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, ...)

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
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:

```
  Communication medium

  Process 1  Process 2  ...  Process n
```

- **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:

  ![Diagram of concurrent system](https://via.placeholder.com/150)

  Communication medium

  ↑↓ ↑↓ ↑↓

  Process 1  Process 2  ...  Process n

- 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

- **single process**
  - a
  - b
  - a
  - c
  - b
  - a

- **shared memory**
  - a
  - b
  - a
  - b
  - c
  - a
  - b

Classification and Objectives
Form of communication

- **single process**
- **shared memory**
- **message passing/broadcasting**
System architecture

... static & known

\begin{itemize}
\item ...
\item static & known
\end{itemize}
System architecture

Classification and Objectives
System architecture

- Static & known
- Static & unknown (parameterized)
- Dynamic
Type of single process

finite-state
Type of single process

finite-state

recursive
Type of single process

finite-state

recursive

timed
The various settings ...

<table>
<thead>
<tr>
<th>Single process</th>
<th>Shared memory</th>
<th>Message passing/ broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Single process diagram" /></td>
<td><img src="image2.png" alt="Shared memory diagram" /></td>
<td><img src="image3.png" alt="Message passing diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Static &amp; Known</th>
<th>Static &amp; Unknown (parameterized)</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image4.png" alt="Static &amp; known diagram" /></td>
<td><img src="image5.png" alt="Static &amp; unknown diagram" /></td>
<td><img src="image6.png" alt="Dynamic diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Finite-state</th>
<th>Recursive</th>
<th>Timed</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image7.png" alt="Finite-state diagram" /></td>
<td><img src="image8.png" alt="Recursive diagram" /></td>
<td><img src="image9.png" alt="Timed diagram" /></td>
</tr>
</tbody>
</table>
The various settings ...

<table>
<thead>
<tr>
<th>Single Process</th>
<th>Shared Memory</th>
<th>Message Passing/Broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static &amp; Known</td>
<td>Static &amp; Unknown (Parameterized)</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Finite-State</td>
<td>Recursive</td>
<td>Timed</td>
</tr>
</tbody>
</table>
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 ...

<table>
<thead>
<tr>
<th>Single process</th>
<th>Shared memory</th>
<th>Message passing/broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static &amp; known</td>
<td>Static &amp; unknown (parameterized)</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Finite-state</td>
<td>Recursive</td>
<td>Timed</td>
</tr>
</tbody>
</table>
The various settings ...

- **Behavior**
  - Words

- **System model**
  - Finite automata
  - Kripke structures

<table>
<thead>
<tr>
<th>Setting</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single process</td>
<td><img src="single_process.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Shared memory</td>
<td><img src="shared_memory.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Message passing/broadcasting</td>
<td><img src="message_passing.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Static &amp; known</td>
<td><img src="static_known.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Static &amp; unknown (parameterized)</td>
<td><img src="static_unknown.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Dynamic</td>
<td><img src="dynamic.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Finite-state</td>
<td><img src="finite_state.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Recursive</td>
<td><img src="recursive.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Timed</td>
<td><img src="timed.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>
The various settings ...

[Diagram showing various settings and their corresponding models:
- Single process
- Shared memory
- Message passing/broadcasting
- Static & known
- Static & unknown (parameterized)
- Dynamic
- Finite-state
- Recursive
- Timed

Behavior
- Words

System model
- Finite automata
- Kripke structures

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

Classification and Objectives
The various settings ...

<table>
<thead>
<tr>
<th>Single Process</th>
<th>Shared Memory</th>
<th>Message Passing/Broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td>static &amp; known</td>
<td>static &amp; unknown (parameterized)</td>
<td>dynamic</td>
</tr>
<tr>
<td>finite-state</td>
<td>recursive</td>
<td>timed</td>
</tr>
</tbody>
</table>
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 ...
The various settings ...

<table>
<thead>
<tr>
<th>Single process</th>
<th>Shared memory</th>
<th>Message passing/broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static &amp; known</td>
<td>Static &amp; unknown (parameterized)</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Finite-state</td>
<td>Recursive</td>
<td>Timed</td>
</tr>
</tbody>
</table>
The various settings ...

<table>
<thead>
<tr>
<th>Single process</th>
<th>Shared memory</th>
<th>Message passing/broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
<td><img src="image3" alt="Diagram" /></td>
</tr>
</tbody>
</table>

- **Behavior**
  - Message sequence charts

<table>
<thead>
<tr>
<th>Static &amp; known</th>
<th>Static &amp; unknown (parameterized)</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image4" alt="Diagram" /></td>
<td><img src="image5" alt="Diagram" /></td>
<td><img src="image6" alt="Diagram" /></td>
</tr>
</tbody>
</table>

- **Classification and Objectives**
  - Finite-state
  - Recursive
  - Timed
The various settings ...

- **single process**
- **shared memory**
- **message passing/broadcasting**

**Behavior**
- Message sequence charts

**System model**
- Communicating automata
  [Brand-Zafiropulo '83]
- Lossy channel systems
  [Finkel '87, Abdulla-Jonsson '96]
The various settings ...
The various settings ...

single process  
shared memory  
message passing/broadcasting

static & known  
static & unknown (parameterized)  
dynamic

finite-state  
recursive  
timed
The various settings ...

Behavior

- Dynamic message sequence charts
The various settings ...

<table>
<thead>
<tr>
<th>Setting</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single process</td>
<td>![Diagram]</td>
</tr>
<tr>
<td>Shared memory</td>
<td>![Diagram]</td>
</tr>
<tr>
<td>Message passing/broadcasting</td>
<td>![Diagram]</td>
</tr>
</tbody>
</table>

**Behavior**
- Dynamic message sequence charts

**System model**
- Dynamic communicating automata
  - [B., Cyriac, Héluët, Kara, Schwentick '13]
The various settings ...

Behavior
- Dynamic message sequence charts

System model
- Dynamic communicating automata
  [B., Cyriac, Héloüët, Kara, Schwentick '13]

Specification
- High-level expressions with registers

Classification and Objectives
The various settings …

- Single process
- Shared memory
- Message passing/broadcasting

- Static & known
- Static & unknown (parameterized)
- Dynamic

- Finite-state
- Recursive
- Timed
The various settings ...

- Single process
- Shared memory
- Message passing/broadcasting

- Static & known
- Static & unknown (parameterized)
- Dynamic

- Finite-state
- Recursive
- Timed

Behavior

▶ Words?
The various settings ...

**Behavior**

- Words?

**System model**

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

---

**Classification and Objectives**
The various settings ...

- **Behavior**
  - Words?

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

- **Specification**
  - Reachability questions
The various settings ...

- **single process**
- **shared memory**
- **message passing/broadcasting**

- **static & known**
- **static & unknown (parameterized)**
- **dynamic**

- **finite-state**
- **recursive**
- **timed**
The various settings ...

Behavior
- Nested traces
The various settings ...

