Chapter 2: Modelling Concurrent Systems

Prof. Ali Movaghar
Verification of Reactive Systems
A prerequisite for model checking is to provide a model of the system.

- We introduce transition systems, a standard class of models to represent hardware and software system.
- We explain different aspects for modeling concurrent systems.
- Finally we explain about the state-space explosion problem encountered in model checking!
Transition Systems

- Transition Systems (TSs) model the behavior of systems.
- TSs are directed graphs where nodes and edges represent states and transitions respectively.
Transition Systems (Con.)

- A state describes information about a system at a certain moment of its behavior:
  - The current color of a traffic light.
  - The current values of all program variables + the program counter.
  - The current value of the registers together with the values of the input bits.
Transition Systems (Con.)

- Transitions specify how the system evolve from one state to another.
  - A switch from one color to another (for traffic light).
  - The execution of a program statement
  - The change of the registers and output bits for a new input.

- In following we formally define a transition system.
Transition Systems (Con.)

- **Definition**: A transition system TS is a tuple $(S, \text{Act}, \rightarrow, I, \text{AP}, L)$ where:
  - $S$ is a set of states;
  - Act is a set of actions;
  - $\rightarrow \subseteq S \times \text{Act} \times S$ is a transition relation;
  - $I \subseteq S$ is a set of initial states;
  - AP is a set of atomic proposition, and
  - $L: S \rightarrow 2^{\text{AP}}$ is a labeling function.

- TS is called finite if $S$, Act, and AP are finite.
Transition Systems (Con.)

- We write \( s \alpha \rightarrow s' \) instead of \( (s, \alpha, s') \in \rightarrow \).

- The intuitive behavior of a TS is:
  - The TS starts in some initial state \( s_0 \in I \) and evolves according to the transition relation \( \rightarrow \).
  - Evolution means: if \( s \) is the current state, a transition \( s \alpha \rightarrow s' \) is selected nondeterministically and taken.
  - Take means: the action \( \alpha \) is performed and the TS evolves from state \( s \) to state \( s' \).
  - This procedure is repeated in state \( s' \) and finishes once a state is encountered that has no outgoing transition.
The labeling function $L$ relates the set of propositions $L(s) \in 2^{AP}$ to any state $s$.

- $L(s)$ stands for atomic propositions $a \in AP$ which are satisfied by state $s$.
- Hence, given a propositional logic formula $\varphi$, $s$ satisfies formula $\varphi$, if $L(s)$ satisfies it:

$$s \models \varphi \iff L(s) \models \varphi$$
**Example 2.2:** Beverage Vending Machine (BVM)

- $S = \{\text{pay, select, coffee, tea}\}$
- $\text{Act} = \{\text{insert\_coin, get\_coffee, get\_tea, } \tau\}$
The atomic propositions in the TS depends on the properties under consideration.

- $AP = S$, $L(s) = \{s\}$, or
- $AP = \{\text{paid}, \text{drink}\}$ to verify “The vending machine only delivers a drink after paying”.

Hence, $L(\text{pay}) = \{}$, $L(\text{coffee}) = L(\text{tea}) = \{\text{paid, drink}\}$, $L(\text{select}) = \{\text{paid}\}$. 
Transition Systems (Con.)

- To formally model a system using LTs, we should define the set of Act and AP.
  - Actions are only necessary for modeling communication mechanism. When action names are irrelevant (e.g. internal actions), we use symbol $\tau$ or even remove it.
  - Propositions are always chosen depending on the characteristics of interest.
Transition Systems (Con.)

- Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system. For $s \in S, \alpha \in Act$, the set of direct $\alpha$–successor of $s$ is:

  \[
  \text{Post}(s, \alpha) = \{ s' \in S | s \xrightarrow{\alpha} s' \}, \quad \text{Post}(s) = \bigcup_{\alpha \in \text{Act}} \text{Post}(s, \alpha)
  \]

- Similarly the set of $\alpha$-predecessor of $s$ is defined by:

  \[
  \text{Pre}(s, \alpha) = \{ s' \in S | s' \xleftarrow{\alpha} s \}, \quad \text{Pre}(s) = \bigcup_{\alpha \in \text{Act}} \text{Pre}(s, \alpha)
  \]
The notation for the sets of direct successors are expanded to subsets of $S$ ($C \subseteq S$):

$$Post(C, \alpha) = \bigcup_{s \in C} Post(s, \alpha), \quad Post(C) = \bigcup_{s \in C} Post(s)$$

Similary

$$Pre(C, \alpha) = \bigcup_{s \in C} Pre(s, \alpha), \quad Pre(C) = \bigcup_{s \in C} Pre(s)$$
State $s$ in transition system $TS$ is called **terminal** if and only if $\text{Post}(s)=\emptyset$.

