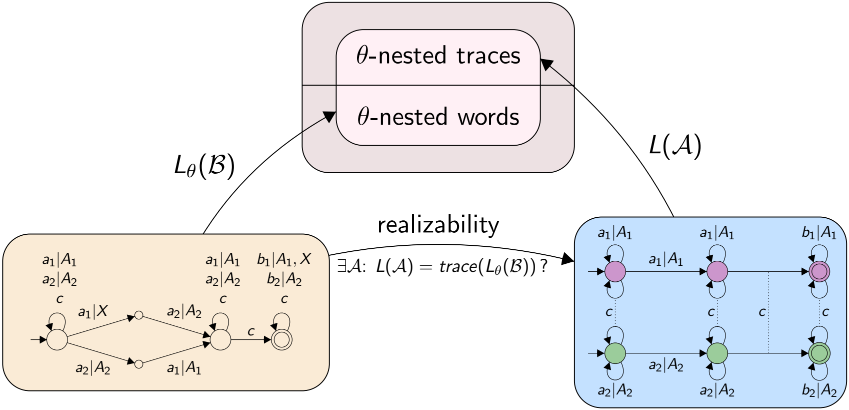
2-phase representation



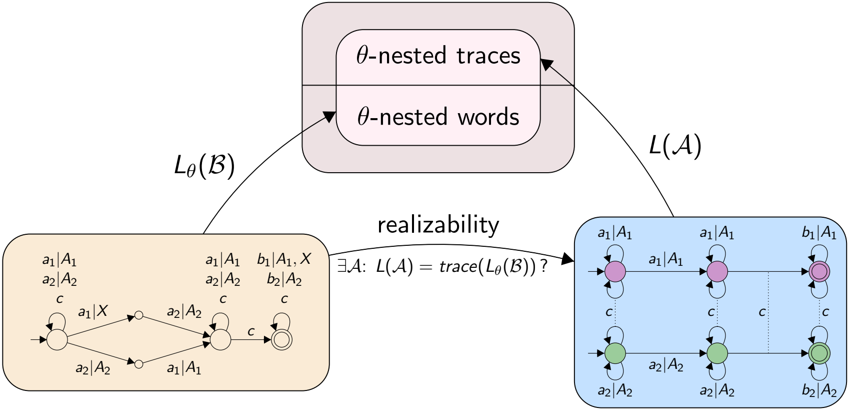
Recursive Shared-Memory Systems



Recursive Shared-Memory Systems



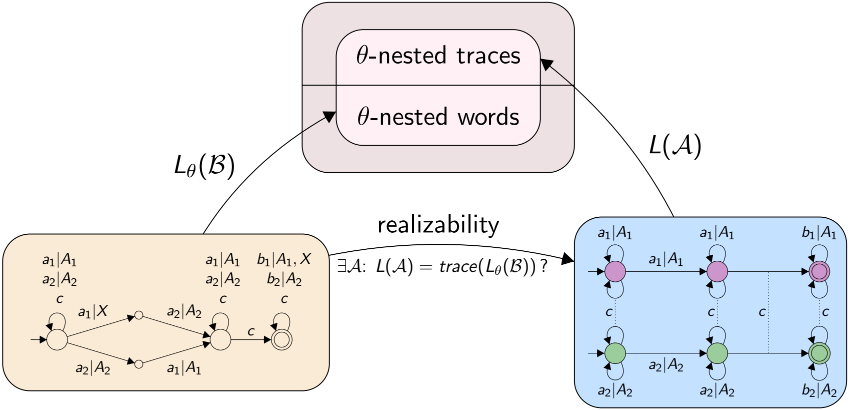
Recursive Shared-Memory Systems



Theorem (B.-Grindei-Habermehl '09)

Let $\mathcal{B}$ be some sequential MPA such that $L_\theta(\mathcal{B})$ is a θ-representation. There is an asynchronous MPA $\mathcal{A}$ such that $L(\mathcal{A}) = trace(L_\theta(\mathcal{B}))$.

Recursive Shared-Memory Systems



Theorem

For a sequential MPA $\mathcal{B}$ it is decidable if $L_\theta(\mathcal{B})$ is a θ-representation (in elementary time).

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

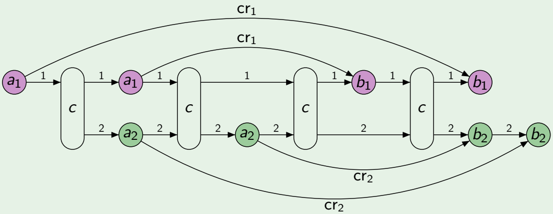
- $x \rightarrow_p y$ — x and y are successive events on process $p \in Proc$
- $x \curvearrowright_p y$ — x and y form a call-return pair of process $p \in Proc$
- $a(x)$ — event x is labeled with $a \in \Sigma$

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

- $x \rightarrow_p y$ — x and y are successive events on process $p \in Proc$
- $x \curvearrowright_p y$ — x and y form a call-return pair of process $p \in Proc$
- $a(x)$ — event x is labeled with $a \in \Sigma$

Example



$$\models \; \exists x \exists y \exists z \, (x \curvearrowright_1 y \;\wedge\; a_2(z) \;\wedge\; x \leq z \leq y)$$

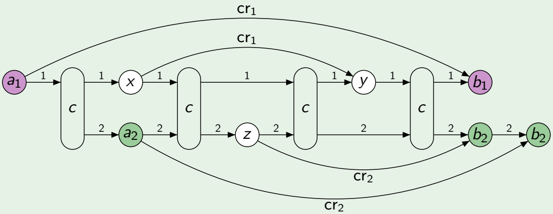
where $\leq \, = (\rightarrow_1 \cup \rightarrow_2)^*$

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

- $x \to_p y$ — x and y are successive events on process $p \in Proc$
- $x \curvearrowright_p y$ — x and y form a call-return pair of process $p \in Proc$
- $a(x)$ — event x is labeled with $a \in \Sigma$

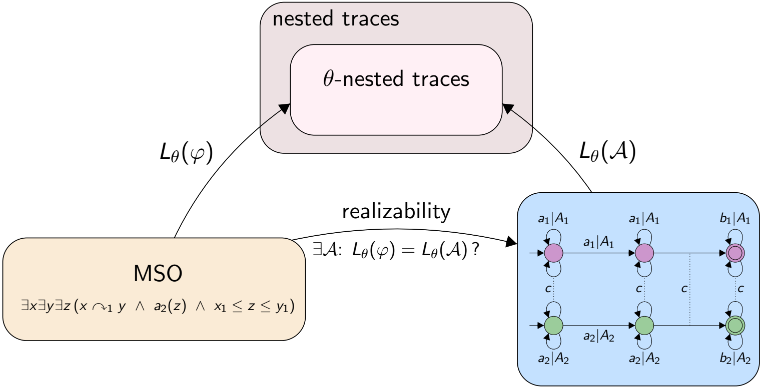
Example



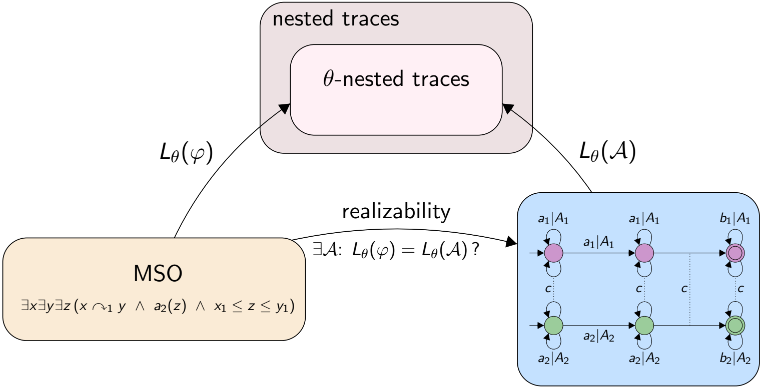
$$\models \; \exists x \exists y \exists z \, (x \curvearrowright_1 y \;\wedge\; a_2(z) \;\wedge\; x \leq z \leq y)$$

