Overview Lecture #4

- **Concurrency**
  - The interleaving paradigm

- **Communication principles**
  - Shared variable “communication”
  - Handshaking
    ⇒ **Synchronous communication**

- **Channel systems**

- **The state-space explosion problem**
Interleaving of transition systems

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$ $i=1, 2$, be two transition systems.

Transition system

$$TS_1 \parallel TS_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$$

where $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$ and the transition relation $\rightarrow$ is defined by the rules:

- $s_1 \xrightarrow{\alpha} s'_1$ and $\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s'_1, s_2 \rangle$
- $s_2 \xrightarrow{\alpha} s'_2$ and $\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s'_2 \rangle$
Interleaving of program graphs

Let $PG_i = (Loc_i, Act_i, Effect_i, \rightarrow, Loc_{0,i}, g_{0,i})$ over variables $Var_i$.

Program graph $PG_1 || PG_2$ over $Var_1 \cup Var_2$ is defined by:

$$(Loc_1 \times Loc_2, Act_1 \uplus Act_2, Effect, \rightarrow, Loc_{0,1} \times Loc_{0,2}, g_{0,1} \land g_{0,2})$$

where $\rightarrow$ is defined by the inference rules:

$$\begin{align*}
\ell_1 \xrightarrow{g:\alpha} \ell_1' && \quad \text{and} \quad \ell_2 \xrightarrow{g:\alpha} \ell_2' \\
\langle \ell_1, \ell_2 \rangle \xrightarrow{g:\alpha} \langle \ell_1', \ell_2 \rangle \quad \text{and} \quad \langle \ell_1, \ell_2 \rangle \xrightarrow{g:\alpha} \langle \ell_1, \ell_2' \rangle
\end{align*}$$

and $\text{Effect}(\alpha, \eta) = \text{Effect}_i(\alpha, \eta)$ if $\alpha \in Act_i$. 
Handshaking

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i), \ i=1, 2$ and $H \subseteq Act_1 \cap Act_2$

$$TS_1 \parallel_H TS_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$$

where $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$ and with $\rightarrow$ defined by:

- Interleaving for $\alpha \notin H$
  $$\begin{align*}
  s_1 & \xrightarrow{\alpha_1} s'_1 \\
  \langle s_1, s_2 \rangle & \xrightarrow{\alpha} \langle s'_1, s_2 \rangle
  \end{align*}$$

- Handshaking for $\alpha \in H$
  $$\begin{align*}
  s_1 & \xrightarrow{\alpha_1} s'_1 \\
  \langle s_1, s_2 \rangle & \xrightarrow{\alpha} \langle s'_1, s'_2 \rangle
  \end{align*}$$

  $$\begin{align*}
  s_2 & \xrightarrow{\alpha_2} s'_2 \\
  \langle s_1, s_2 \rangle & \xrightarrow{\alpha} \langle s_1, s'_2 \rangle
  \end{align*}$$
Synchronous parallelism

Let $TS_i = (S_i, Act, \rightarrow_i, I_i, AP_i, L_i)$ and $Act \times Act \rightarrow Act$, $(\alpha, \beta) \rightarrow \alpha \ast \beta$

$$TS_1 \otimes TS_2 = (S_1 \times S_2, Act, \rightarrow, I_1 \times I_2, AP_1 \uplus AP_2, L)$$

with $L$ as defined before and $\rightarrow$ is defined by the following rule:

$$\langle s_1, s_2 \rangle \xrightarrow{\alpha \ast \beta} \langle s'_1, s'_2 \rangle$$

typically used for synchronous hardware circuits
Example
Overview Lecture #4

- **Concurrency**
  - The interleaving paradigm

- **Communication principles**
  - Shared variable “communication”
  - Handshaking
  - Synchronous communication

⇒ **Channel systems**

- The state-space explosion problem
Channels

- Processes communicate via *channels* \((c \in Chan)\)
  - first-in, first-out buffers that may carry messages

- Process \(P_i = \text{program graph } PG_i + \text{communication actions}\)
  
  \(c!v\) transmit the value \(v\) along channel \(c\)
  \(c?x\) receive a message via channel \(c\) and assign it to variable \(x\)

- \(Comm = \{ c!v, c?x \mid c \in Chan, v \in dom(c), x \in Var. \ dom(x) \supseteq dom(c) \}\)
Channels

