Overview

Introduction

**Modelling parallel systems**
- Transition systems
- Modeling hard- and software systems
- Parallelism and communication

Linear Time Properties
Regular Properties
Linear Temporal Logic
Computation-Tree Logic
Equivalences and Abstraction
representation of data-dependent parallel systems with

- communication over shared variables
- synchronous message passing
- asynchronous message passing
Channel systems

representation of data-dependent parallel systems with

- communication over shared variables
- synchronous message passing
- asynchronous message passing

} communication over channels
representation of data-dependent parallel systems with

- communication over **shared variables**
- synchronous message passing
- asynchronous message passing

\[
\begin{align*}
\mathcal{P}_3 & \quad c_3 & \quad \mathcal{P}_1 \\
\mathcal{P}_2 & \quad c_2 & \quad c_1 & \quad \mathcal{P}_4 \\
\end{align*}
\]

\text{shared variables}
Channel systems

representation of data-dependent parallel systems with

- communication over **shared variables**
- **synchronous** message passing
- **asynchronous** message passing

\[ \begin{array}{c}
\mathcal{P}_1 \\
\mathcal{P}_2 \\
\mathcal{P}_3 \\
\mathcal{P}_4 \\
\end{array} \]

channel types: synchronous or FIFO
representation of data-dependent parallel systems with
- communication over **shared variables**
- **synchronous** message passing
- **asynchronous** message passing

channel types: synchronous or FIFO

capacity \( \equiv \) number of buffer cells
Channel systems

representation of data-dependent parallel systems with

- communication over **shared variables**
- **synchronous** message passing $\iff$ capacity $0$
- **asynchronous** message passing $\iff$ capacity $\geq 1$

channel types: synchronous or **FIFO**

capacity $0$ $\iff$ number of buffer cells
representation of data-dependent parallel systems with

- communication over shared variables
- synchronous message passing
- asynchronous message passing

formalization through program graphs for $P_1, \ldots, P_n$
Channel systems

representation of data-dependent parallel systems with

• communication over shared variables
• synchronous message passing
• asynchronous message passing

communication over channels

formalization through program graphs for \( P_1, \ldots, P_n \)

• with conditional transitions \( \ell_i \xleftarrow{g:\alpha} \ell'_i \) (as before)
Channel systems

representation of data-dependent parallel systems with
• communication over shared variables
• synchronous message passing
• asynchronous message passing
}

communication over channels

formalization through program graphs for $P_1, ..., P_n$

• with conditional transitions $l_i \xleftarrow{g: \alpha} l'_i$ (as before)
• and communication actions

$\begin{align*}
l_i & \xleftarrow{c!\nu} l'_i \quad \text{sending value } \nu \text{ via channel } c \\
l_i & \xleftarrow{c?x} l'_i \quad \text{receiving a value for variable } x \text{ via channel } c
\end{align*}$
Typed variable: variable $x$ with data domain $\text{Dom}(x)$
Typed variables and channels

**typed variable:** variable $x$ with data domain $\text{Dom}(x)$

evaluation for a set $\text{Var}$ of typed variables:

- type-consistent function $\eta : \text{Var} \rightarrow \text{Values}$
  
  i.e., $\eta(x) \in \text{Dom}(x)$
Typed variables and channels

*typed variable*: variable $x$ with data domain $\text{Dom}(x)$

evaluation for a set $\text{Var}$ of typed variables:

- type-consistent function $\eta : \text{Var} \rightarrow \text{Values}$

  i.e., $\eta(x) \in \text{Dom}(x)$

*typed channel*: channel $c$ with capacity $\text{cap}(c) \in \mathbb{N} \cup \{\infty\}$ and domain $\text{Dom}(c)$
Typed variables and channels

**typed variable:** variable \( x \) with data domain \( \text{Dom}(x) \)

evaluation for a set \( \text{Var} \) of typed variables:

- type-consistent function \( \eta : \text{Var} \to \text{Values} \)

\[ \uparrow \]

i.e., \( \eta(x) \in \text{Dom}(x) \)

**typed channel:** channel \( c \) with

- capacity \( \text{cap}(c) \in \mathbb{N} \cup \{\infty\} \)
- and domain \( \text{Dom}(c) \)

evaluation for a set \( \text{Chan} \) of typed channels:

- type-consistent function \( \xi : \text{Chan} \to \text{Values}^* \)
**Typed variables and channels**

**typed variable:** variable \( x \) with data domain \( \text{Dom}(x) \)

evaluation for a set \( \text{Var} \) of typed variables:

- type-consistent function \( \eta : \text{Var} \rightarrow \text{Values} \)

\( \uparrow \)

i.e., \( \eta(x) \in \text{Dom}(x) \)

**typed channel:** channel \( c \) with

- capacity \( \text{cap}(c) \in \mathbb{N} \cup \{\infty\} \)
- domain \( \text{Dom}(c) \)

evaluation for a set \( \text{Chan} \) of typed channels:

- type-consistent function \( \xi : \text{Chan} \rightarrow \text{Values}^* \)

s.t. \( \xi(c) \) is a word over \( \text{Dom}(c) \) of length \( \leq \text{cap}(c) \)
Channel system (CS)

\[
\begin{bmatrix}
\mathcal{P}_1 & \mathcal{P}_2 & \ldots & \mathcal{P}_n
\end{bmatrix}
\]

where \( \mathcal{P}_i \) are program graphs
Channel system (CS)

\[
\left[ \mathcal{P}_1 | \mathcal{P}_2 | \ldots | \mathcal{P}_n \right] \quad \text{where} \quad \mathcal{P}_i \quad \text{are program graphs over a pair} \ (\text{Var}, \text{Chan})
\]
Channel system (CS)

\[
\mathcal{P}_1 | \mathcal{P}_2 | \ldots | \mathcal{P}_n
\]

where \( \mathcal{P}_i \) are program graphs over a pair \((Var, Chan)\)