where $\leq \, = (\to_1 \cup \to_2)^*$

Recursive Shared-Memory Systems



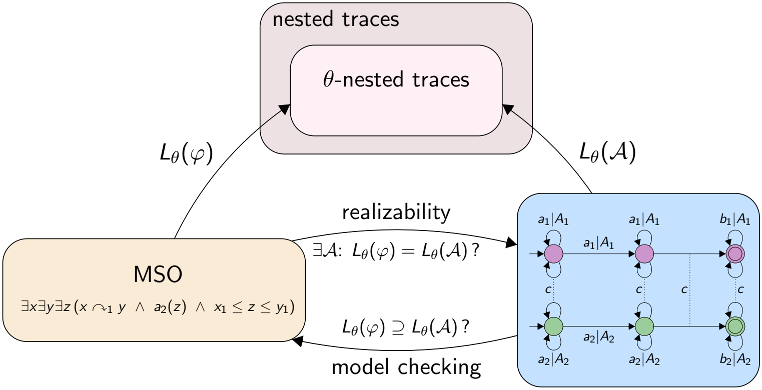
Recursive Shared-Memory Systems



Theorem (La Torre-Madhusudan-Parlato '07-'13)

MSO logic and asynchronous MPA are expressively equivalent wrt. θ-nested traces.

Recursive Shared-Memory Systems



Theorem (La Torre-Madhusudan-Parlato '07-'13)

MSO logic and asynchronous MPA are expressively equivalent wrt. θ-nested traces.

⇨ MSO model checking is decidable.

Local Temporal Logic

Observation

There are lots of (local) temporal logics for nested words/traces!

Local Temporal Logic

Observation

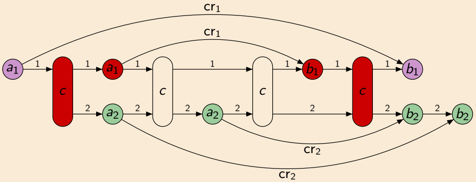
There are lots of (local) temporal logics for nested words/traces!
⇨ Look at MSO-definable ones.

Local Temporal Logic

Observation

There are lots of (local) temporal logics for nested words/traces!
⇨ Look at MSO-definable ones.

Abstract Until $\varphi \, \mathsf{U}^{\mathsf{a}}_{p} \, \psi$

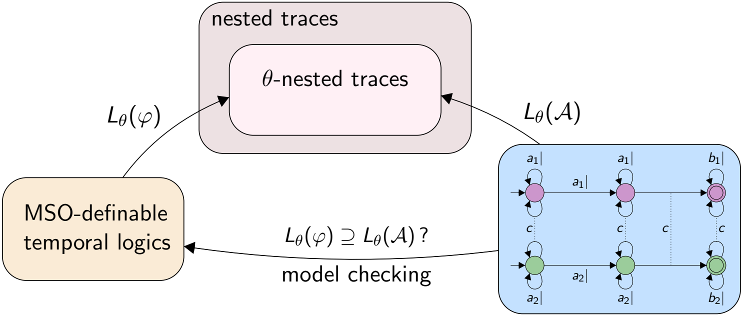


$$\mathrm{MSO}^{\mathsf{U}^{\mathsf{a}}_{p}}(x, X_1, X_2) =$$

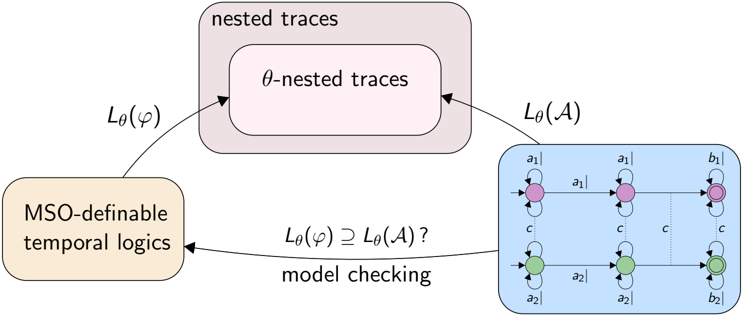
$$\exists Y \exists x' \Big(x' \in X_2 \wedge Y \subseteq X_1 \wedge \forall z (z \in Y \vee z = x') \to \big(z = x \vee \exists y \, (y \in Y \wedge \varphi_p(y, z))\big)\Big)$$

where $\varphi_p(y, z) = y \curvearrowright_p z \vee (\neg \exists z' \, y \curvearrowright_p z' \wedge \neg \exists y' \, (y' \curvearrowright_p z \wedge y \to_p z))$.

Model Checking (θ = "k-phase bounded")



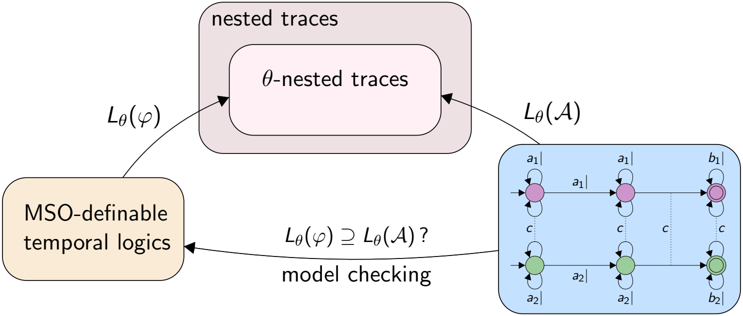
Model Checking (θ = "k-phase bounded")



Theorem (B.-Cyriac-Gastin-Zeitoun '11)

Model checking for any MSO-definable temporal logic is in EXPTIME when k is fixed.

Model Checking (θ = "k-phase bounded")



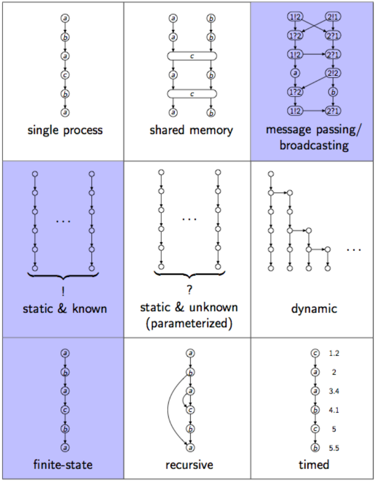
Theorem (B.-Cyriac-Gastin-Zeitoun '11)

Model checking for any MSO-definable temporal logic is in EXPTIME when k is fixed.