Classifcation and Objectives

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

Asynchronous automata
Message-passing automata
Multi-stack automata
Landscape and Objectives

MSO logic
Temporal logic
High-level expressions

Words
Mazurkiewicz traces
Message Sequence Charts
Nested words

\( L(\varphi) \)

Asynchronous automata
Message-passing automata
Multi-stack automata

\( L(\mathcal{A}) \)
Classification and Objectives

Landscape and Objectives

- Words
- Mazurkiewicz traces
- Message Sequence Charts
- Nested words

\[ L(\varphi) \quad \text{realizability} \quad L(\mathcal{A}) \]

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

- MSO logic
- Temporal logic
- High-level expressions

- Asynchronous automata
- Message-passing automata
- Multi-stack automata
Landscape and Objectives

Words
Mazurkiewicz traces
Message Sequence Charts
Nested words

$L(\varphi)$

realizability

$\exists A: L(\varphi) = L(A)$?

$L(\varphi) \supseteq L(A)$?

model checking

MSO logic
Temporal logic
High-level expressions

$\varphi$

Asynchronous automata
Message-passing automata
Multi-stack automata

A
Landscape and Objectives

- MSO logic
- Temporal logic
- High-level expressions
- \( \varphi \)
- Asynchronous automata
- Message-passing automata
- Multi-stack automata
- Words
- Mazurkiewicz traces
- Message Sequence Charts
- Nested words

### Realizability

\[ \exists A : L(\varphi) = L(A) ? \]

### Model Checking

\[ L(\varphi) \supseteq L(A) ? \]

### Satisfiability

\[ L(\varphi) \neq \emptyset ? \]

---

Fomulae and Objectives
Landscape and Objectives

- MSO logic
- Temporal logic
- High-level expressions

Words
Mazurkiewicz traces
Message Sequence Charts
Nested words

\( L(\varphi) \)
\( L(A) \)

realizability
\[ \exists A: L(\varphi) = L(A) ? \]

model checking
\[ L(\varphi) \supseteq L(A) ? \]

satisfiability
\[ L(\varphi) \neq \emptyset ? \]

nonemptiness
\[ L(A) \neq \emptyset ? \]
Landscape and Objectives: Linear-Time Setting

- MSO logic
- Temporal logic
- High-level expressions
- \( \varphi \)
- Asynchronous automata
- Message-passing automata
- Multi-stack automata
- Words
- Mazurkiewicz traces
- Message Sequence Charts
- Nested words

\[ L(\varphi) = L(A) \] 
realizability

\[ L(\varphi) \supseteq L(A) \] 
model checking

\[ L(\varphi) \neq \emptyset \] 
satisfiability

\[ L(A) \neq \emptyset \] 
nonemptiness
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

\[ \{a, b, c\}^* \]

\[ L(\varphi) \]

\[ L(A) \]

**LTL**

\[ G(a \rightarrow Fb) \]

\[ ((b + c)^* a(a + c)^* b)^* \]

\[ \forall x (a(x) \rightarrow \exists y (x \leq y \land b(y))) \]

**RExp**

**MSO**

**Diagram**

State: \( b, c \)

Transition: \( a \rightarrow b \)

State: \( a, c \)

Transition: \( b \rightarrow a \)

State: \( b \)

Transition: \( a \rightarrow b \)
Finite-State Sequential Systems

\[ \{a, b, c\}^* \]

\(L(\varphi)\)

\(L(A)\)

realizability

\[ \exists A: L(\varphi) = L(A) ? \]

LTL

RExp

MSO

\[ G(a \rightarrow Fb) \]

\[ ((b + c)^* a(a + c)^* b)^* \]

\[ \forall x (a(x) \rightarrow \exists y (x \leq y \land b(y))) \]

\[ a \rightarrow Fb \]

\[ ((b + c)^* a(a + c)^* b)^* \]

\[ \forall x (a(x) \rightarrow \exists y (x \leq y \land b(y))) \]
Finite-State Sequential Systems

Theorem (Büchi-Elgot-Trakhtenbrot ’60s)

Every MSO formula is equivalent to some (deterministic) finite automaton.
Finite-State Sequential Systems

$L(\varphi)$

$L(A)$

$L(\varphi) \supseteq L(A)$?

model checking

$LTL$

$RExp$

$MSO$

$G(a \rightarrow Fb)$

$((b + c)^* a (a + c)^* b)^*$

$\forall x (a(x) \rightarrow \exists y (x \leq y \land b(y)))$

$\{a, b, c\}^*$
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

LTL
MSO logic
finite automata
asynchronous automata
Mazurkiewicz traces
L(ϕ) ⊇ L(A) ?
realizability
∃A: L(ϕ) = L(A) ?
model checking

L(ϕ)
L(A)
Asynchronous Automata and Mazurkiewicz Traces

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \]
Asynchronous Automata and Mazurkiewicz Traces

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

Asynchronous Automaton

\[(s_0) \xrightarrow{a_1} (s_1)\]
\[(s_0, t_0) \xrightarrow{c} (s_0, t_0)\]
\[(s_1, t_1) \xrightarrow{c} (s_2, t_2)\]
\[(s_0, t_1) \xrightarrow{c} (s_0, t_1)\]
Asynchronous Automata and Mazurkiewicz Traces

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \]

Asynchronous Automaton

Mazurkiewicz Trace
Asynchronous Automata and Mazurkiewicz Traces

\( \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \)

Asynchronous Automaton

Mazurkiewicz Trace
Asynchronous Automata and Mazurkiewicz Traces

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \]

**Asynchronous Automaton**

**Mazurkiewicz Trace**
Asynchronous Automata and Mazurkiewicz Traces

\( \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \)

### Asynchronous Automaton

![Diagram of Asynchronous Automaton]

### Mazurkiewicz Trace

![Diagram of Mazurkiewicz Trace]
Asynchronous Automata and Mazurkiewicz Traces

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \]

**Asynchronous Automaton**

**Mazurkiewicz Trace**
Asynchronous Automata and Mazurkiewicz Traces

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

Asynchronous Automaton

Mazurkiewicz Trace
Asynchronous Automata and Mazurkiewicz Traces

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

Asynchronous Automaton

Mazurkiewicz Trace
Asynchronous Automata and Mazurkiewicz Traces

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \]

Asynchronous Automaton

Mazurkiewicz Trace
Asynchronous Automata and Mazurkiewicz Traces

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

Asynchronous Automaton

Mazurkiewicz Trace
Asynchronous Automata and Mazurkiewicz Traces

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \]

Asynchronous Automaton

Mazurkiewicz Trace
Asynchronous Automata and Mazurkiewicz Traces

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \]

Asynchronous Automaton

Mazurkiewicz Trace
Asynchronous Automata and Mazurkiewicz Traces

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \]

Asynchronous Automaton

Mazurkiewicz Trace
Asynchronous Automata and Mazurkiewicz Traces

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \]

### Asynchronous Automaton

![Asynchronous Automaton Diagram]