- **Var** set of typed variables
- **Chan** set of typed channels with capacities \( \text{cap}(\cdot) \) and domains \( \text{Dom}(\cdot) \)
Channel system (CS)

\[
\left[ \mathcal{P}_1 | \mathcal{P}_2 | \ldots | \mathcal{P}_n \right] \text{ where } \mathcal{P}_i \text{ are program graphs over a pair } (\text{Var}, \text{Chan})
\]

- **Var** set of typed variables
- **Chan** set of typed channels with capacities \(\text{cap}(\cdot)\) and domains \(\text{Dom}(\cdot)\)

Program graphs \(\mathcal{P}_i = (\text{Loc}_i, \text{Act}_i, \text{Effect}_i, \leftarrow_i, \text{Loc}_{0,i}, g_0)\) with conditional transitions

\[
\ell \xleftarrow{g:\alpha} \ell_i \xrightarrow{\cdot} \ell' \text{ guarded command}
\]
Channel system (CS)

\[
\begin{bmatrix}
\mathcal{P}_1 | \mathcal{P}_2 | \ldots | \mathcal{P}_n
\end{bmatrix}
\]

where \( \mathcal{P}_i \) are program graphs over a pair \((\text{Var}, \text{Chan})\)

\textbf{Var} set of typed variables
\textbf{Chan} set of typed channels with capacities \( \text{cap}(\cdot) \) and domains \( \text{Dom}(\cdot) \)

Program graphs \( \mathcal{P}_i = (\text{Loc}_i, \text{Act}_i, \text{Effect}_i, \leftarrow_i, \text{Loc}_{0,i}, g_0) \)

with conditional transitions

\[
\ell \xleftarrow{g: \alpha} i \ell'
\]

where \( g \in \text{Cond}(\text{Var}), \alpha \in \text{Act}_i \)
Channel system (CS)

\[
\begin{bmatrix}
\mathcal{P}_1 \mid \mathcal{P}_2 \mid \ldots \mid \mathcal{P}_n
\end{bmatrix}
\]

where \( \mathcal{P}_i \) are program graphs

over a pair \((\text{Var}, \text{Chan})\)

\text{Var} \quad \text{set of typed variables}

\text{Chan} \quad \text{set of typed channels with}

capacities \( \text{cap}(\cdot) \) and domains \( \text{Dom}(\cdot) \)

program graphs \( \mathcal{P}_i = (\text{Loc}_i, \text{Act}_i, \text{Effect}_i, \leftarrow_i, \text{Loc}_{0,i}, g_0) \)

with conditional transitions

\( \ell \xleftarrow{g:\alpha} \ell' \) \quad \text{guarded command}

\( \ell \xleftarrow{c!v} \ell' \) \quad \text{sending value } v \text{ via channel } c
Channel system (CS)

\[
\begin{bmatrix}
\mathcal{P}_1 & \mathcal{P}_2 & \ldots & \mathcal{P}_n
\end{bmatrix}
\]

where \( \mathcal{P}_i \) are program graphs over a pair \((\text{Var}, \text{Chan})\)

- \( \text{Var} \): set of typed variables
- \( \text{Chan} \): set of typed channels with capacities \( \text{cap}(\cdot) \) and domains \( \text{Dom}(\cdot) \)

Program graphs \( \mathcal{P}_i = (\text{Loc}_i, \text{Act}_i, \text{Effect}_i, \leftarrow_i, \text{Loc}_{0,i}, g_0) \)

with conditional transitions

- Guarded command:
  \( \ell \xleftarrow{\mathbf{g}:\alpha} \rightarrow_i \ell' \)

- Sending value via channel:
  \( \ell \xleftarrow{\mathbf{c!v}} \rightarrow_i \ell' \)

- Receiving a value for variable via channel:
  \( \ell \xleftarrow{\mathbf{c?x}} \rightarrow_i \ell' \)
asynchronous message passing via channels of capacity $\geq 1$

<table>
<thead>
<tr>
<th></th>
<th>enabled if ...</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>sending $c!v$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receiving $c?x$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**asynchronous message passing** via channels of capacity \( \geq 1 \)

<table>
<thead>
<tr>
<th>Sending</th>
<th>Receiving</th>
<th>enabled if ...</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c!v)</td>
<td>(c?x)</td>
<td>channel (c) not full</td>
<td>(\text{add}(c, v))</td>
</tr>
</tbody>
</table>

\[\begin{array}{c|c|c|c|c|c}
  & v_1 & \ldots & v_r & c!v & \\
\hline
\end{array}\]

\[\begin{array}{c|c|c|c|c|c}
  & v_1 & \ldots & v_r & v & \\
\hline
\end{array}\]
**Effect of communication actions in CS**

*asynchronous message passing* via channels of capacity \( \geq 1 \)

<table>
<thead>
<tr>
<th>Sending</th>
<th>Receiving</th>
<th>enabled if ...</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c!v )</td>
<td>( c?v )</td>
<td>channel ( c ) not full</td>
<td>( \text{add}(c, v) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{channel } c ) not empty</td>
<td>( x := v )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( v = \text{front}(c) )</td>
<td>( \text{remove}(c) )</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c|c|c}
\hline
\& v_1 & \cdots & v_r & c!v \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
\hline
\& v_1 & \cdots & v_r & v \\
\hline
\end{array}
\]
Effect of communication actions in CS

**asynchronous message passing** via channels of capacity \( \geq 1 \)

<table>
<thead>
<tr>
<th>Sending ( c!v )</th>
<th>enabled if ...</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel ( c ) not full</td>
<td>( \text{add}(c, v) )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Receiving ( c?x )</th>
<th>enabled if ...</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel ( c ) not empty ( v = \text{front}(c) )</td>
<td>( x := v ) ( \text{remove}(c) )</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c|c|c|c}
\hline
v_1 & \ldots & v_r & c!v \\
\hline
\hline
v & v_2 & \ldots & v_r & c?x \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
\hline
v_1 & \ldots & v_r & v \\
\hline
\hline
v_2 & \ldots & v_r & x = v \\
\hline
\end{array}
\]