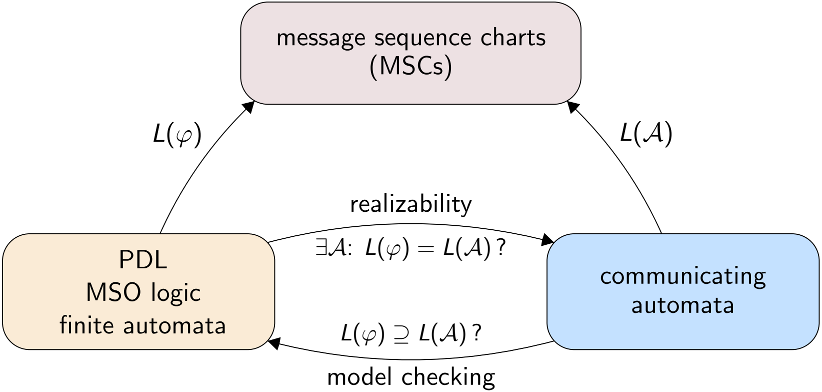
Theorem (B.-Kuske-Mennicke '13)

Model checking for any MSO-definable temporal logic is elementary when k is part of the input.

6. Message-Passing Systems



Message-Passing Systems



Communicating Automata and MSCs

$Proc = \{1, 2\}$

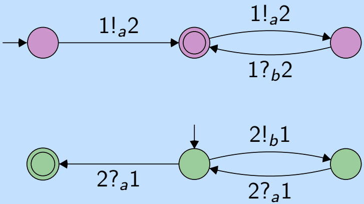
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\qquad \Sigma_1 = \{1!2\,,\, 1?2\}$ $\qquad \Sigma_2 = \{2!1\,,\, 2?1\}$

Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\, 1?2\}$ $\Sigma_2 = \{2!1\,,\, 2?1\}$

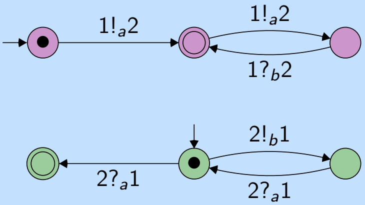
Communicating Automaton



Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\,1?2\}$ $\Sigma_2 = \{2!1\,,\,2?1\}$

Communicating Automaton

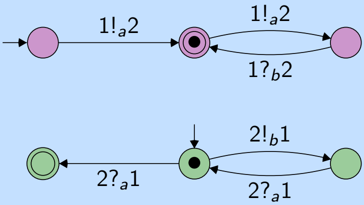


Message Sequence Chart (MSC)

Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\, ,\, 1?2\}$ $\Sigma_2 = \{2!1\, ,\, 2?1\}$

Communicating Automaton

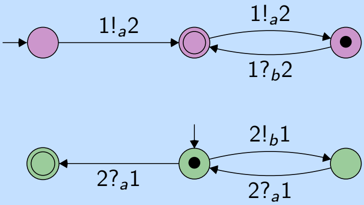


Message Sequence Chart (MSC)

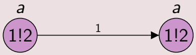
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\, ,\, 1?2\}$ $\Sigma_2 = \{2!1\, ,\, 2?1\}$

Communicating Automaton



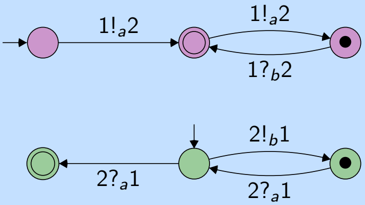
Message Sequence Chart (MSC)



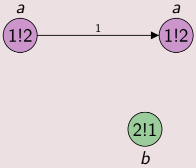
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\ 1?2\}$ $\Sigma_2 = \{2!1\,,\ 2?1\}$

Communicating Automaton



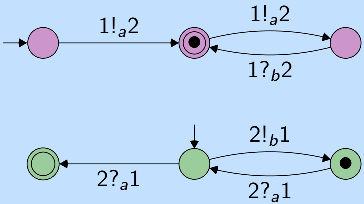
Message Sequence Chart (MSC)



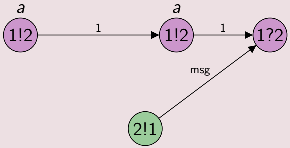
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\ 1?2\}$ $\Sigma_2 = \{2!1\,,\ 2?1\}$

Communicating Automaton



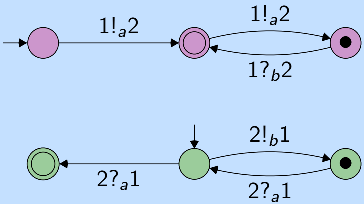
Message Sequence Chart (MSC)



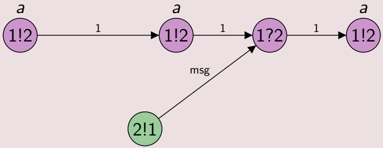
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\ 1?2\}$ $\Sigma_2 = \{2!1\,,\ 2?1\}$

Communicating Automaton



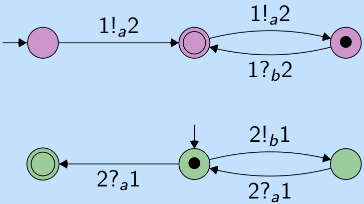
Message Sequence Chart (MSC)



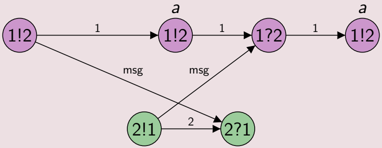
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\, ,\, 1?2\}$ $\Sigma_2 = \{2!1\, ,\, 2?1\}$

Communicating Automaton



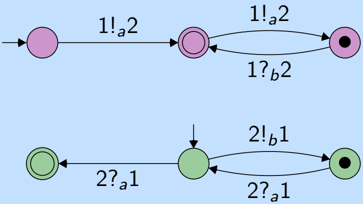
Message Sequence Chart (MSC)



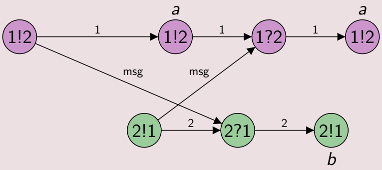
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\ 1?2\}$ $\Sigma_2 = \{2!1\,,\ 2?1\}$

Communicating Automaton



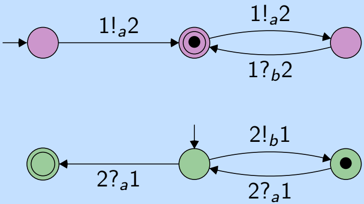
Message Sequence Chart (MSC)



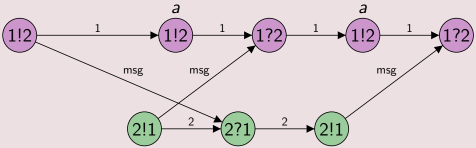
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\,1?2\}$ $\Sigma_2 = \{2!1\,,\,2?1\}$

Communicating Automaton



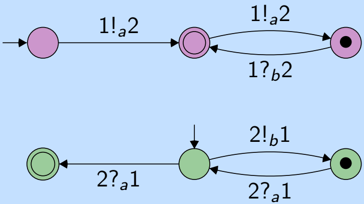
Message Sequence Chart (MSC)



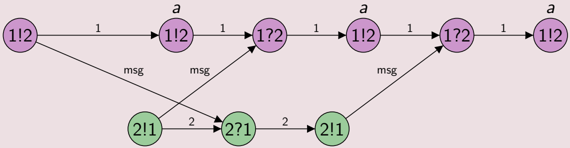
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\, 1?2\}$ $\Sigma_2 = \{2!1\,,\, 2?1\}$

Communicating Automaton



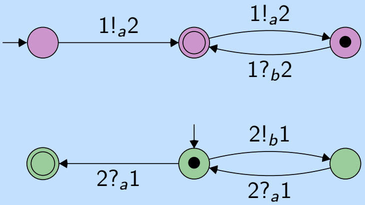
Message Sequence Chart (MSC)