### Mazurkiewicz Trace

![Mazurkiewicz Trace Diagram]
Mazurkiewicz Traces and Their Linearizations

Mazurkiewicz Trace \( t = (E, \rightarrow_1, \rightarrow_2, \lambda) \) \( \lambda : E \rightarrow \Sigma \overset{\text{def}}{=} \Sigma_1 \cup \Sigma_2 \)
Mazurkiewicz Traces and Their Linearizations

Mazurkiewicz Trace

\[ t = (E, \to_1, \to_2, \lambda) \quad \lambda : E \to \Sigma \overset{\text{def}}{=} \Sigma_1 \cup \Sigma_2 \]

Linearizations

\[ w \in \text{Lin}(t) \subseteq \Sigma^* \quad \sim \quad trace(w) = t \]
Mazurkiewicz Traces and Their Linearizations

Mazurkiewicz Trace

\[ t = (E, \rightarrow_1, \rightarrow_2, \lambda) \]

\[ \lambda : E \rightarrow \Sigma \overset{\text{def}}{=} \Sigma_1 \cup \Sigma_2 \]

Linearizations

\[ w \in \text{Lin}(t) \subseteq \Sigma^* \]

\[ \sim \rightarrow \text{trace}(w) = t \]
Mazurkiewicz Traces and Their Linearizations

**Mazurkiewicz Trace**

\[ t = (E, \rightarrow_1, \rightarrow_2, \lambda) \]

where

\[ \lambda : E \rightarrow \Sigma \overset{\text{def}}{=} \Sigma_1 \cup \Sigma_2 \]

\[ a_1 \quad 1 \quad 1 \quad a_1 \quad 1 \quad 1 \quad b_1 \quad 1 \quad 1 \quad b_1 \]

\[ c \quad \quad \quad \quad \quad \quad \quad \quad c \quad \quad \quad \quad \quad \quad \quad \quad b_2 \quad 2 \quad 2 \quad b_2 \]

\[ a_2 \quad 2 \quad 2 \quad a_2 \quad 2 \quad 2 \quad b_2 \quad 2 \quad 2 \quad b_2 \]

**Linearizations**

\[ w \in \text{Lin}(t) \subseteq \Sigma^* \overset{\sim}{\sim} \text{trace}(w) = t \]

\[ a_1 \quad \sim \quad a_1 \quad c \quad \sim \quad a_1 \quad a_2 \quad c \quad b_1 \quad b_1 \quad b_2 \quad b_2 \]

\[ a_1 \quad \sim \quad a_1 \quad c \quad \sim \quad a_1 \quad a_2 \quad c \quad b_1 \quad b_1 \quad b_2 \quad b_2 \]

\[ a_1 \quad \sim \quad a_1 \quad c \quad \sim \quad a_1 \quad a_2 \quad c \quad b_1 \quad b_1 \quad b_2 \quad b_2 \]
Mazurkiewicz Traces and Their Linearizations

Mazurkiewicz Trace

\[ t = (E, \rightarrow_1, \rightarrow_2, \lambda) \]

\[ \lambda : E \rightarrow \Sigma \overset{\text{def}}{=} \Sigma_1 \cup \Sigma_2 \]

Linearizations

\[ w \in \text{Lin}(t) \subseteq \Sigma^* \overset{\sim}{\rightsquigarrow} \text{trace}(w) = t \]
Finite-State Shared-Memory Systems

\[ L(B) \]

\[ L(A) \]

\[ L(B) \]

\[ L(A) \]
Finite-State Shared-Memory Systems

traces
Σ

realizability
∃A: L(A) = trace(L(B)) ?

L(B)
L(A)
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 $A$ such that $L(A) = \text{trace}(L(B))$. 

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 $p \in \text{Proc}$
Monadic Second-Order Logic

<table>
<thead>
<tr>
<th>Monadic Second-Order Logic (MSO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \rightarrow^p y$</td>
</tr>
<tr>
<td>$a(x)$</td>
</tr>
</tbody>
</table>

Example

$\exists x \exists y (b_1(x) \land b_2(y) \land x \leq y)$

where $\leq = (\rightarrow_1 \cup \rightarrow_2)^*$
Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

- $x \rightarrow_p y$  
  $x$ and $y$ are successive events on process $p \in \text{Proc}$

- $a(x)$  
  Event $x$ is labeled with $a \in \Sigma$

- $x = y$

Example

$$a_1 \quad c \quad a_2 \quad c \quad a_1 \quad c \quad a_2 \quad c \quad b_1 \quad c \quad b_2$$

where $\leq = (\rightarrow_1 \cup \rightarrow_2)^*$
Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

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

Example

\[
\begin{align*}
&\exists x \exists y (b_1(x) \land b_2(y) \land x \leq y) \\
&\text{where} \quad \leq = (\rightarrow_1 \cup \rightarrow_2)^* 
\end{align*}
\]
Monadic Second-Order Logic

### Monadic Second-Order Logic (MSO)

- **$x \rightarrow_p y$** - $x$ and $y$ are successive events on process $p \in \text{Proc}$
- **$a(x)$** - 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 $\varphi$

Example:

\[
|x|y|
\]

where

$\leq = (\rightarrow_1 \cup \rightarrow_2)^*$
Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

- $x \rightarrow_p y$ \hspace{1cm} $x$ and $y$ are successive events on process $p \in \text{Proc}$
- $a(x)$ \hspace{1cm} event $x$ is labeled with $a \in \Sigma$
- $x = y$ \hspace{1cm} $x$ is equal to $y$
- $x \in X$ \hspace{1cm} event $x$ is contained in set of events $X$
- $\exists x \varphi$ \hspace{1cm} there is event $x$ such that $\varphi$
- $\exists X \varphi$ \hspace{1cm} there is set of event $X$ such that $\varphi$
## Monadic Second-Order Logic

### Monadic Second-Order Logic (MSO)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \rightarrow_p y$</td>
<td>$x$ and $y$ are successive events on process $p \in \text{Proc}$</td>
</tr>
<tr>
<td>$a(x)$</td>
<td>Event $x$ is labeled with $a \in \Sigma$</td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
</tr>
<tr>
<td>$x \in X$</td>
<td>Event $x$ is contained in set of events $X$</td>
</tr>
<tr>
<td>$\exists x \varphi$</td>
<td>There is event $x$ such that $\varphi$</td>
</tr>
<tr>
<td>$\exists X \varphi$</td>
<td>There is set of event $X$ such that $\varphi$</td>
</tr>
<tr>
<td>$\neg \varphi$</td>
<td>$\varphi \lor \psi$</td>
</tr>
</tbody>
</table>
Monadic Second-Order Logic

Monadic Second-Order Logic (MSO)

- \( x \rightarrow_p y \): \( x \) and \( y \) are successive events on process \( p \in \text{Proc} \)
- \( a(x) \): 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 \( \varphi \)
- \( \exists X \varphi \): there is set of event \( X \) such that \( \varphi \)
- \( \neg \varphi \) \( \varphi \lor \psi \)

Example

\[ \models \exists x \exists y (b_1(x) \land b_2(y) \land x \leq y) \]

where \( \leq = (\rightarrow_1 \cup \rightarrow_2)^* \)
Theorem (Thomas ’90)
MSO logic and asynchronous automata are expressively equivalent.
Theorem (Thomas ’90)

MSO logic and asynchronous automata are expressively equivalent.