27 / 131
**Effect of communication actions in CS**

asynchronous message passing via channels of capacity $\geq 1$

<table>
<thead>
<tr>
<th>Sending</th>
<th>Receiving</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c!v$</td>
<td>$c?x$</td>
<td></td>
</tr>
<tr>
<td>enabled if ...</td>
<td>channel $c$ not full</td>
<td>$\text{add}(c, v)$</td>
</tr>
<tr>
<td></td>
<td>channel $c$ not empty</td>
<td>$x := v$ remove($c$)</td>
</tr>
</tbody>
</table>

synchronous message passing via channels of capacity 0

- $c!v$ and $c?x$ are executed at the same time
- effect $x := v$
channel system over \((\text{Var}, \text{Chan})\)

\[ C = [P_1 \mid \ldots \mid P_n] \]

transition system \(\mathcal{T}_C\)
TS-semantics of channel systems

channel system over \((\text{Var}, \text{Chan})\)

\[\mathcal{C} = [\mathcal{P}_1 \mid \ldots \mid \mathcal{P}_n]\]

transition system \(\mathcal{T}_\mathcal{C}\)

states of \(\mathcal{T}_\mathcal{C}\) have the form

\[\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle\]

locations of \(\mathcal{P}_1, \ldots, \mathcal{P}_n\)

channel evaluation

variable valuation
states $\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle$ where

- $\ell_i$ location of program graph $\mathcal{P}_i$, 
- $\eta \in \text{Eval}(\text{Var})$ variable evaluation 
- $\xi \in \text{Eval}(\text{Chan})$ channel evaluation
states $\langle \ell_1, ..., \ell_n, \eta, \xi \rangle$ where

- $\ell_i$ location of program graph $P_i$,
- $\eta \in \text{Eval}(\text{Var})$ variable evaluation
- $\xi \in \text{Eval}(\text{Chan})$ channel evaluation

variable evaluation:

- $\eta : \text{Var} \rightarrow \bigcup_{x \in \text{Var}} \text{Dom}(x)$ with $\eta(x) \in \text{Dom}(x)$

channel evaluation:

- $\xi : \text{Chan} \rightarrow \bigcup_{c \in \text{Chan}} \text{Dom}(c)^*$ with $\xi(c) \in \text{Dom}(c)^*$
  and $|\xi(c)| \leq \text{cap}(c)$
TS-semantics of channel systems

states $\langle l_1, \ldots, l_n, \eta, \xi \rangle$ where

- $l_i$ location of program graph $P_i$,
- $\eta \in \text{Eval}(\text{Var})$ variable evaluation
- $\xi \in \text{Eval}(\text{Chan})$ channel evaluation

variable evaluation:

$$\eta : \text{Var} \rightarrow \bigcup_{x \in \text{Var}} \text{Dom}(x) \quad \text{with } \eta(x) \in \text{Dom}(x)$$

channel evaluation:

$$\xi : \text{Chan} \rightarrow \bigcup_{c \in \text{Chan}} \text{Dom}(c)^* \quad \text{with } \xi(c) \in \text{Dom}(c)^*$$

- only channels $c$ with $\text{cap}(c) \geq 1$ are relevant
Transition relation of channel systems

states $\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle$ where $\ell_i \in \text{Loc}_i$, $\eta \in \text{Eval(Var)}$, $\xi \in \text{Eval(Chan)}$
states $\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle$ where $\ell_i \in \text{Loc}_i$, $\eta \in \text{Eval(Var)}$, $\xi \in \text{Eval(Chan)}$

transition relation $\longrightarrow$ is given by SOS-rules:

- interleaving rules for $\alpha \in \text{Act}_i$
- rules for message passing along channels
Transitions relation of channel systems

states \( \langle \ell_1, ..., \ell_n, \eta, \xi \rangle \) where \( \ell_i \in \text{Loc}_i \), \( \eta \in \text{Eval}(\text{Var}) \), \( \xi \in \text{Eval}(\text{Chan}) \)

transition relation \( \longrightarrow \) is given by SOS-rules:

- interleaving rules for \( \alpha \in \text{Act}_i \)
- rules for message passing along channels

interleaving rule for actions \( \alpha \in \text{Act}_i \):

\[
\ell_i \xrightarrow{\alpha} \ell_i' \quad \land \quad \eta \models g \\
\langle \ell_1, .., \ell_i, ..., \ell_n, \eta, \xi \rangle \xrightarrow{\alpha} \langle \ell_1, .., \ell_i', ..., \ell_n, \text{Effect}_i(\alpha, \eta), \xi \rangle
\]
Transition relation of channel systems

states \( \langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle \) where \( \ell_i \in \text{Loc}_i, \eta \in \text{Eval(Var)}, \xi \in \text{Eval(Chan)} \)

transition relation \( \rightarrow \) is given by SOS-rules:

- interleaving rules for \( \alpha \in \text{Act}_i \)
- rules for message passing along channels

interleaving rule for actions \( \alpha \in \text{Act}_i \):

\[
\begin{array}{c}
\ell_i \xrightarrow{\text{g:} \alpha} \ell_i' \land \eta \models g \\
\hline
\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\alpha} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \text{Effect}_i(\alpha, \eta), \xi \rangle \\
\uparrow
\end{array}
\]

does not affect the channel evaluation \( \xi \)
for channel $c$ with $\text{cap}(c) \geq 1$
SOS-rules for asynchronous message passing

for channel \( c \) with \( \text{cap}(c) \geq 1 \)

receiving a message:

\[
\ell_i \xleftarrow{c?x} \_i \ell_i' \land \xi(c) = v_1v_2...v_k \land k \geq 1
\]

\[
\langle \ell_1, ..., \ell_i, ..., \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, ..., \ell_i', ..., \ell_n, \eta', \xi' \rangle
\]
SOS-rules for asynchronous message passing

for channel \( c \) with \( \text{cap}(c) \geq 1 \)

receiving a message:

\[
\ell_i \xleftrightarrow{c?x} \ell_i' \wedge \xi(c) = v_1 v_2 \ldots v_k \wedge k \geq 1
\]

\[
\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi' \rangle
\]

where \( \eta' = \eta[x:=v_1] \)

\[
\eta[x:=v_1](y) = \begin{cases} 
\eta(y) & \text{if } y \neq x \\
v_1 & \text{if } y = x 
\end{cases}
\]
SOS-rules for asynchronous message passing

for channel $c$ with \( \text{cap}(c) \geq 1 \)

receiving a message:

\[
\ell_i \xleftarrow{c?x} \ell'_{i} \land \xi(c) = v_1 v_2 \ldots v_k \land k \geq 1
\]

\[
\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell'_{i}, \ldots, \ell_n, \eta', \xi' \rangle
\]

where $\eta' = \eta[x:=v_1]$ and $\xi' = \xi[c:=v_2 \ldots v_k]$

\[
\eta[x:=v_1](y) = \begin{cases} 
\eta(y) & \text{if } y \neq x \\
v_1 & \text{if } y = x
\end{cases}
\]

\[
\xi[c:=v_2 \ldots v_k](d) = \begin{cases} 
\xi(d) & \text{if } d \neq c \\
v_2 \ldots v_k & \text{if } d = c
\end{cases}
\]
SOS-rules for asynchronous message passing

for channel $c$ with $\text{cap}(c) \geq 1$

receiving a message:

\[
\begin{align*}
\ell_i & \xrightarrow[c?x]{c} \ell_i' & \xi(c) &= v_1v_2...v_k & \land & k \geq 1 \\
\langle \ell_1, ..., \ell_i, ..., \ell_n, \eta, \xi \rangle & \xrightarrow{\tau} \langle \ell_1, ..., \ell_i', ..., \ell_n, \eta', \xi' \rangle
\end{align*}
\]

where $\eta' = \eta[x:=v_1]$ and $\xi' = \xi[c:=v_2...v_k]$  

sending a message:

\[
\begin{align*}
\ell_i & \xleftarrow[c!v]{c} \ell_i' & \xi(c) &= v_1...v_k & \land & k < \text{cap}(c) \\
\langle \ell_1, ..., \ell_i, ..., \ell_n, \eta, \xi \rangle & \xrightarrow{\tau} \langle \ell_1, ..., \ell_i', ..., \ell_n, \eta, \xi[c:=v_1...v_k v] \rangle
\end{align*}
\]
for synchronous channel \( c \):

\[
\ell_i \overset{c?x}{\rightarrow} i \ell_i' \land \ell_j \overset{c!v}{\rightarrow} j \ell_j' \land i \neq j
\]

\[
\langle \ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{T} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_j', \ldots, \ell_n, \eta', \xi' \rangle
\]
SOS-rules for synchronous message passing

for synchronous channel $c$:

\[
\begin{align*}
\ell_i & \xleftarrow{c?x} i \ell'_i \land \ell_j & \xleftarrow{c!v} j \ell'_j \land i \neq j \\
\langle \ell_1, ..., \ell_i, ..., \ell_j, ..., \ell_n, \eta, \xi \rangle & \xrightarrow{T} \langle \ell_1, ..., \ell'_i, ..., \ell'_j, ..., \ell_n, \eta', \xi' \rangle
\end{align*}
\]

where $\eta' = \eta[x:=v]$
for synchronous channel $c$:

\[
\ell_i \xleftarrow{c?x} i \ell'_i \land \ell_j \xrightarrow{c!v} j \ell'_j \land i \neq j
\]

\[
\langle \ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{T} \langle \ell_1, \ldots, \ell'_i, \ldots, \ell'_j, \ldots, \ell_n, \eta', \xi' \rangle
\]

where $\eta' = \eta[x := v]$ and $\xi' = \xi$
How many states...

has a transition system for a channel system with ... ?

- 2 processes with 2 locations each
- 2 Boolean variables
- 2 channels of capacity 10 and Boolean values
How many states... has a transition system for a channel system with ... ?

- 2 processes with 2 locations each
- 2 Boolean variables
- 2 channels of capacity 10 and Boolean values

**Answer:**

\[2 \times 2 \times 2 \times 2 \times (2^{11} - 1) \times (2^{11} - 1)\]

**Note:** \(2^{11} - 1 = 1 + 2 + 2^2 + \ldots + 2^{10}\)
How many states...

has a transition system for a channel system with ...

- 2 processes with 2 locations each
- 2 Boolean variables
- 2 channels of capacity 10 and Boolean values

**answer:**

\[ 2 \times 2 \times 2 \times 2 \times (2^{11} - 1) \times (2^{11} - 1) > 2^{24} > 25 \text{ mio} \]

**note:** \( 2^{11} - 1 = 1 + 2 + 2^2 + \ldots + 2^{10} \)
How many states...  

has a transition system for a channel system with ... ?

- 2 processes with 2 locations each
- 2 Boolean variables
- 2 channels of capacity 10 and Boolean values

**answer:**

\[
2 \times 2 \times 2 \times 2 \times (2^{11} - 1) \times (2^{11} - 1) > 2^{24} > 25 \text{ mio}
\]

**note:**  

\[
2^{11} - 1 = 1 + 2 + 2^2 + \ldots + 2^{10}
\]

... with an **unbounded channel**?
How many states...

has a transition system for a channel system with ...