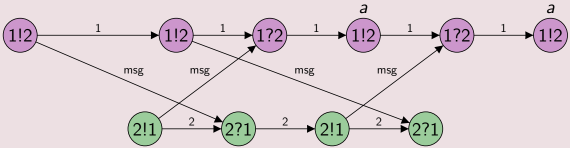
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\, 1?2\}$ $\Sigma_2 = \{2!1\,,\, 2?1\}$

Communicating Automaton



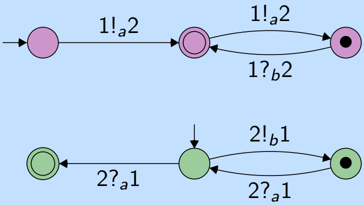
Message Sequence Chart (MSC)



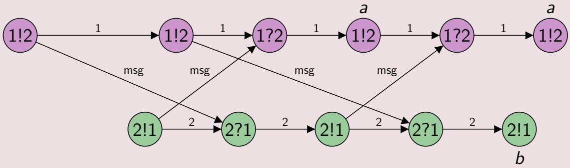
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\, 1?2\}$ $\Sigma_2 = \{2!1\,,\, 2?1\}$

Communicating Automaton



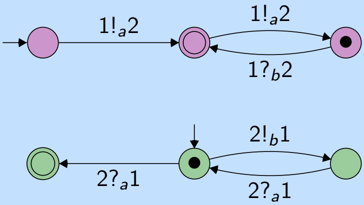
Message Sequence Chart (MSC)



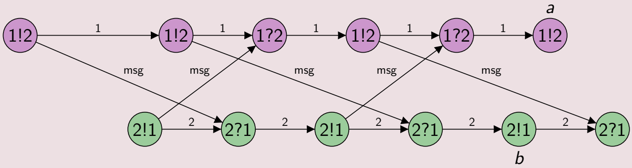
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\, 1?2\}$ $\Sigma_2 = \{2!1\,,\, 2?1\}$

Communicating Automaton



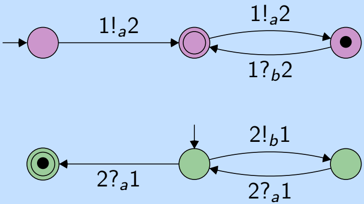
Message Sequence Chart (MSC)



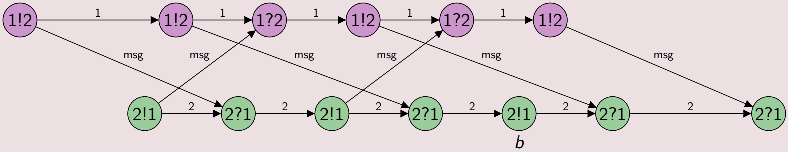
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\Sigma_1 = \{1!2\,,\ 1?2\}$ $\Sigma_2 = \{2!1\,,\ 2?1\}$

Communicating Automaton



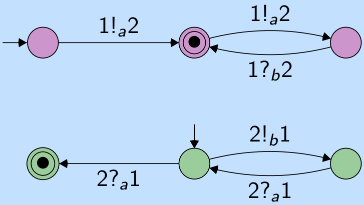
Message Sequence Chart (MSC)



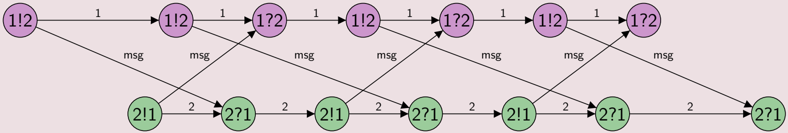
Communicating Automata and MSCs

$Proc = \{1, 2\} \qquad \Sigma_1 = \{1!2\,,\ 1?2\} \qquad \Sigma_2 = \{2!1\,,\ 2?1\}$

Communicating Automaton



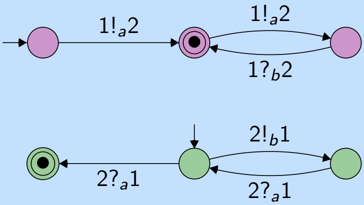
Message Sequence Chart (MSC)



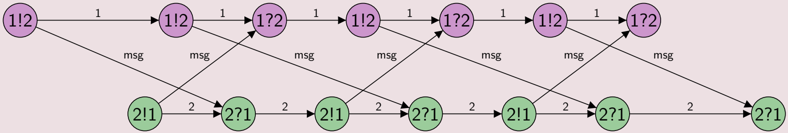
Communicating Automata and MSCs

$Proc = \{1, 2\}$ $\quad\quad \Sigma_1 = \{1!2\,,\, 1?2\}$ $\quad\quad \Sigma_2 = \{2!1\,,\, 2?1\}$

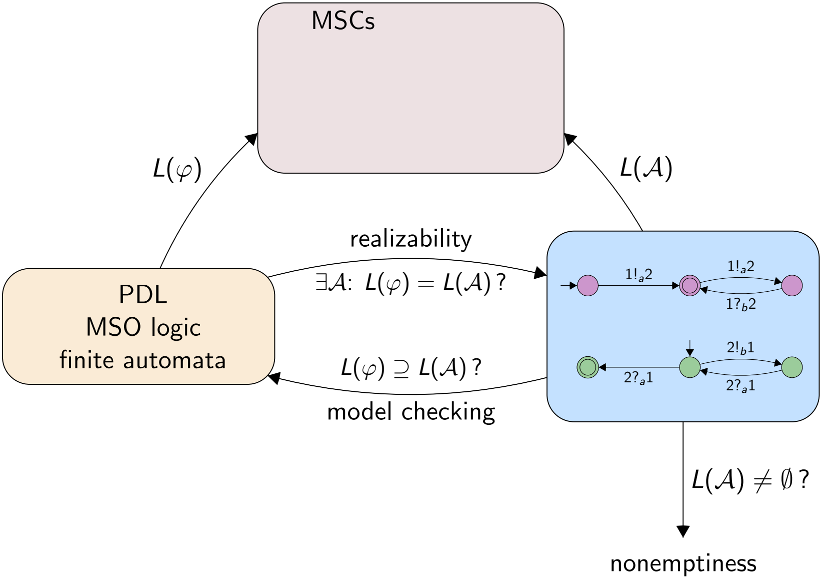
Communicating Automaton



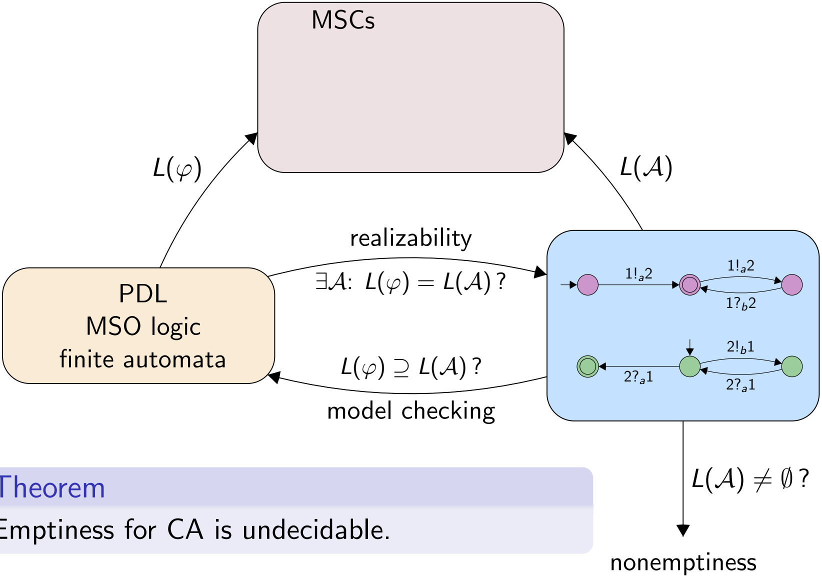
Message Sequence Chart (MSC) $\quad M = (E, \to_1, \to_2, \xrightarrow{\text{msg}}, \lambda)$