- Capacity of channel = maximum \# messages that can be stored
  - when \( \text{cap}(c) \in \mathbb{N} \), \( c \) is a channel with finite capacity
  - \( \text{cap}(c) = \infty \) indicates that \( c \) has an infinite capacity

- Sending and receiving a message
  - \( c!v \) puts the value \( v \) at the rear of the buffer \( c \) (if \( c \) is not full)
  - \( c?x \) retrieves the front element of the buffer and assigns it to \( x \) (if \( c \) is not empty)
  - if \( \text{cap}(c) = 0 \), channel \( c \) has no buffer

- If \( \text{cap}(c) = 0 \), communication via \( c \) amounts to handshaking

- If \( \text{cp}(c) > 0 \), there is a “delay” between sending and receipt
  - sending and receiving can never take place simultaneously
  - this is called \textit{asynchronous message passing}
Channel systems

A program graph over \((Var, Chan)\) is a tuple

\[
PG = (Loc, Act, Effect, \rightarrow, Loc_0, g_0)
\]

where

\[
\rightarrow \subseteq Loc \times (\text{Cond}(Var) \times Act) \times Loc \cup \overset{\text{communication actions}}{\text{Loc} \times \text{Comm} \times \text{Loc}}
\]

A channel system \(CS\) over \((\bigcup_{0<i\leq n} Var_i, Chan)\):

\[
CS = [PG_1 | \ldots | PG_n]
\]

with program graphs \(PG_i\) over \((Var_i, Chan)\)
Communication actions

• **Handshaking**
  
  – if $\text{cap}(c) = 0$, then process $P_i$ can perform $\ell_i \xrightarrow{c!v} \ell'_i$ only
  
  – . . . if $P_j$, say, can perform $\ell_j \xrightarrow{c?x} \ell'_j$
  
  – the effect corresponds to the (atomic) *distributed* assignment $x := v$.

• **Asynchronous message passing**

  – if $\text{cap}(c) > 0$, then process $P_i$ can perform $\ell_i \xrightarrow{c!v} \ell'_i$
  
  – . . . if and only if less than $\text{cap}(c)$ messages are stored in $c$
  
  – $P_j$ may perform $\ell_j \xrightarrow{c?v} \ell'_j$ if and only if the buffer of $c$ is not empty
  
  – then the first element $v$ of the buffer is extracted and assigned to $x$ (atomically)

<table>
<thead>
<tr>
<th>executable if . . .</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c!v$</td>
<td>$c$ is not “full”</td>
</tr>
<tr>
<td>$c?x$</td>
<td>$c$ is not empty</td>
</tr>
</tbody>
</table>
The alternating bit protocol
The alternating bit protocol: sender

\[
\begin{align*}
\text{snd msg}(0) & \quad \text{st tmr}(0) & \quad \text{wait}(0) & \quad \text{chk ack}(0) \\
\text{lost} & \quad \text{tmr on} & \quad x = 1 & \\
\text{timeout} & \quad \text{timeout} & \quad \text{lost} \\
\text{c!}\langle m, 0 \rangle & \quad \text{d?x} & \quad \text{c!}\langle m, 1 \rangle \\
x = 1 : \text{tmr off} & \quad x = 0 : \text{tmr off} & \\
x = 0 & \quad x = 0 \quad \text{timeout} & \quad \text{timeout}
\end{align*}
\]
The alternating bit protocol: receiver

\begin{align*}
\text{wait}(0) & \xrightarrow{\text{pr} \text{msg}(0)} \text{snd} \text{ack}(0) \\
\text{pr} \text{msg}(1) & \xrightarrow{\text{snd} \text{ack}(1)} \text{wait}(1) \\
\text{snd} \text{ack}(0) & \xrightarrow{\text{pr} \text{msg}(0)} \text{wait}(0) \\
\text{snd} \text{ack}(1) & \xrightarrow{\text{pr} \text{msg}(1)} \text{wait}(1) \\
\end{align*}
Channel evaluations

- A **channel evaluation** $\xi$ is
  - a mapping from channel $c \in Chan$ onto a sequence $\xi(c) \in \text{dom}(c)^*$ such that
  - current length cannot exceed the capacity of $c$: $\text{len}(\xi(c)) \leq \text{cap}(c)$
  - $\xi(c) = v_1 v_2 \ldots v_k$ ($\text{cap}(c) \geq k$) denotes $v_1$ is at front of buffer etc.