- 2 processes with 2 locations each
- 2 Boolean variables
- 2 channels of capacity 10 and Boolean values

answer:

\[ 2 \times 2 \times 2 \times 2 \times (2^{11} - 1) \times (2^{11} - 1) > 2^{24} > 25 \text{ mio} \]

note: \( 2^{11} - 1 = 1 + 2 + 2^2 + \ldots + 2^{10} \)

... with an unbounded channel?

answer: \( \infty \)
Alternating bit protocol (ABP)

send acknowledgement

sender ➔ receiver

message
Alternating bit protocol (ABP)

sender

receiver

send acknowledgement

send message \( \pm \) bit \( y \)
via unreliable channel
Alternating bit protocol (ABP)

return the received bit $y$

sender

receiver

send message $\pm$ bit $y$
via unreliable channel
Alternating bit protocol (ABP)

return the received bit $y$

sender

receiver

send message \(+\) bit $y$
via unreliable channel

timer
Protocol for the sender

return the received bit $y$

LOOP FOREVER
(1) send message $+$ bit $y$ and activate timer
(2) AWAIT timeout or acknowledgement DO
    IF timeout THEN goto (1)
    ELSE turn off timer; $y := \neg y$
    FI
OD
Protocol for the sender

LOOP FOREVER
(1) send message ++bit y++ and activate timer
(2) AWAIT timeout or acknowledgement DO
    IF timeout THEN goto (1)
    ELSE turn off timer; \( y := \neg y \)
    FI
OD
If both channels are unreliable ...

acknowledgement bit \( x \)
via unreliable channel \( d \)

message + bit \( y \)
via unreliable channel \( c \)
If both channels are unreliable ...

acknowledgement bit $x$
via unreliable channel d

message + bit $y$
via unreliable channel c

LOOP FOREVER

(1) send message + bit $y$ and activate timer

(2) AWAIT timeout or acknowledgement $x$ DO

IF timeout THEN goto (1)
ELSE IF $x=y$ THEN turn off timer; $y:=\neg y$
ELSE ignore $x$

FI

FI

OD
Alternating bit protocol (ABP)

Channel system: \[
\begin{array}{c|c|c}
\text{Sender} & \text{Timer} & \text{Receiver} \\
\end{array}
\]
Alternating bit protocol (ABP)

channel system: \[
\text{Sender} \mid \text{Timer} \mid \text{Receiver}
\]

- synchronous message passing between \text{Timer} and \text{Sender}
- asynchronous message passing between \text{Receiver} and \text{Sender}
Alternating bit protocol (ABP)

Channel system: \[
\text{[Sender} \mid \text{Timer} \mid \text{Receiver}]\]

- Synchronous message passing between \textit{Timer} and \textit{Sender} ← channels \textit{e} and \textit{f}
- Asynchronous message passing between \textit{Receiver} and \textit{Sender} ← channels \textit{c}, \textit{d}
Alternating bit protocol (ABP)

Channel system: \[
\text{Sender} | \text{Timer} | \text{Receiver}
\]
Alternating bit protocol (ABP)

Channel system: \[ \text{Sender} \mid \text{Timer} \mid \text{Receiver} \]

Actions of \textit{Sender}:
\[
\begin{align*}
e!\text{timer\_on} \\
e!\text{timer\_off} \\
f?z' \\
\end{align*}
\]
specify the **sender** by a program graph using

- asynchronous channels **c** and **d**
- synchronous channels **e** and **f**
Program graph for the sender

specify the **sender** by a program graph using

- asynchronous channels **c** and **d**
- synchronous channels **e** and **f**

simply write

```
!timeout
?timer_on
?timer_off
```
specify the sender by a program graph using

- asynchronous channels c and d
- synchronous channels e and f
- Boolean variable x for the acknowledgement bit sent by the receiver
generate message(0)

try to send(0)
generate message(0) → try to send(0) → activate timer(0)

lost \(\rightarrow\) c!0
generate message(0)
try to send(0)
activate timer(0)

lost

wait(0)

?timeout

!timer_on

x=1

d?x

check ack(0)
generate message(0)

try to send(0)

activate timer(0)

wait(0)

check ack(0)

x = 1

d?x

x = 0:

!timer_off

generate message(1)
Program graph for the receiver

Sender

Timer

Receiver

wait for message(0)

process message(0)

acknowledge receipt(0)

wait for message(1)

process message(1)

acknowledge receipt(1)

reliable channel \( d \)

unreliable channel \( c \)

\( f \)

\( e \)

\( y = 1 \)
\[
\begin{align*}
G(0) & \rightarrow S(0) & S(0) & \rightarrow T(0) \\
G(1) & \rightarrow S(1) & S(1) & \rightarrow T(1) \\
W(0) & \rightarrow C(0) & C(0) & \rightarrow W(0) \\
W(1) & \rightarrow C(1) & C(1) & \rightarrow W(1)
\end{align*}
\]
Generate(0)  off  Wait(0)  \( c = \varepsilon \)  \( d = \varepsilon \)
Generate(0)   off  Wait(0)  \( c=\varepsilon \)  \( d=\varepsilon \)
Send(0)      off  Wait(0)  \( c=\varepsilon \)  \( d=\varepsilon \)
Generate(0) off Wait(0) $c=\varepsilon$ $d=\varepsilon$
Send(0) off Wait(0) $c=\varepsilon$ $d=\varepsilon$ message lost
Generate(0)  off  Wait(0)  $c=\varepsilon$  $d=\varepsilon$
Send(0)  off  Wait(0)  $c=\varepsilon$  $d=\varepsilon$  message lost
Timer_on(0)  off  Wait(0)  $c=\varepsilon$  $d=\varepsilon$
Generate(0) off Wait(0) $c=\epsilon$ $d=\epsilon$
Send(0) off Wait(0) $c=\epsilon$ $d=\epsilon$ message lost
Timer_on(0) off Wait(0) $c=\epsilon$ $d=\epsilon$
Wait(0) on Wait(0) $c=\epsilon$ $d=\epsilon$
Generate(0) off Wait(0) $c=\varepsilon$ $d=\varepsilon$
Send(0) off Wait(0) $c=\varepsilon$ $d=\varepsilon$ message lost
Timer_on(0) off Wait(0) $c=\varepsilon$ $d=\varepsilon$
Wait(0) on Wait(0) $c=\varepsilon$ $d=\varepsilon$ timeout
Generate(0) off Wait(0) \( c = \varepsilon \quad d = \varepsilon \)