⇒ MSO model checking is decidable.
Global Temporal Logic

\[ \text{LTrL} \forall \quad \varphi ::= \text{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \text{U} \forall \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma \]
Global Temporal Logic

<table>
<thead>
<tr>
<th>LTrL∀</th>
<th>ϕ ::= tt</th>
<th>⟨a⟩ϕ</th>
<th>ϕ₁ Uₐ ϕ₂</th>
<th>¬ϕ</th>
<th>ϕ₁ ∨ ϕ₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTrL∃</td>
<td>ϕ ::= U∃</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( a \in \Sigma \)
Global Temporal Logic

\[
\text{LTrL}_\forall \quad \phi ::= \text{tt} \mid \langle a \rangle \phi \mid \phi_1 \text{ U } \phi_2 \mid \neg \phi \mid \phi_1 \lor \phi_2 \\
\text{LTrL}_\exists \quad \phi ::= \quad U \exists \\
\quad a \in \Sigma
\]

Semantics

Finite-State Shared-Memory Systems
Global Temporal Logic

\[
\begin{align*}
\text{LTrL}_\forall & \quad \varphi ::= \top \mid \langle a \rangle \varphi \mid \varphi_1 \mathsf{U}_\forall \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma \\
\text{LTrL}_\exists & \quad \varphi ::= \mathsf{U}_\exists 
\end{align*}
\]

Semantics

\[
\models \langle a_1 \rangle \varphi
\]
Global Temporal Logic

LTrL∀ \( \varphi ::= tt \mid \langle a \rangle \varphi \mid \varphi_1 U \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma \)

LTrL∃ \( \varphi ::= \quad U \exists \)

Semantics

\( \models \langle a_1 \rangle \varphi \)

\( \models \varphi U \forall \psi \)
**Global Temporal Logic**

\[
\text{LTrL}_\forall \quad \phi ::= \text{tt} \mid \langle a \rangle \phi \mid \phi_1 \bigoplus \phi_2 \mid \neg \phi \mid \phi_1 \bigvee \phi_2 \quad a \in \Sigma
\]

\[
\text{LTrL}_\exists \quad \phi ::= U_\exists
\]

**Semantics**

\[
\models \langle a_1 \rangle \phi \]

\[
\models \phi \bigoplus \psi
\]
Global Temporal Logic

Global Temporal Logic

\[ \text{LTrL}_\forall \quad \varphi ::= \text{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \text{U}_\forall \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma \]

\[ \text{LTrL}_\exists \quad \varphi ::= \quad \text{U}_\exists \]

Semantics

\[ \models \langle a_1 \rangle \varphi \]

\[ \models \varphi \text{ U}_\forall \psi \]
Global Temporal Logic

LTrL_∀ \quad \varphi ::= \texttt{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \U \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \\
\text{a} \in \Sigma

LTrL_∃ \quad \varphi ::= \U \exists

Semantics

\begin{align*}
\langle a_1 \rangle \varphi & \quad \models \langle a_1 \rangle \varphi \\
\varphi & \quad \models \varphi \U \forall \psi
\end{align*}
# Global Temporal Logic

## Global Temporal Logic

<table>
<thead>
<tr>
<th>$\text{LTrL}_\forall$</th>
<th>$\varphi ::= \text{tt} \mid \langle a \rangle \varphi \mid \varphi _U_ \varphi_2 \mid \neg \varphi \mid \varphi _\lor_ \varphi_2$</th>
<th>$a \in \Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{LTrL}_\exists$</td>
<td>$\varphi ::=$</td>
<td>$_U_ \exists$</td>
</tr>
</tbody>
</table>

## Semantics

\[
\begin{align*}
\langle a_1 \rangle \varphi & \quad \models \quad \langle a_1 \rangle \varphi \\
\varphi & \quad \models \quad \varphi \_U\_ \varphi_2
\end{align*}
\]

Finite-State Shared-Memory Systems
Global Temporal Logic

\[ \begin{align*}
\text{LTrL}_\forall & \quad \varphi ::= \text{tt} \mid \langle a \rangle \varphi \mid \varphi_1 U \forall \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \\
\text{a} & \in \Sigma
\end{align*} \]

\[ \begin{align*}
\text{LTrL}_\exists & \quad \varphi ::= U \exists
\end{align*} \]

Semantics

\[ \begin{align*}
\models \langle a_1 \rangle \varphi \\
\models \varphi \lor \psi
\end{align*} \]
Global Temporal Logic

\[ LTrL_\forall \quad \varphi ::= \text{tt} \mid \langle a \rangle \varphi \mid \varphi_1 U_\forall \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma \]

\[ LTrL_\exists \quad \varphi ::= U_\exists \]

Semantics

\[ \models \langle a_1 \rangle \varphi \]
\[ \models \varphi U_\forall \psi \]
Global Temporal Logic

**LTrL**: \( \varphi ::= \text{tt} \mid \langle a \rangle \varphi \mid \varphi_1 U \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma \)

**LTrL\(\exists\)**: \( \varphi ::= \exists \varphi \mid \exists \psi \)

**Semantics**

\[ \begin{array}{c}
\langle a_1 \rangle \varphi \quad \varphi \\
\Rightarrow \quad \langle a_1 \rangle \varphi
\end{array} \]

\[ \begin{array}{c}
\psi \quad \varphi \quad \varphi \\
\Rightarrow \quad \varphi U \varphi
\end{array} \]
Global Temporal Logic

LTrL∀ \( \varphi ::= tt \mid \langle a \rangle \varphi \mid \varphi_1 U∀ \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma \)

LTrL∃ \( \varphi ::= \quad U∃ \)

Semantics

\( \models \langle a_1 \rangle \varphi \)

\( \models \varphi U∀ \psi \)
Global Temporal Logic

\[ \text{LTrL}_\forall \quad \varphi ::= \text{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \text{ U } \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma \]

\[ \text{LTrL}_\exists \quad \varphi ::= \quad \text{U } \exists \]

Semantics

\[ \models \langle a_1 \rangle \varphi \]

\[ \models \text{tt U } \forall \langle b_1 \rangle \langle b_2 \rangle \text{tt} \]

Finite-State Shared-Memory Systems
Global Temporal Logic

\[ \text{LTrL}_\forall \quad \varphi ::= \text{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \text{U} \forall \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma \]

\[ \text{LTrL}_\exists \quad \varphi ::= \exists \]