- $\xi[c := v_1, \ldots, v_k]$ denotes the channel evaluation

\[
\xi[c := v_1 \ldots v_k](c') = \begin{cases} 
\xi(c') & \text{if } c \neq c' \\
 v_1 \ldots v_k & \text{if } c = c'. 
\end{cases}
\]

- Initial channel evaluation $\xi_0$ equals $\xi_0(c) = \varepsilon$ for any $c$
Transition system semantics of a channel system

Let \( CS = [PG_1 \mid \ldots \mid PG_n] \) be a \textit{channel system} over \((Chan, Var)\) with

\[
PG_i = (Loc_i, Act_i, Effect_i, \sim_i, Loc_{0,i}, g_{0,i}), \quad \text{for } 0 < i \leq n
\]

\( TS(CS) \) is the \textit{transition system} \((S, Act, \rightarrow, I, AP, L)\) where:

- \( S = (Loc_1 \times \ldots \times Loc_n) \times \text{Eval(Var)} \times \text{Eval(Chan)} \)
- \( Act = \left( \bigcup_{0<i\leq n} Act_i \right) \cup \{\tau\} \)
- \( \rightarrow \) is defined by the inference rules on the next slides
- \( I = \left\{ \langle\ell_1, \ldots, \ell_n, \eta, \xi_0\rangle \mid \forall i. (\ell_i \in Loc_{0,i} \land \eta \models g_{0,i}) \land \forall c. \xi_0(c) = \epsilon \right\} \)
- \( AP = \bigcup_{0<i\leq n} Loc_i \cup \text{Cond(Var)} \)
- \( L(\langle\ell_1, \ldots, \ell_n, \eta, \xi\rangle) = \{\ell_1, \ldots, \ell_n\} \cup \{g \in \text{Cond(Var)} \mid \eta \models g\} \)
Inference rules (I)

- Interleaving for $\alpha \in Act_i$:

$$
\ell_i \xrightarrow{g;\alpha} \ell_i' \land \eta \models g \\
\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\alpha} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi \rangle
$$

where $\eta' = Effect(\alpha, \eta)$

- Synchronous message passing over $c \in Chan$, $cap(c) = 0$:

$$
\ell_i \xrightarrow{c?x} \ell_i' \land \ell_j \xrightarrow{c!v} \ell_j' \land i \neq j \\
\langle \ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_j', \ldots, \ell_n, \eta', \xi \rangle
$$

where $\eta' = \eta[x := v]$.  

© JPK
Inference rules (II)

- Asynchronous message passing for $c \in Chan$, $\text{cap}(c) > 0$:
  - receive a value along channel $c$ and assign it to variable $x$:
    \[
    \ell_i \xrightarrow{c?x} \ell_i' \land \text{len}(\xi(c)) = k > 0 \land \xi(c) = v_1 \ldots v_k \\
    \langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi' \rangle
    \]
    where $\eta' = \eta[x := v_1]$ and $\xi' = \xi[c := v_2 \ldots v_k]$.

  - transmit value $v \in \text{dom}(c)$ over channel $c$:
    \[
    \ell_i \xrightarrow{c!v} \ell_i' \land \text{len}(\xi(c)) = k < \text{cap}(c) \land \xi(c) = v_1 \ldots v_k \\
    \langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta, \xi' \rangle
    \]
    where $\xi' = \xi[c := v_1 v_2 \ldots v_k v]$.
#4: Channel systems