Send(0) off Wait(0) \( c = \varepsilon \quad d = \varepsilon \) message lost

Timer_on(0) off Wait(0) \( c = \varepsilon \quad d = \varepsilon \)

Wait(0) on Wait(0) \( c = \varepsilon \quad d = \varepsilon \) timeout

Send(0) off Wait(0) \( c = \varepsilon \quad d = \varepsilon \)

........ try again

\[ 82/131 \]
Generate(0)  off  Wait(0)  \(c=\varepsilon\)  \(d=\varepsilon\)
Generate(0)  off  Wait(0)  c=\epsilon  d=\epsilon
Send(0)    off  Wait(0)  c=\epsilon  d=\epsilon
Generate(0)  off  Wait(0)  \( c=\varepsilon \)  \( d=\varepsilon \)  message 0 sent
Generate(0) off Wait(0) \( c = \varepsilon \) \( d = \varepsilon \)
Send(0) off Wait(0) \( c = \varepsilon \) \( d = \varepsilon \) message 0 sent
Timer_on(0) off Wait(0) \( c = 0 \) \( d = \varepsilon \)
Generate(0) off Wait(0) c=ε d=ε
Send(0) off Wait(0) c=ε d=ε message 0 sent
Timer_on(0) off Wait(0) c=0 d=ε
Wait(0) on Wait(0) c=0 d=ε
Generate(0)  off  Wait(0)  \(c=\epsilon\)  \(d=\epsilon\)
Send(0)    off  Wait(0)  \(c=\epsilon\)  \(d=\epsilon\)  message 0 sent
Timer_on(0) off  Wait(0)  \(c=0\)  \(d=\epsilon\)
Wait(0)    on   Wait(0)  \(c=0\)  \(d=\epsilon\)  timeout
Generate(0)  off  Wait(0)  \( c=\varepsilon \)  \( d=\varepsilon \)
Send(0)  off  Wait(0)  \( c=\varepsilon \)  \( d=\varepsilon \)  message 0 sent
Timer_on(0)  off  Wait(0)  \( c=0 \)  \( d=\varepsilon \)
Wait(0)  on  Wait(0)  \( c=0 \)  \( d=\varepsilon \)  timeout
Send(0)  off  Wait(0)  \( c=0 \)  \( d=\varepsilon \)
Generate(0) off Wait(0) c=ε d=ε
Send(0) off Wait(0) c=ε d=ε message 0 sent
Timer_on(0) off Wait(0) c=0 d=ε
Wait(0) on Wait(0) c=0 d=ε timeout
Send(0) off Wait(0) c=0 d=ε 0 sent again
Timer_on(0) off Wait(0) c=00 d=ε
Generate(0) off Wait(0) $c=\epsilon$ $d=\epsilon$
Send(0) off Wait(0) $c=\epsilon$ $d=\epsilon$ message 0 sent
Timer_on(0) off Wait(0) $c=0$ $d=\epsilon$
Wait(0) on Wait(0) $c=0$ $d=\epsilon$ timeout
Send(0) off Wait(0) $c=0$ $d=\epsilon$ 0 sent again
Timer_on(0) off Wait(0) $c=00$ $d=\epsilon$
Timer_on(0) off Proc(0) $c=0$ $d=\epsilon$ message received
Generate(0) off Wait(0) $c = \varepsilon$ $d = \varepsilon$  
Send(0) off Wait(0) $c = \varepsilon$ $d = \varepsilon$ message 0 sent  
Timer_on(0) off Wait(0) $c = 0$ $d = \varepsilon$  
Wait(0) on Wait(0) $c = 0$ $d = \varepsilon$ timeout  
Send(0) off Wait(0) $c = 0$ $d = \varepsilon$  
Timer_on(0) off Wait(0) $c = 00$ $d = \varepsilon$  
Timer_on(0) off Proc(0) $c = 0$ $d = \varepsilon$ message received  
Timer_on(0) off Ack(0) $c = 0$ $d = \varepsilon$
Generate(0) off Wait(0) c=ε d=ε
Send(0) off Wait(0) c=ε d=ε message 0 sent
Timer_on(0) off Wait(0) c=0 d=ε
Wait(0) on Wait(0) c=0 d=ε timeout
Send(0) off Wait(0) c=0 d=ε 0 sent again
Timer_on(0) off Wait(0) c=00 d=ε
Timer_on(0) off Proc(0) c=0 d=ε message received
Timer_on(0) off Ack(0) c=0 d=ε send ack via d
Wait(0) on Wait(0) $c=0$ $d=\varepsilon$ timeout
Send(0) off Wait(0) $c=0$ $d=\varepsilon$ 0 sent again
Timer_on(0) off Wait(0) $c=00$ $d=\varepsilon$ message received
Timer_on(0) off Proc(0) $c=0$ $d=\varepsilon$ send ack via $d$
Timer_on(0) off Ack(0) $c=0$ $d=\varepsilon$ receiver changes its mode
Timer_on(0) off Wait(1) $c=0$ $d=0$
receiver reads the same message again
receiver reads the same message again
receiver reads the same message again

receiver discards the message
receiver reads the same message again
receiver discards the message
Timer_on(0) off Wait(1) $c=0 \quad d=0$ receiver reads the same message again