Semantics

\[ \models \langle a_1 \rangle \varphi \]

\[ \models \text{tt U} \forall \langle b_1 \rangle \langle b_2 \rangle \text{tt} \]

\[ \models \varphi \text{U} \exists \psi \]
Global Temporal Logic

\[ \text{LTrL}_\forall \]
\[ \varphi ::= \text{tt} \mid \langle a \rangle \varphi \mid \varphi_1 U \forall \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma \]

\[ \text{LTrL}_\exists \]
\[ \varphi ::= \text{U} \exists \]

**Semantics**

\[ \models \langle a_1 \rangle \varphi \]
\[ \models \text{tt U} \forall \langle b_1 \rangle \langle b_2 \rangle \text{tt} \]
\[ \models \varphi \text{ U} \exists \psi \]
Global Temporal Logic

$LTrL_\forall \quad \varphi ::= tt \mid \langle a \rangle \varphi \mid \varphi_1 U \forall \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma$

$LTrL_\exists \quad \varphi ::= \quad U \exists$

Semantics

$\models \langle a_1 \rangle \varphi$

$\models tt U \forall \langle b_1 \rangle \langle b_2 \rangle tt$

$\models \varphi U \exists \psi$
Global Temporal Logic

\[ \text{LTrL}_\forall \quad \varphi ::= \text{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \text{U}_\forall \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \quad a \in \Sigma \]

\[ \text{LTrL}_\exists \quad \varphi ::= \text{U}_\exists \]

Semantics

\[ \models \langle a_1 \rangle \varphi \]

\[ \models \text{tt} \text{U}_\forall \langle b_1 \rangle \langle b_2 \rangle \text{tt} \]

\[ \models \varphi \text{U}_\exists \psi \]
Global Temporal Logic

LTrL_∀ \quad \varphi ::= \text{tt} \mid \langle a \rangle \varphi \mid \varphi_1 \text{U}_∀ \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 
\quad a \in \Sigma

LTrL_∃ \quad \varphi ::= \text{U}_∃

Semantics

\begin{align*}
\models & \langle a_1 \rangle \varphi \\
\models & \text{tt} \text{U}_∀ \langle b_1 \rangle \langle b_2 \rangle \text{tt} \\
\models & \varphi \text{U}_∃ \psi
\end{align*}
Finite-State Shared-Memory Systems

LTrL∀
LTrL∃
traces

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

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

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

LTrL∀
LTrL∃
traces

$L(\varphi)$  $L(\mathcal{A})$
Finite-State Shared-Memory Systems

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

- $\text{LTrL}_\forall$ model checking is nonelementary.
Theorem (Walukiewicz '98; Alur-McMillan-Peled '98)

- $\text{LTrL}_\forall$ model checking is nonelementary.
- $\text{LTrL}_\exists$ model checking is undecidable.
Local Temporal Logic

\[ \varphi ::= a \mid \text{EX}\varphi \mid \text{EX}_p\varphi \mid \varphi_1 U \varphi_2 \mid \varphi_1 U_p \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \]

\[ a \in \Sigma, \ p \in \text{Proc} \]
Local Temporal Logic

\[ \varphi ::= a \mid \text{EX}\varphi \mid \text{EX}_p\varphi \mid \varphi_1 \mathbf{U} \varphi_2 \mid \varphi_1 \mathbf{U}_p \varphi_2 \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \]

\( a \in \Sigma, \ p \in \text{Proc} \)

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

- \( t, e \models \text{EX}\varphi \) if there is \( f \in E \) such that \( e \preceq f \) and \( t, f \models \varphi \)
Local Temporal Logic

\[ \varphi ::= a \mid \text{EX}\varphi \mid \text{EX}\_p\varphi \mid \varphi \_1 \cup \varphi \_2 \mid \varphi \_1 \cup_p \varphi \_2 \mid \neg \varphi \mid \varphi \_1 \lor \varphi \_2 \]

\[ a \in \Sigma, p \in \text{Proc} \]

Semantics  

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

- \( t, e \models \text{EX}\varphi \) if there is \( f \in E \) such that \( e \prec f \) and \( t, f \models \varphi \)
Local Temporal Logic

\[ \phi ::= a \mid \text{EX}\phi \mid \text{EX}_p\phi \mid \phi_1 \text{U} \phi_2 \mid \phi_1 \text{U}_p\phi_2 \mid \neg\phi \mid \phi_1 \lor \phi_2 \]

\(a \in \Sigma, p \in \text{Proc}\)

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

- \(t, e \models \text{EX}\phi\) if there is \(f \in E\) such that \(e \prec f\) and \(t, f \models \phi\)

- \(t, e \models \text{EX}_p\phi\) if there is \(f \in E\) such that \(e \rightarrow_p f\) and \(t, f \models \phi\)
Local Temporal Logic

\[ \varphi ::= a | \text{EX} \varphi | \text{EX}_p \varphi | \varphi_1 \cup \varphi_2 | \varphi_1 \cup_p \varphi_2 | \neg \varphi | \varphi_1 \lor \varphi_2 \]

\[ a \in \Sigma, p \in \text{Proc} \]

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

- \( t, e \models \text{EX} \varphi \) if there is \( f \in E \) such that \( e \prec f \) and \( t, f \models \varphi \)

- \( t, e \models \text{EX}_p \varphi \) if there is \( f \in E \) such that \( e \rightarrow_p f \) and \( t, f \models \varphi \)
Temporal Logic

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

- $t, e \models \text{EX}_p \varphi$ 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 $t = (E, (\rightarrow_p)_{p \in \text{Proc}}, \lambda)$ and $e \in E$)

- $t, e \models \exists_p \varphi$ 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 $t = (E, (\rightarrow_p)_{p \in \text{Proc}}, \lambda)$ and $e \in E$)

- $t, e \models \overline{\text{EX}_p \varphi}$ 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. $\leq$

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

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

- $t, e \models \mathbf{EX}_p\varphi$ 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. $\leq$

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

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

- $t, e \models \mathbf{EX}_p \varphi$ 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. $\leq$

- $t, e \models \varphi \mathbf{U} \psi$ if there is $f \in E$ such that $e \leq f$ and $t, f \models \psi$ 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!

<table>
<thead>
<tr>
<th>Semantics</th>
<th>(wrt. trace $t = (E, (\rightarrow_p)_{p \in \text{Proc}}, \lambda)$ and $e \in E$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t, e \models \text{EX}\varphi$</td>
<td>if there is $f \in E$ such that $e \preceq f$ and $t, f \models \varphi$</td>
</tr>
<tr>
<td>$t, e \models \text{EX}_p\varphi$</td>
<td>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. $\leq$</td>
</tr>
<tr>
<td>$t, e \models \varphi \text{U} \psi$</td>
<td>if there is $f \in E$ such that $t, f \models \psi$ and $t, e' \models \varphi$ for all $e' \in E$ with $e \leq e' &lt; f$</td>
</tr>
</tbody>
</table>
Observation (Gastin-Kuske '03)
All these modalities are MSO-definable!