Message-Passing Systems



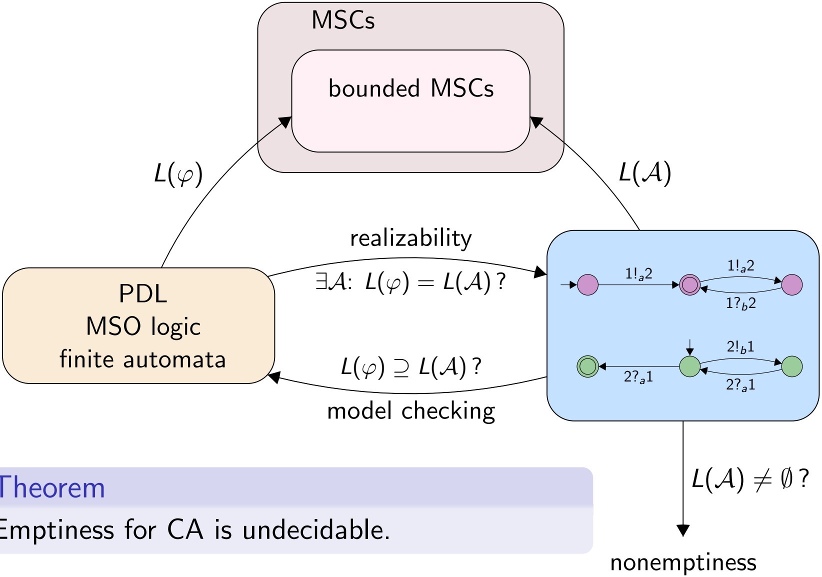
Message-Passing Systems



Theorem

Emptiness for CA is undecidable.

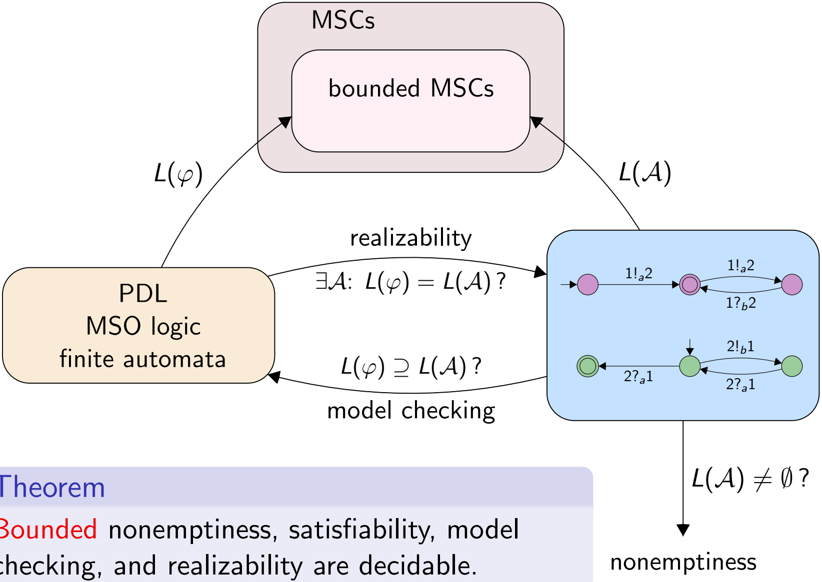
Message-Passing Systems



Theorem

Emptiness for CA is undecidable.

Message-Passing Systems

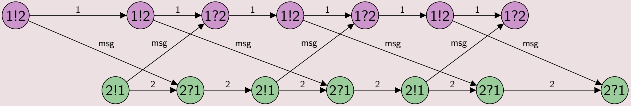


Theorem

Bounded nonemptiness, satisfiability, model checking, and realizability are decidable.

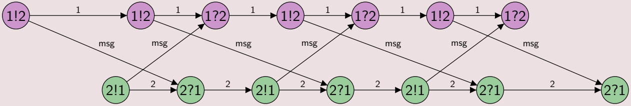
Channel-Bounded MSCs

MSC M

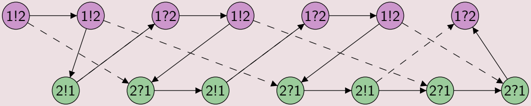


Channel-Bounded MSCs

MSC M

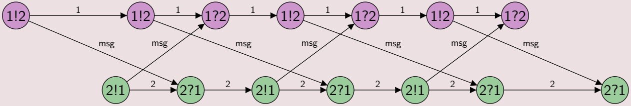


3-bounded linearization $w \in Lin(M) \subseteq \Sigma^*$ $\rightsquigarrow$ $msc(w) = M$

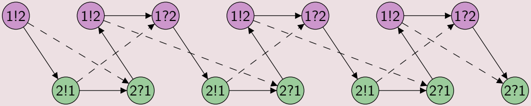


Channel-Bounded MSCs

MSC M

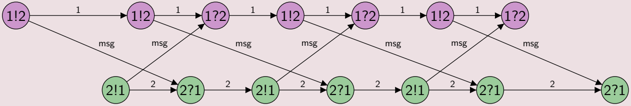


1-bounded linearization $w \in Lin(M) \subseteq \Sigma^*$ $\rightsquigarrow$ $msc(w) = M$

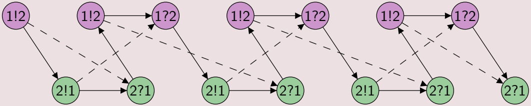


Channel-Bounded MSCs

MSC M



1-bounded linearization $w \in Lin(M) \subseteq \Sigma^*$ $\rightsquigarrow$ $msc(w) = M$



Definition

Let $B \in \mathbb{N}$. An MSC is

- $\exists B$-bounded if some linearization is B-bounded linearization.
- $\forall B$-bounded if every linearization is B-bounded.

Representations

Definition

A set $L \subseteq \Sigma^*$ (of well-formed words) is a

Representations

Definition

A set $L \subseteq \Sigma^*$ (of well-formed words) is a

- $\exists B$-representation if, for all MSCs M, L contains either
  - all B-bounded linearizations of M, or
  - none of its linearizations.

Representations

Definition

A set $L \subseteq \Sigma^*$ (of well-formed words) is a

- $\exists B$-representation if, for all MSCs M, L contains either
  - all B-bounded linearizations of M, or
  - none of its linearizations.
- $\forall$-representation if, for all MSCs M, L contains either
  - all linearizations of M, or
  - none of its linearizations.

Representations

Definition

A set $L \subseteq \Sigma^*$ (of well-formed words) is a

- $\exists B$-representation if, for all MSCs M, L contains either
  - all B-bounded linearizations of M, or
  - none of its linearizations.
- $\forall$-representation if, for all MSCs M, L contains either
  - all linearizations of M, or
  - none of its linearizations.

Example

$\left(1!2\ 2?1\right)^*$ is an $\exists 1$-representation, but no $\forall$-representation.