Timer_on(0) off Proc(1) $c=\varepsilon \quad d=0$ receiver discards the message

Timer_on(0) off Wait(1) $c=\varepsilon \quad d=0$
Alternating bit protocol (ABP)

number of states in the TS:

\[ 10 \cdot 2 \cdot 6 \cdot \#\text{channel evaluations} \]
Alternating bit protocol (ABP)

number of states in the TS:

\[ 10 \cdot 2 \cdot 6 \cdot \# \text{channel evaluations} \]

\[ > 10^8 \quad \text{for FIFOs with capacity 10} \]
Variants of channel systems

• conditional communication actions $\ell \xrightarrow{c?x} \ell'$
Variants of channel systems

- conditional communication actions $\ell \xrightarrow{\mathbf{g}:c?x} \ell'$
- generalized sending instructions $c!\text{expr}$ instead of $c!v$

\[ c!2x+7 \]

\[ \ell \xrightarrow{d?x} \ell' \]
Variants of channel systems

- conditional communication actions $\ell \xrightarrow{c?x} \ell'$
- generalized sending instructions $c!expr$ instead of $c!v$
  e.g. $d?x$
  \begin{align*}
  \ell & \xrightarrow{g:c?x} \ell' \\
  \ell & \xrightarrow{c!2x+7} \ell'
  \end{align*}
- communication as conditions $\ell \xleftarrow{c?x:\alpha} \ell'$
Variants of channel systems

- conditional communication actions $\ell \xleftarrow{g: c?x} \ell'$
- generalized sending instructions $c!expr$ instead of $c!v$
  
  e.g.

  $d?x$

  $c!2x+7$

- communication as conditions $\ell \xleftarrow{c?x:\alpha} \ell'$

  $\rightsquigarrow$ more compact TS-representations
Variants of channel systems

- conditional communication actions $\ell \xleftarrow{g : c?x} \ell'$

- generalized sending instructions $c!\text{expr}$ instead of $c!v$

- communication as conditions $\ell \xleftarrow{c?x : \alpha} \ell'$

- *open* channel systems $\mathcal{P}_1 | \ldots | \mathcal{P}_n$
  instead of *closed* channel systems $[\mathcal{P}_1 | \ldots | \mathcal{P}_n]$
Variants of channel systems

- conditional communication actions $\ell \xrightleftharpoons{g : c?x} \ell'$
- generalized sending instructions $c!expr$ instead of $c!v$
- communication as conditions $\ell \xrightleftharpoons{c?x : \alpha} \ell'$
- open channel systems $P_1 \mid \ldots \mid P_n$
  instead of closed channel systems $[P_1 \mid \ldots \mid P_n]$
Summary: parallel operators
Summary: parallel operators

(pure) interleaving for TS $\mathcal{T}_1 \parallel \mathcal{T}_2$

- only concurrency, no communication
- not applicable for competing systems
Summary: parallel operators

(pure) interleaving for TS $\mathcal{T}_1 \parallel \mathcal{T}_2$

- only concurrency, no communication
- not applicable for competing systems

synchronous message passing for TS $\mathcal{T}_1 \parallel_{\text{Syn}} \mathcal{T}_2$

- interleaving for concurrent actions
- synchronization via actions in $\text{Syn}$
Summary: parallel operators

(pure) interleaving for TS $T_1 \parallel T_2$

- only concurrency, no communication
- not applicable for competing systems

synchronous message passing for TS $T_1 \parallel_{\text{Syn}} T_2$

- interleaving for concurrent actions
- synchronization via actions in $\text{Syn}$

interleaving for program graphs $P_1 \parallel P_2$

- interleaving for concurrent actions
- communication via shared variables
Summary: parallel operators

(pure) interleaving for TS $\mathcal{T}_1 \parallel \mathcal{T}_2$

- only concurrency, no communication
- not applicable for competing systems

Synchronous message passing for TS $\mathcal{T}_1 \parallel_{\text{Syn}} \mathcal{T}_2$

- interleaving for concurrent actions
- synchronization via actions in $\text{Syn}$

Interleaving for program graphs $\mathcal{P}_1 \parallel \mathcal{P}_2$

- interleaving for concurrent actions
- communication via shared variables

Channel systems: open $\mathcal{P}_1 | \ldots | \mathcal{P}_n$ or closed $[\mathcal{P}_1 | \ldots | \mathcal{P}_n]$

- interleaving, shared variables, message passing
(pure) interleaving for TS $\mathcal{I}_1 \ ||| \ \mathcal{I}_2$
- only concurrency, no communication

Synchronous message passing for TS $\mathcal{I}_1 \ ||_{\text{Syn}} \ \mathcal{I}_2$
- interleaving, synchronization via actions in $\text{Syn}$

Interleaving for program graphs $\mathcal{P}_1 \ ||| \ \mathcal{P}_2$
- interleaving, shared variables

Channel systems: open $\mathcal{P}_1 \mid \ldots \mid \mathcal{P}_n$ or closed $[ \mathcal{P}_1 \mid \ldots \mid \mathcal{P}_n ]$
- interleaving, shared variables
- synchronous and asynchronous message passing

Synchronous product for TS $\mathcal{I}_1 \otimes \mathcal{I}_2$
- no interleaving, “pure” synchronization
Synchronous product

for parallel systems with fully synchronized processes

\[ \mathcal{T}_1 = (S_1, Act_1, \rightarrow_1, ...) \]
\[ \mathcal{T}_2 = (S_2, Act_2, \rightarrow_2, ...) \]

\} two TS

synchronous product:

\[ \mathcal{T}_1 \otimes \mathcal{T}_2 = (S_1 \times S_2, Act, \rightarrow, ...) \]
Synchronous product

for parallel systems with fully synchronized processes

\[ T_1 = (S_1, \text{Act}_1, \rightarrow_1, \ldots) \]
\[ T_2 = (S_2, \text{Act}_2, \rightarrow_2, \ldots) \]