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

- $t, e \models \text{EX}\varphi$ if there is $f \in E$ such that $e \prec f$ and $t, f \models \varphi$
- $t, e \models \text{EX}_p\varphi$ 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. $\unlhd$
- $t, e \models \varphi \cup \psi$ 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

- $\text{MSO}^{\text{EX}}(x, Y) = \exists y (y \in Y \land x \prec y)$
Temporal Logic

Observation (Gastin-Kuske ’03)
All these modalities are MSO-definable!

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

- \( t, e \models \text{EX} \varphi \) if there is \( f \in E \) such that \( e \preceq f \) and \( t, f \models \varphi \)
- \( t, e \models \text{EX}_p \varphi \) 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. \( \leq \)
- \( t, e \models \varphi \text{ U } \psi \) 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

- \( \text{MSO}^{\text{EX}}(x, Y) = \exists y \ (y \in Y \land x \preceq y) \)
- \( \text{MSO}^{\text{U}}(x, X, Y) = \exists y \ (y \in Y \land x \leq y \land \forall x'(x \leq x' < y \rightarrow x' \in X)) \)
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.
Theorem (Gastin-Kuske ’03)

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

Finite-State Shared-Memory Systems
5. Recursive Shared-Memory Systems
Recursive Shared-Memory Systems

- LTL
- MSO logic
- multi-pushdown automata
- asynchronous multi-pushdown automata
- nested traces

Realizability:
\[ \exists A : L(\varphi) = L(A) ? \]

Model Checking:
\[ L(\varphi) \supseteq L(A) ? \]
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 Multi-Pushdown Automata

\( \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{a_1, b_1, c\} \quad \Sigma_2 = \{a_2, b_2, c\} \quad \Sigma_{\text{call}} = \{a_1, a_2\} \quad \Sigma_{\text{ret}} = \{b_1, b_2\} \)
Asynchronous Multi-Pushdown Automata

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

Asynchronous MPA

Nested Trace
Asynchronous Multi-Pushdown Automata

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

Asynchronous MPA

Nested Trace

Recursive Shared-Memory Systems
Asynchronous Multi-Pushdown Automata

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

Asynchronous MPA

Nested Trace
Asynchronous Multi-Pushdown Automata

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

Asynchronous MPA

Nested Trace
Asynchronous Multi-Pushdown Automata

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

**Asynchronous MPA**

**Nested Trace**
Asynchronous Multi-Pushdown Automata

Proc = \{1, 2\} \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

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

Asynchronous MPA

Nested Trace

Recursive Shared-Memory Systems
Asynchronous Multi-Pushdown Automata

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

Asynchronous MPA

Nested Trace
Asynchronous Multi-Pushdown Automata

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

Asynchronous MPA

Nested Trace

Recursive Shared-Memory Systems
Asynchronous Multi-Pushdown Automata

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

Asynchronous MPA

Nested Trace
Asynchronous Multi-Pushdown Automata

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

Asynchronous MPA

Nested Trace
Asynchronous Multi-Pushdown Automata

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

Asynchronous MPA

Nested Trace

Recursive Shared-Memory Systems
Asynchronous Multi-Pushdown Automata

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

Asynchronous MPA

Nested Trace

Recursive Shared-Memory Systems
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

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

Asynchronous MPA

Nested Trace
Asynchronous Multi-Pushdown Automata

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

Asynchronous MPA

Nested Trace
Asynchronous Multi-Pushdown Automata

Proc = \{1, 2\} \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

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

Asynchronous MPA

\[
\begin{array}{c}
\text{Stack 1} \\
\text{Stack 2}
\end{array}
\]

Nested Trace

\[ t = (E, \rightarrow_1, \rightarrow_2, \leftarrow_1, \leftarrow_2, \lambda) \]
Recursive Shared-Memory Systems

LTL
MSO logic
multi-pushdown automata
nested traces

realizability
∃A: L(\varphi) = L(A) ?

model checking
L(\varphi) \supseteq L(A) ?

nonemptiness
L(A) \neq \emptyset ?

L(\varphi)

L(A)
Theorem

Emptiness for (asynchronous) MPA is undecidable.
Theorem

Emptiness for (asynchronous) MPA is undecidable.
Theorem

**Bounded nonemptiness, satisfiability, model checking, and realizability are decidable.**
Nested Traces and Their Linearizations

Nested Trace \( t = (E, \rightarrow_1, \rightarrow_2, \sim_1, \sim_2, \lambda) \)
Nested Traces and Their Linearizations

Nested Trace
\[ t = (E, \rightarrow_1, \rightarrow_2, \bowtie_1, \bowtie_2, \lambda) \]

Linearizations
\[ w \in \text{Lin}(t) \leadsto \text{trace}(w) = t \]
Nested Traces and Their Linearizations

Nested Trace

\[ t = (E, \rightarrow_1, \rightarrow_2, \bowtie_1, \bowtie_2, \lambda) \]

\[
\begin{array}{c}
\begin{array}{c}
\text{a}_1 \\
\downarrow 1 \\
\text{c} \\
\downarrow 2 \\
\text{a}_2 \\
\end{array} \\
\begin{array}{c}
\text{a}_1 \\
\downarrow 1 \\
\text{c} \\
\downarrow 2 \\
\text{a}_2 \\
\end{array} \\
\begin{array}{c}
\text{b}_1 \\
\downarrow 1 \\
\text{c} \\
\downarrow 2 \\
\text{b}_2 \\
\end{array}
\end{array}
\]

Linearizations

\[ w \in Lin(t) \iff \text{trace}(w) = t \]
Bounded Nested Words

Definition

- In a **context**, only one process **modifies** 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.
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.

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

LTL

MSO logic

multi-pushdown automata

nested traces

\[ L(\varphi) \]

realizability

\[ \exists A : L(\varphi) = L(A) \]?

model checking

\[ L(\varphi) \supseteq L(A) \]?

nonemptiness

\[ L(A) \neq \emptyset \]?

LTL

MSO logic

multi-pushdown automata

\[ L(A) \]

\[ L(\varphi) \supseteq L(A) \]?
Recursive Shared-Memory Systems

- Nested traces
  - Context bounded
  - Phase bounded
  - Scope bounded
  - Ordered

- $L(\varphi)$
- $L(A)$

- Realizability
  - $\exists A: L(\varphi) = L(A)$?
- Model checking
  - $L(\varphi) \supseteq L(A)$?

- LTL
- MSO logic
- Multi-pushdown automata

- Nonemptiness
  - $L(A) \neq \emptyset$?
Recursive Shared-Memory Systems

**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]

\[ L(\varphi) \supseteq L(A) \]

model checking

\[ L(A) \neq \emptyset \]

nonemptiness
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]

Proof for phases: binary-tree encoding
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]

Proof for phases: binary-tree encoding

nonemptiness
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]

Proof for phases: binary-tree encoding

\[ \exists A : L(\varphi) = L(A) ? \]
Recursive Shared-Memory Systems

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]

Proof for phases: binary-tree encoding
Recursive Shared-Memory Systems

nested traces

L(B)

nested words

L(A)
Recursive Shared-Memory Systems

nested traces

nested words

$L(B)$

realizability

$\exists A: L(A) = \text{trace}(L(B))$?
Theorem (B.-Grindei-Habermehl ’09)

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

It is undecidable if the language of a sequential MPA is ≃-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 $\theta$-nested words is a $\theta$-representation if, for all $\theta$-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 $\theta$-nested words is a $\theta$-representation if, for all $\theta$-nested words $w, w'$ with $w \sim_0 w'$, we have $w \in L$ iff $w' \in L$.