Representations

Definition

A set $L \subseteq \Sigma^*$ (of well-formed words) is a

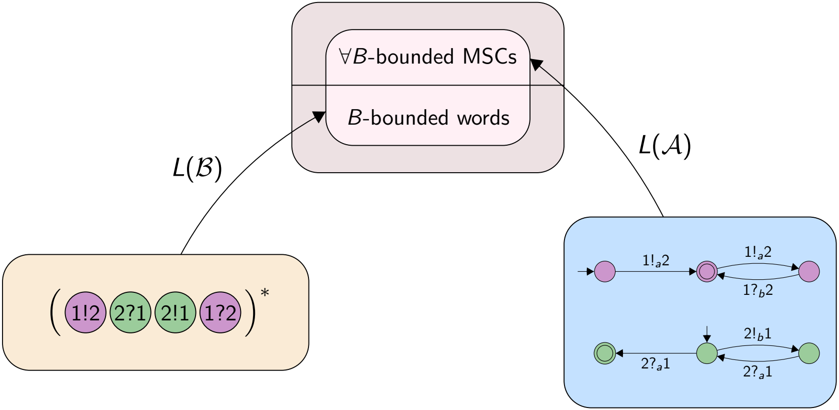
- $\exists B$-representation if, for all MSCs M, L contains either
  - all B-bounded linearizations of M, or
  - none of its linearizations.
- $\forall$-representation if, for all MSCs M, L contains either
  - all linearizations of M, or
  - none of its linearizations.

Example

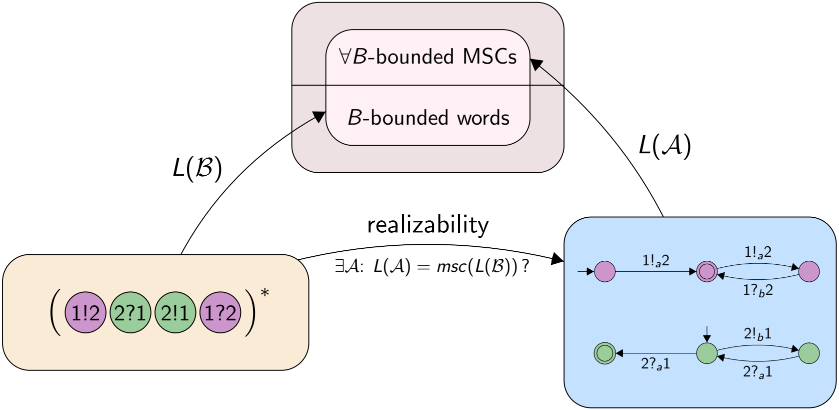
$\left(1!2 \; 2?1 \right)^*$ is an $\exists 1$-representation, but no $\forall$-representation.

$\left(1!2 \; 2?1 \quad 3!4 \; 4?3 \right)^*$ is not an $\exists B$-representation, for any B.

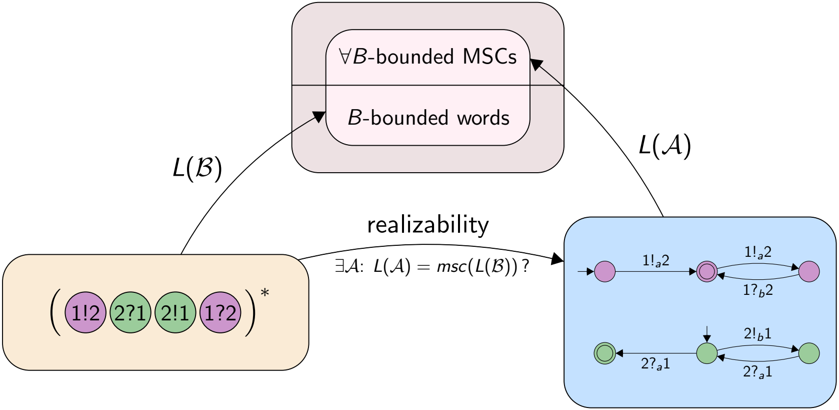
Message-Passing Systems



Message-Passing Systems



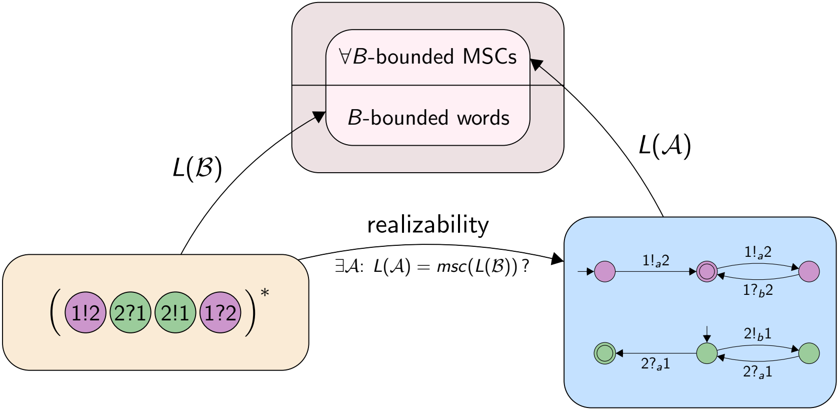
Message-Passing Systems



Theorem (Henriksen et al. '00; Kuske '03)

Let $\mathcal{B}$ be some finite automaton such that $L(\mathcal{B})$ is a $\forall$-representation.
There is a (deterministic) CA $\mathcal{A}$ such that $L(\mathcal{A}) = \mathit{msc}(L(\mathcal{B}))$.

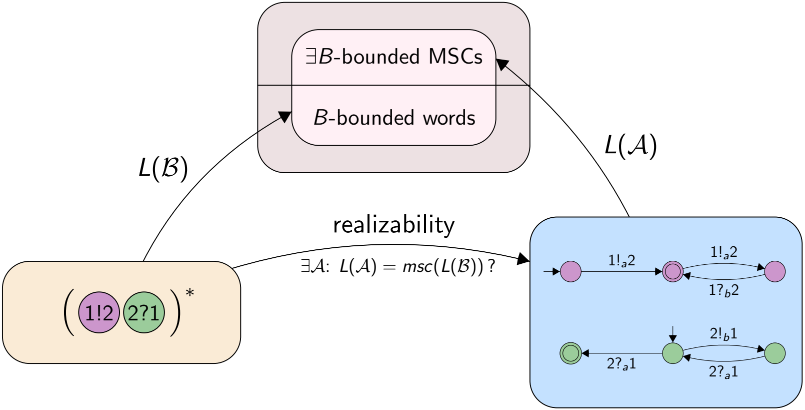
Message-Passing Systems



Theorem (Henriksen et al. '00)

For a finite automaton $\mathcal{B}$ it is decidable if $L(\mathcal{B})$ is a $\forall$-representation.

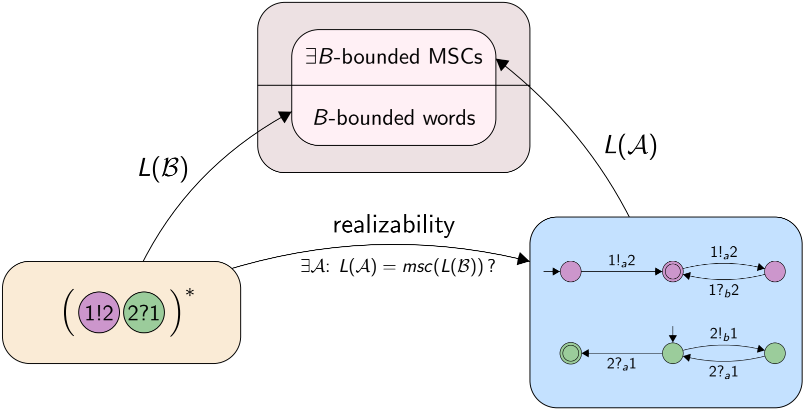
Message-Passing Systems



Theorem (Genest-Kuske-Muscholl '06)

Let $\mathcal{B}$ be some finite automaton such that $L(\mathcal{B})$ is a $\exists B$-representation. There is a CA $\mathcal{A}$ such that $L(\mathcal{A}) = msc(L(\mathcal{B}))$.