## Handling unexpected messages

<table>
<thead>
<tr>
<th>sender $S$</th>
<th>timer</th>
<th>receiver $R$</th>
<th>channel $c$</th>
<th>channel $d$</th>
<th>event</th>
</tr>
</thead>
<tbody>
<tr>
<td>$snd_{msg}(0)$</td>
<td>off</td>
<td>$wait(0)$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>message with bit 0 transmitted</td>
</tr>
<tr>
<td>$st_{tmr}(0)$</td>
<td>off</td>
<td>$wait(0)$</td>
<td>$\langle m, 0 \rangle$</td>
<td>$\emptyset$</td>
<td>timeout</td>
</tr>
<tr>
<td>$wait(0)$</td>
<td>on</td>
<td>$wait(0)$</td>
<td>$\langle m, 0 \rangle$</td>
<td>$\emptyset$</td>
<td>retransmission</td>
</tr>
<tr>
<td>$snd_{msg}(0)$</td>
<td>off</td>
<td>$wait(0)$</td>
<td>$\langle m, 0 \rangle$</td>
<td>$\emptyset$</td>
<td>receiver reads</td>
</tr>
<tr>
<td>$st_{tmr}(0)$</td>
<td>off</td>
<td>$wait(0)$</td>
<td>$\langle m, 0 \rangle \langle m, 0 \rangle$</td>
<td>$\emptyset$</td>
<td>first message</td>
</tr>
<tr>
<td>$st_{tmr}(0)$</td>
<td>off</td>
<td>$pr_{msg}(0)$</td>
<td>$\langle m, 0 \rangle$</td>
<td>$\emptyset$</td>
<td>receiver changes into mode-1</td>
</tr>
<tr>
<td>$st_{tmr}(0)$</td>
<td>off</td>
<td>$snd_{ack}(0)$</td>
<td>$\langle m, 0 \rangle$</td>
<td>$\emptyset$</td>
<td>receiver reads</td>
</tr>
<tr>
<td>$st_{tmr}(0)$</td>
<td>off</td>
<td>$wait(1)$</td>
<td>$\langle m, 0 \rangle$</td>
<td>0</td>
<td>retransmission</td>
</tr>
<tr>
<td>$st_{tmr}(0)$</td>
<td>off</td>
<td>$pr_{msg}(1)$</td>
<td>$\emptyset$</td>
<td>0</td>
<td>receiver reads</td>
</tr>
<tr>
<td>$st_{tmr}(0)$</td>
<td>off</td>
<td>$wait(1)$</td>
<td>$\emptyset$</td>
<td>0</td>
<td>retransmission and ignores it</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>
Overview Lecture #4

- **Concurrency**
  - The interleaving paradigm

- **Communication principles**
  - Shared variable “communication”
  - Handshaking
  - Synchronous communication

- **Channel systems**

⇒ The state-space explosion problem
Sequential programs

- The number of states of a simple sequential computer program is worst case:

\[ |\#\text{program locations}| \cdot \prod_{\text{variable } x} |\text{dom}(x)| \]

\[ \Rightarrow \text{number of states grows exponentially in the number of program variables} \]

- \( N \) variables with \( k \) possible values each yields \( k^N \) states
- this is called the state-space explosion problem

- A program with 10 locations, 3 bools, 5 integers (in range 0...9):

\[ 10 \cdot 2^3 \cdot 10^5 = 800,000 \text{ states} \]

- Adding a single 50-positions bit-array yields \( 800,000 \cdot 2^{50} \) states
Concurrent programs

- The \# states of $P \equiv P_1 \parallel \ldots \parallel P_n$ is maximally:

\[
\text{#states of } P_1 \times \ldots \times \text{#states of } P_n
\]

$\Rightarrow$ \# states grows \textit{exponentially} with the number of components

- The composition of $N$ components of size $k$ each yields $k^N$ states

- This is called \textit{the state-space explosion problem}
Dijkstra’s mutual exclusion program

- two bit-arrays of size $N$
- global variable $k$
  - with value in $1, \ldots, N$
- local variable $l$
  - with value in $1, \ldots, N$
- 6 program locations per process

$\Rightarrow$ totally $2^N \cdot N \cdot (6N)^N$ states
Channel systems

- Asynchronous communication of processes via *channels*
  - each channel $c$ has a bounded capacity $\text{cap}(c)$
  - if a channel has capacity 0, we obtain handshaking

- The number of states of a system with $N$ components and $K$ channels is worst case

\[
\prod_{i=1}^{N} \left( |\text{#program locations}| \prod_{\text{variable } x} |\text{dom}(x)| \right) \cdot \prod_{j=1}^{K} |\text{dom}(c_j)|^{\text{cap}(c_j)}
\]

*This is the underlying structure of Promela*
The alternating bit protocol

Channel capacity 10, and datums are bits, yields $2 \cdot 8 \cdot 6 \cdot 4^{10} \cdot 2^{10} = 3 \cdot 2^{35} \approx 10^{11}$ states