**2-phase representation**

![Diagram of 2-phase representation](image-url)
Representations

Let \( \theta \in \{ k\text{-context, } k\text{-scope, } k\text{-phase, ordered } | \ k \in \mathbb{N} \} \).

**Definition**

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

2-phase representation

A diagram illustrating the 2-phase representation is shown.

Recursive Shared-Memory Systems
Recursive Shared-Memory Systems

θ-nested traces

θ-nested words

$L_\theta(B)$

$L(A)$
Recursive Shared-Memory Systems

θ-nested traces
θ-nested words

$L_\theta(B)$

realizability

∃A: $L(A) = \text{trace}(L_\theta(B))$?
Theorem (B.-Grindei-Habermehl ’09)

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

Recursive Shared-Memory Systems

$\theta$-nested traces

$\theta$-nested words

$L_\theta(\mathcal{B})$

$L(\mathcal{A})$

realizability

$\exists \mathcal{A}: L(\mathcal{A}) = \text{trace}(L_\theta(\mathcal{B}))$?
Theorem

For a sequential MPA $B$ it is decidable if $L_{\theta}(B)$ is a $\theta$-representation (in elementary time).
Monadic Second-Order Logic (MSO)

- $x \rightarrow_p y$  
  $x$ and $y$ are successive events on process $p \in \text{Proc}$

- $x \bowtie_p y$  
  $x$ and $y$ form a call-return pair of process $p \in \text{Proc}$

- $a(x)$  
  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 \( p \in \text{Proc} \)
- \( x \rightleftharpoons_p y \): \( x \) and \( y \) form a call-return pair of process \( p \in \text{Proc} \)
- \( a(x) \): event \( x \) is labeled with \( a \in \Sigma \)

Example

\[ \models \exists x \exists y \exists z \left( x \rightleftharpoons_1 y \land a_2(z) \land x \leq z \leq y \right) \]

where \( \leq = (\rightarrow_1 \cup \rightarrow_2)^* \)
Monadic Second-Order Logic (MSO)

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

Example

\[
\models \exists x \exists y \exists z \left( x \leftrightarrow_1 y \land a_2(z) \land x \leq z \leq y \right)
\]

where $\leq = (\rightarrow_1 \cup \rightarrow_2)^*$
Recursive Shared-Memory Systems

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

MSO logic and asynchronous MPA are expressively equivalent wrt. \( \theta \)-nested traces.

\[ \exists A : L_\theta(\varphi) = L_\theta(A) ? \]

\[ L_\theta(\varphi) \Rightarrow \text{realizability} \Rightarrow L_\theta(A) \]

MSO

\[ \exists x \exists y \exists z (x \rightarrow_1 y \land a_2(z) \land x_1 \leq z \leq y_1) \]
Theorem (La Torre-Madhusudan-Parlato ’07-’13)

MSO logic and asynchronous MPA are expressively equivalent wrt. \( \theta \)-nested traces.
Theorem (La Torre-Madhusudan-Parlato ’07-’13)

MSO logic and asynchronous MPA are expressively equivalent wrt. \( \theta \)-nested traces.

\[ \exists \mathcal{A}: L_\theta(\varphi) = L_\theta(\mathcal{A})? \]

\[ L_\theta(\varphi) \supseteq L_\theta(\mathcal{A})? \]

\[ \Rightarrow \text{MSO model checking is decidable.} \]
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 \( \varphi U^a_p \psi \)

\[
\text{MSO}^{U^a_p}(x, X_1, X_2) = \\
\exists Y \exists x' \left(x' \in X_2 \land Y \subseteq X_1 \land \right. \\
\left. \forall z (z \in Y \lor z = x') \rightarrow (z = x \lor \exists y \ (y \in Y \land \varphi_p(y, z))) \right)
\]

where \( \varphi_p(y, z) = y \bowtie_p z \lor (\neg \exists z' y \bowtie_p z' \land \neg \exists y' (y' \bowtie_p z \land y \rightarrow_p z)) \).
Model Checking ($\theta = \text{"k-phase bounded"}$)
Theorem (B.-Cyriac-Gastin-Zeitoun '11)

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

\[
L_\theta(\varphi) \supseteq L_\theta(A) ?
\]
Model Checking ($\theta = \text{"k-phase bounded"}$)

**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

- Single process
- Shared memory
- Message passing/broadcasting

- Static & known
- Static & unknown (parameterized)
- Dynamic

- Finite-state
- Recursive
- Timed
Message-Passing Systems

- Message sequence charts (MSCs)
- $L(\varphi)$
- $L(A)$
- Realizability: $\exists A: L(\varphi) = L(A)$?
- Model checking: $L(\varphi) \supseteq L(A)$?

- PDL
- MSO logic
- Finite automata

- Communicating automata
Communicating Automata and MSCs

$$\text{Proc} = \{1, 2\}$$
Communicating Automata and MSCs

\( \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1!2, 1?2\} \quad \Sigma_2 = \{2!1, 2?1\} \)
Communicating Automata and MSCs

\[\text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1!2, 1?2\} \quad \Sigma_2 = \{2!1, 2?1\}\]

Communicating Automaton

\[\begin{array}{c}
1!_a2 \quad 1!_a2 \\
\quad \quad 1?_b2 \\
\quad \quad 2!_b1 \\
\quad \quad 2?_a1
\end{array}\]
Communicating Automata and MSCs

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1!2, 1?2\} \quad \Sigma_2 = \{2!1, 2?1\} \]

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

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

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

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

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

\[\text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1!2, 1?2\} \quad \Sigma_2 = \{2!1, 2?1\}\]

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1!2, 1?2\} \quad \Sigma_2 = \{2!1, 2?1\} \]

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1!2, 1?2\} \quad \Sigma_2 = \{2!1, 2?1\} \]

**Communicating Automaton**

**Message Sequence Chart (MSC)**
Communicating Automata and MSCs

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1!2, 1?2\} \quad \Sigma_2 = \{2!1, 2?1\} \]

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1!2, 1?2\} \quad \Sigma_2 = \{2!1, 2?1\} \]

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

\( \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1!2, 1?2\} \quad \Sigma_2 = \{2!1, 2?1\} \)

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

\[\text{Proc} = \{1, 2\}\]

\[\Sigma_1 = \{1!2, 1?2\}\]

\[\Sigma_2 = \{2!1, 2?1\}\]