Message-Passing Systems



Theorem

For a finite automaton $\mathcal{B}$ it is decidable if $L(\mathcal{B})$ is an $\exists B$-representation.

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

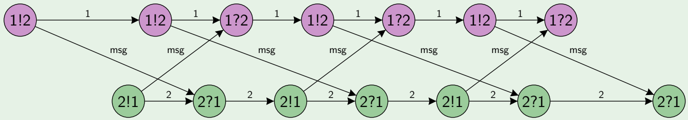
- $x \rightarrow_p y$ — x and y are successive events on process $p \in Proc$
- $x \stackrel{\text{msg}}{\rightarrow} y$ — x and y form a message
- $a(x)$ — event x is labeled with $a \in \Sigma$

Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

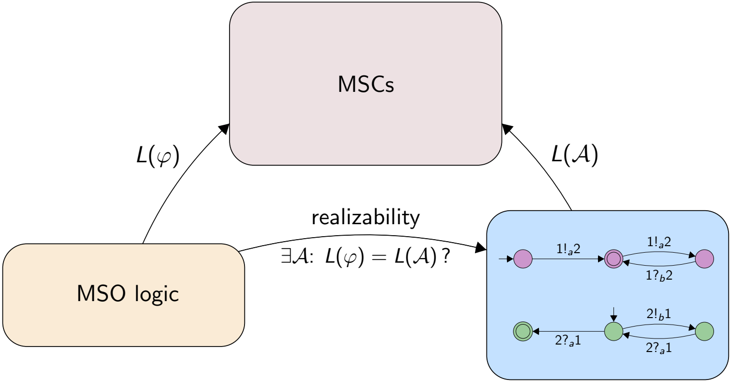
- $x \rightarrow_p y$ — x and y are successive events on process $p \in Proc$
- $x \xrightarrow{\text{msg}} y$ — x and y form a message
- $a(x)$ — event x is labeled with $a \in \Sigma$

Example



$$\models \; \exists x, y, x', y' \, (x \xrightarrow{\text{msg}} y \wedge x' \xrightarrow{\text{msg}} y' \wedge x \rightarrow_1^* y' \wedge x' \rightarrow_2^* y)$$

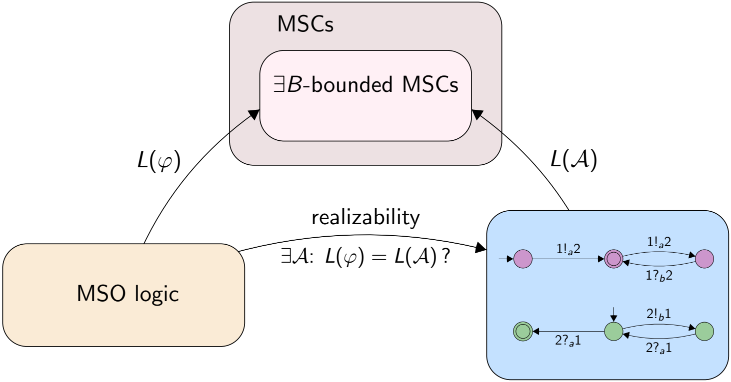
Message-Passing Systems



Theorem (B.-Leucker '04)

EMSO logic ($\exists X_1 \ldots X_n \varphi$ with φ first-order) and communicating automata are expressively equivalent. MSO logic is strictly more expressive.

Message-Passing Systems

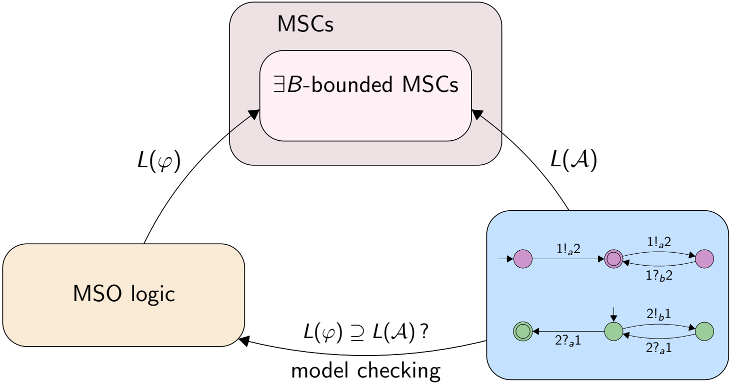


Theorem (Genest-Kuske-Muscholl '04)

Let L be a set of $\exists B$-bounded MSCs. The following are equivalent:

- There is an MSO sentence φ such that $L = L(\varphi)$.
- There is a CA $\mathcal{A}$ such that $L = L(\mathcal{A})$.

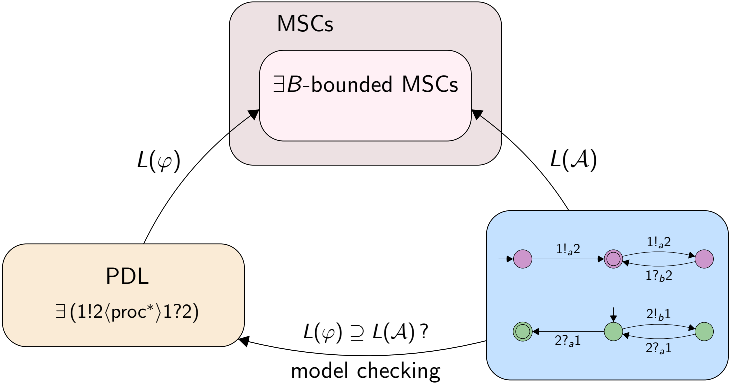
Message-Passing Systems



Theorem (Genest-Kuske-Muscholl '04)

Given a CA $\mathcal{A}$ and an MSO sentence φ, it is decidable if all $\exists B$-bounded MSCs from $L(\mathcal{A})$ satisfy φ.