\} two TS

synchronous product:

\[ T_1 \otimes T_2 = (S_1 \times S_2, \text{Act}, \rightarrow, \ldots) \]

where the action set \( \text{Act} \) is given by a function

\[ \text{Act}_1 \times \text{Act}_2 \rightarrow \text{Act}, \quad (\alpha, \beta) \mapsto \alpha * \beta \]

action name for the concurrent execution of \( \alpha \) and \( \beta \)
Synchronous product

for parallel systems with fully synchronized processes

\[ T_1 = (S_1, \text{Act}_1, \rightarrow_1, ...) \]
\[ T_2 = (S_2, \text{Act}_2, \rightarrow_2, ...) \]
\[ \{ \text{two TS} \} \]
synchronous product:

\[ T_1 \otimes T_2 = (S_1 \times S_2, \text{Act}, \rightarrow, ...) \]

where the action set \( \text{Act} \) is given by a function

\[ \text{Act}_1 \times \text{Act}_2 \rightarrow \text{Act}, \ (\alpha, \beta) \mapsto \alpha \ast \beta \]

action name for the concurrent execution of \( \alpha \) and \( \beta \)

if action names are irrelevant: \( \text{Act}_1 = \text{Act}_2 = \text{Act} = \{ \tau \} \)
Synchronous product

for parallel systems with fully synchronized processes

\[ T_1 = (S_1, \text{Act}_1, \rightarrow_1, ...) \]
\[ T_2 = (S_2, \text{Act}_2, \rightarrow_2, ...) \]

two TS synchronous product:

\[ T_1 \otimes T_2 = (S_1 \times S_2, \text{Act}, \rightarrow, ...) \]

transition relation \( \rightarrow \):

\[
\begin{align*}
  s_1 &\xrightarrow{\alpha} s'_1 & \land & s_2 &\xrightarrow{\beta} s'_2 \\
  \langle s_1, s_2 \rangle &\xrightarrow{\alpha*\beta} \langle s'_1, s'_2 \rangle 
\end{align*}
\]
Synchronous product for composing circuits

2 sequential circuits
Synchronous product for composing circuits

\[ x_1, \ldots, x_n \rightarrow C_1 \rightarrow y_1, \ldots, y_n \rightarrow C_2 \rightarrow z_1, \ldots, z_j \]

\[ w_1, \ldots, w_i \rightarrow C_1 \rightarrow r_1, \ldots, r_k \rightarrow C_2 \rightarrow t_1, \ldots, t_h \]
Synchronous product: example
Synchronous product: example

Initially:
\[ r_1 = 0 \]

Transition function:
\[ \delta_{r_1} = \neg r_1 \]
Synchronous product: example

Transition function:
\[ \delta_{r_1} = \neg r_1 \]

initially:
\[ r_1 = 0 \]

Transition function:
\[ \delta_{r_2} = r_2 \lor x \]

initially:
\[ r_2 = 0 \]
Synchronous product: example

TS for the composite circuit $\mathcal{T}_1 \otimes \mathcal{T}_2$
Synchronous product: example

TS for the composite circuit $\mathcal{T}_1 \otimes \mathcal{T}_2$
Synchronous product: example

TS for the composite circuit $T_1 \otimes T_2$
State explosion problem

TS for reactive systems can be enormously large
State explosion problem

TS for reactive systems can be enormously large

- **infinite** for systems with
  - variables of infinite domains, e.g., $\mathbb{N}$
  - infinite data structures, e.g., stacks, queues, lists,...
State explosion problem

TS for reactive systems can be enormously large

- infinite for systems with
  - variables of infinite domains, e.g., $\mathbb{N}$
  - infinite data structures, e.g., stacks, queues, lists,...

- if finite: exponential growth in
State explosion problem

TS for reactive systems can be enormously large

- **infinite** for systems with
  - variables of infinite domains, e.g., $\mathbb{N}$
  - infinite data structures, e.g., stacks, queues, lists,...

- if finite: **exponential growth** in
  - number of parallel components,
    e.g., state space of $\mathcal{T}_1 \parallel \ldots \parallel \mathcal{T}_n$ is $S_1 \times \ldots \times S_n$
State explosion problem

TS for reactive systems can be enormously large

- **infinite** for systems with
  - * variables of infinite domains, e.g., $\mathbb{N}$
  - * infinite data structures, e.g., stacks, queues, lists,...

- if finite: **exponential growth** in
  - * number of parallel components,
    e.g., state space of $T_1 \parallel \ldots \parallel T_n$ is $S_1 \times \ldots \times S_n$
  - * number of variables and channels
State explosion problem

TS for reactive systems can be enormously large

- **infinite** for systems with
  - variables of infinite domains, e.g., $\mathbb{N}$
  - infinite data structures, e.g., stacks, queues, lists, ...
- if finite: **exponential growth** in
  - number of parallel components, e.g., state space of $T_1 \parallel \ldots \parallel T_n$ is $S_1 \times \ldots \times S_n$
  - number of variables and channels

E.g., for channel systems: size of the state space is

$$|Loc_1| \cdot \ldots \cdot |Loc_n| \cdot \prod_{x \in Var} |Dom(x)| \cdot \prod_{c \in Chan} |Dom(c)|^{cap(c)}$$
Model checking

system $P_1 \parallel \ldots \parallel P_n$

transition system $\mathcal{T}$

requirements

specification $spec$

model checker

does $\mathcal{T}$ satisfy $spec$ ?

yes

no $\ominus$ error indication
Model checking

system \( P_1 \parallel \ldots \parallel P_n \)

transition system \( \mathcal{T} \)
for \( P_1 \parallel \ldots \parallel P_n \)

model checker

does \( \mathcal{T} \) satisfy \( \text{spec} \) ?

yes

no \( \Rightarrow \) error indication

requirements

specification \text{spec}
Model checking

- syntactic description of $P_1, \ldots, P_n$
- requirements
- specification $\text{spec}$
- SOS-rules
- transition system $\mathcal{T}$ for $P_1 \parallel \ldots \parallel P_n$
- model checker
  - does $\mathcal{T}$ satisfy $\text{spec}$?

- yes
- no $\pm$ error indication