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

\( \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1!2, 1?2\} \quad \Sigma_2 = \{2!1, 2?1\} \)

**Communicating Automaton**

![Communicating Automaton Diagram]

**Message Sequence Chart (MSC)**

![Message Sequence Chart Diagram]
Communicating Automata and MSCs

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1\#2, 1?2\} \quad \Sigma_2 = \{2\#1, 2?1\} \]

### Communicating Automaton

- \(1\#a\) \(\rightarrow\) \(1\#a\)
- \(1?b\) \(\rightarrow\) \(1\#a\)
- \(2\#a\) \(\rightarrow\) \(2\#a\)
- \(2?b\) \(\rightarrow\) \(2\#a\)

### Message Sequence Chart (MSC)

\[ M = (E, \rightarrow_1, \rightarrow_2, \text{msg} \rightarrow, \lambda) \]
Communicating Automata and MSCs

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

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

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

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

\[ \text{Proc} = \{1, 2\} \quad \Sigma_1 = \{1!2, 1?2\} \quad \Sigma_2 = \{2!1, 2?1\} \]

Communicating Automaton

Message Sequence Chart (MSC)
Communicating Automata and MSCs

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

Communicating Automaton

Message Sequence Chart (MSC) \quad M = (E, \rightarrow_1, \rightarrow_2, \rightarrow^{\text{msg}}, \lambda)
Message-Passing Systems

MSCs

$L(\varphi)$

realizability

$\exists A: L(\varphi) = L(A)$ ?

model checking

$L(\varphi) \supseteq L(A)$ ?

nonemptiness

$L(A) \neq \emptyset$ ?

PDL

MSO logic

finite automata
Theorem

Emptiness for CA is undecidable.
Theorem

Emptiness for CA is undecidable.
Theorem

**Bounded nonemptiness, satisfiability, model checking, and realizability are decidable.**
Channel-Bounded MSCs

MSC $M$

1-bounded linearization

$w \in \text{Lin}(M) \subseteq \Sigma^* \Rightarrow \text{msc}(w) = M$

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.
Channel-Bounded MSCs

MSC $M$

3-bounded linearization $w \in \text{Lin}(M) \subseteq \Sigma^* \leadsto \text{msc}(w) = M$
Channel-Bounded MSCs

MSC $M$

1-bounded linearization $w \in Lin(M) \subseteq \Sigma^* \leadsto msc(w) = M$
Channel-Bounded MSCs

MSC $M$

1-bounded linearization $w \in \text{Lin}(M) \subseteq \Sigma^* \leadsto \text{msc}(w) = M$

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.

Example

- $\left( \begin{array}{c} 1!2 \\ 2?1 \end{array} \right)^*$ is an $\exists 1$-representation, but no $\forall$-representation.
- $\left( \begin{array}{c} 1!2 \\ 2?1 \\ 3!4 \\ 4?3 \end{array} \right)^*$ is not an $\exists B$-representation, for any $B$. 
## 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

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

- **∀-representation** if, for all MSCs $M$, $L$ contains either
  - all linearizations of $M$, or
  - none of its linearizations.

Example

\[(1!2\ 2?1)^*\] is an **∃1-representation**, but no **∀-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**

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

$(1!2\ 2?1\ 3!4\ 4?3)^* \text{ is not an } \exists B\text{-representation, for any } B.
Message-Passing Systems

∀B-bounded MSCs

B-bounded words

L(B)

L(A)

(1!2 2?1 2!1 1?2)^*
∀B-bounded MSCs

B-bounded words

L(\mathcal{B})

L(\mathcal{A})

∀B-bounded MSCs

B-bounded words

( ) ∗

1!2 2?1 2!1 1?2

1!a2 1!a2

1?b2

2?a1

2!b1

2?a1

L(\mathcal{B})

L(\mathcal{A})

realizability

∃A: L(\mathcal{A}) = msc(L(\mathcal{B})) ?
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 $L(\mathcal{A}) = msc(L(\mathcal{B}))$. 

∀B-bounded MSCs
B-bounded words
( ) ∗
1!2 2?1 2!1 1?2
1!a2 1!a2
1?b2
2?a1
2!b1
2?a1
L(B)
L(A)
realizability
∃A: L(A) = msc(L(B))?

Message-Passing Systems
Theorem (Henriksen et al. ’00)

For a finite automaton $B$ it is decidable if $L(B)$ is a $\forall$-representation.
Theorem (Genest-Kuske-Muscholl ’06)

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

\[
\begin{align*}
\exists B\text{-bounded MSCs} & \quad B\text{-bounded words} \\
L(B) & \quad L(A) \\
\exists A: L(A) = msc(L(B)) & ? \quad \text{realizability}
\end{align*}
\]
Theorem

For a finite automaton $B$ it is decidable if $L(B)$ is an $\exists B$-representation.
### Monadic Second-Order Logic (MSO)

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

Example

$$\models \exists x, y, x', y' (x \rightarrow^{\text{msg}} y \land x' \rightarrow^{\text{msg}} y' \land x \rightarrow_1^* y' \land x' \rightarrow_2^* y)$$
Theorem (B.-Leucker ’04)

EMSO logic ($\exists X_1 \ldots X_n \varphi$ with $\varphi$ first-order) and communicating automata are expressively equivalent. MSO logic is strictly more expressive.
Theorem (Genest-Kuske-Muscholl ’04)

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

- There is an MSO sentence $\varphi$ such that $L = L(\varphi)$.
- There is a CA $A$ such that $L = L(A)$. 
Theorem (Genest-Kuske-Muscholl ’04)

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

Message-Passing Systems
Theorem (B., Kuske, Meinecke 2007; Mennicke 2012)

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

Message-Passing Systems
Message-Passing Systems

PDL
MSO logic
finite automata

MSCs

$L(\varphi)$

realizability
$\exists A: L(\varphi) = L(A)$ ?

model checking
$L(\varphi) \supseteq L(A)$ ?

L(A) ≠ \emptyset ?

nonemptiness
Message-Passing Systems

- PDL
- MSO logic
- finite automata
- MSCs
- lossy MSCs

\[ L(\varphi) \]

realizability

\[ \exists A : L(\varphi) = L(A) ? \]

model checking

\[ L(\varphi) \supseteq L(A) ? \]

nonemptiness

\[ L(A) \not= \emptyset ? \]
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

single process
shared memory
message passing/broadcasting

static & known
static & unknown
(dynamic)

finite-state
recursive
timed
Conclusion: Recursive Shared-Memory Systems

Realizability

Model Checking

Conclusion and Perspectives
Conclusion: Message-Passing Systems

- Realizability
- Model Checking

Diagram showing different system types:
- Single process
- Shared memory
- Message passing/broadcasting
- Static & known
- Static & unknown (parameterized)
- Dynamic
- Finite-state
- Recursive
- Timed
Realizability
Model Checking
Perspectives: Parameterized Systems

Realizability
Model Checking
Reachability

Conclusion and Perspectives