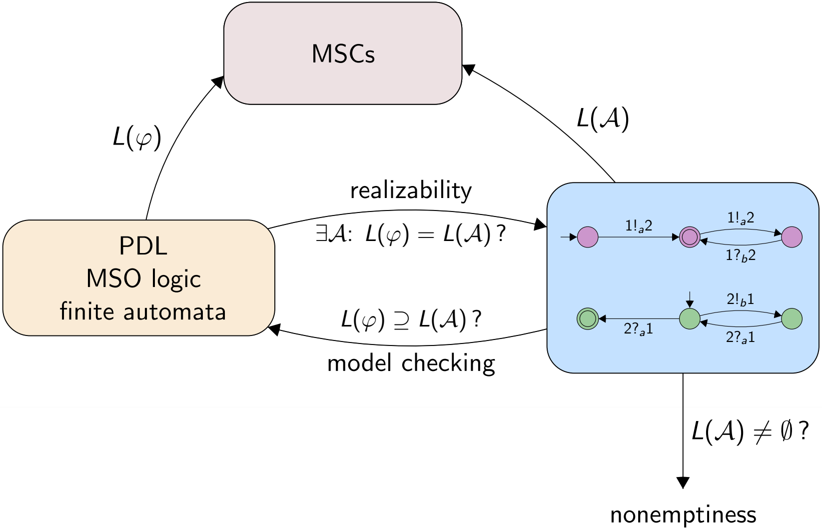
Message-Passing Systems



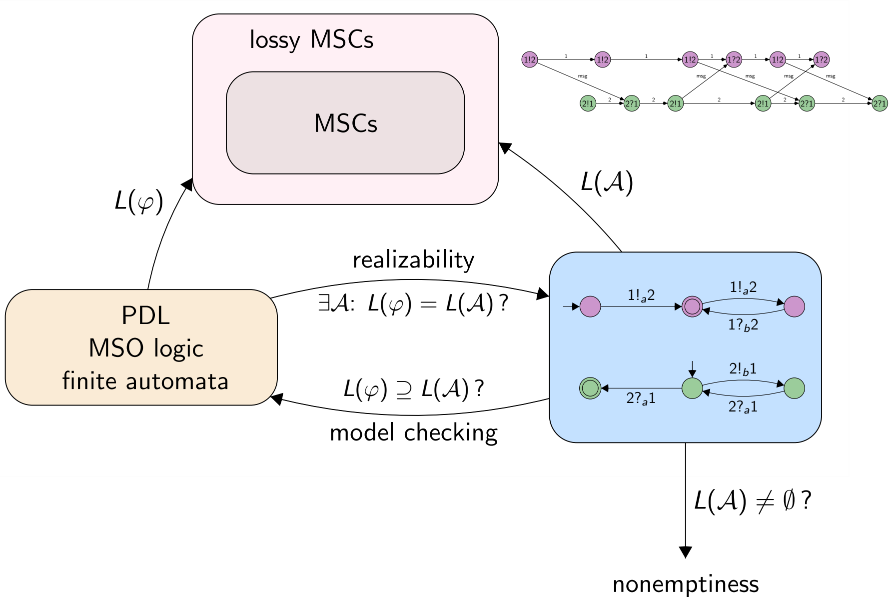
Theorem (B., Kuske, Meinecke 2007; Mennicke 2012)

Given a CA $\mathcal{A}$ and a PDL formula φ, it is decidable in PSPACE if all $\exists B$-bounded MSCs from $L(\mathcal{A})$ satisfy φ.

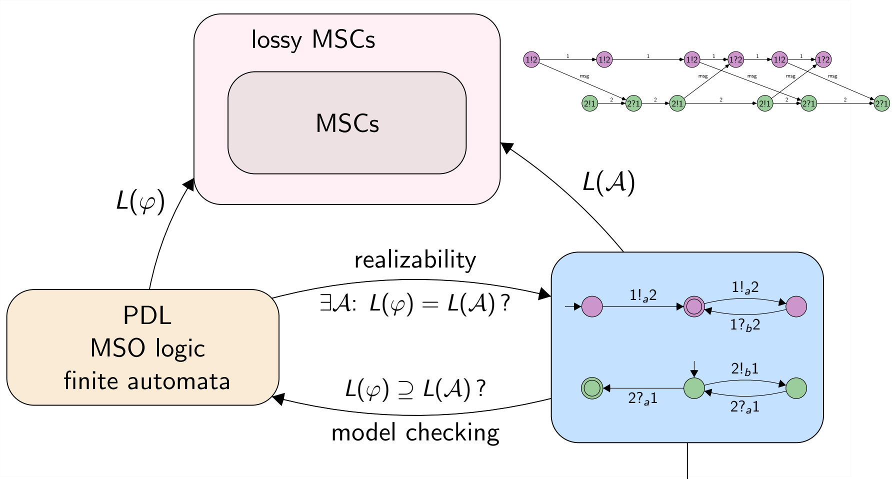
Message-Passing Systems



Message-Passing Systems



Message-Passing Systems

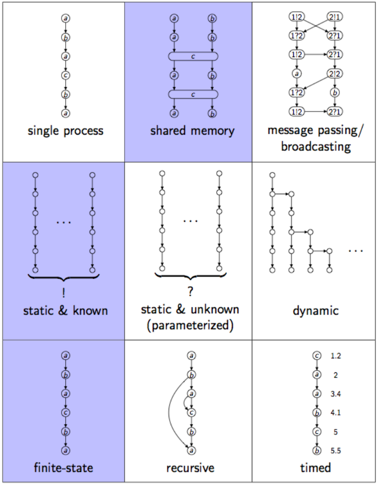


Theorem (Finkel '87, Abdulla-Jonsson '96)

Emptiness for lossy CA is decidable.

7. Conclusion and Perspectives

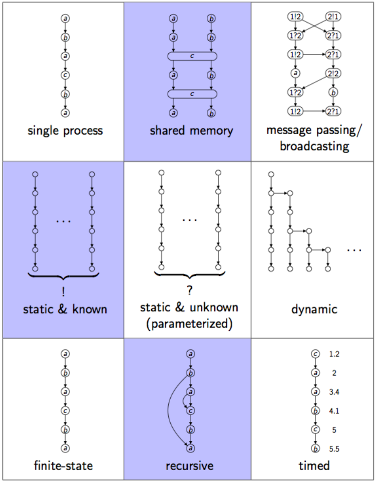
Conclusion: Finite-State Shared-Memory Systems



Realizability ✔

Model Checking ✔

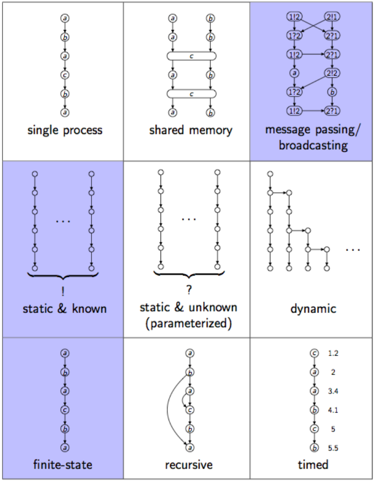
Conclusion: Recursive Shared-Memory Systems



Realizability ✔

Model Checking ✔

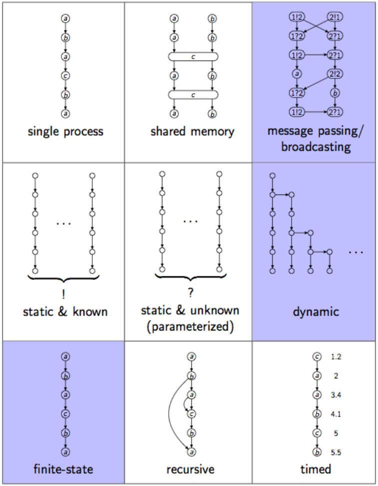
Conclusion: Message-Passing Systems



Realizability ✔

Model Checking ✔

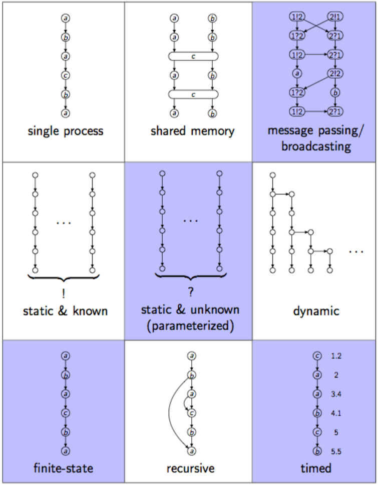
Perspectives: Dynamic Message-Passing Systems



Realizability ✔ ✗

Model Checking ✔ ✗

Perspectives: Parameterized Systems



Realizability ✗

Model Checking ✓ ✗

Reachability ✓

Thank You!