Transition systems modelling a sequential computer program, terminal states represents termination of the program.
The behavior of a transition system is formally defined by the notion of executions (also called runs).

Informally, an execution of a TS results from the resolution of the possible non-determinism in the system.
Definition: A finite execution fragment $\sigma$ of $TS = (S, Act, \rightarrow, I, AP, L)$ is an alternating sequence of states and actions ending with a state:

$$\sigma = s_0 \alpha_1 s_1 \alpha_2 ... \alpha_n s_n \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \forall 0 \leq i \leq n$$

where $n$ is called the length of execution fragment $\sigma$. 
An infinite execution fragment $\rho$ of TS is an infinite alternating sequence of states and actions:

$$\rho = s_0 \alpha_1 s_1 \alpha_2 \ldots \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \forall 0 \leq i$$

A maximal execution fragment is a finite execution fragment that ends in a terminal state or an infinite execution fragment.

An execution fragment, started in an initial state $s_0 \in S$, is called initial.
Transition Systems (Con.)

- **Example 2.8**: executions of the BVM

  \[ \rho_1 = \text{pay} \xrightarrow{\text{coin}} \text{select} \xrightarrow{\tau} \text{tea} \xrightarrow{\text{tget}} \text{pay} \xrightarrow{\text{coin}} \text{select} \xrightarrow{\tau} \text{tea} \xrightarrow{\text{tget}} \ldots \]
  \[ \rho_2 = \text{select} \xrightarrow{\tau} \text{tea} \xrightarrow{\text{tget}} \text{pay} \xrightarrow{\text{coin}} \text{select} \xrightarrow{\tau} \text{coffee} \xrightarrow{\text{cget}} \ldots \]
  \[ \sigma = \text{pay} \xrightarrow{\text{coin}} \text{select} \xrightarrow{\tau} \text{tea} \xrightarrow{\text{tget}} \text{pay} \xrightarrow{\text{coin}} \text{select} \xrightarrow{\tau} \text{tea}. \]

- \( \rho_1 \) and \( \sigma \) are initial, while only \( \rho_1 \) and \( \rho_2 \) are maximal.
Definition: An execution of transition system TS is an initial and maximal execution fragment.

- In example 2.8, \( \rho_1 \) is an execution while \( \rho_2 \) and \( \sigma \) are not.
Transition Systems (Con.)

- **Definition**: A state \( s \in S \) is called **reachable** in \( TS = (S, Act, \rightarrow, I, AP, L) \) if there exists an initial, finite execution fragment

\[
S_0 \xrightarrow{\alpha_1} S_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} S_n = S.
\]

- Reach(TS) denotes the set of all reachable states in TS.
**Example 2.11**: consider the sequential circuit with input variable $x$, output variable $y$ and register $r$ where

$$\lambda_y = \overline{(x \oplus r)} \text{ and } \delta_r = x \lor r$$
The circuit behavior is modeled by the transition system TS with state space \( S = \text{Eval}(x, r) \) where \( \text{Eval}(x, r) \) stands for the set of evaluations of \( x \) and \( r \).

The initial states of TS are \( I = \{ <x=0, r=0>, <x=1, r=0> \} \).

The set of actions is irrelevant and omitted here.
The transitions result directly from the functions $\lambda_y$ and $\delta_r$:

- For instance if the next input bit ($x$) equals $i=0,1$, then $<x = 0, r = 1> \rightarrow <x = i, r = 1>$
AP={x,y,r} and the labeling relation L assign the set of atomic proposition which has a value equal to 1.

The state <x=0,r=1> is labeled with {r} since the circuit function ¬(x ⊕ r) result 0 for y.
Sequential Hardware Circuits (Con.)

- Alternatively, using the set of proposition $\text{AP'} = \{x,y\}$, the register evaluation is invisible, we obtain:
  - $L'(\langle x=0, r=0 \rangle) = \{y\}$  $L'(\langle x=0, r=1 \rangle) = \{\}$
  - $L'(\langle x=1, r=0 \rangle) = \{x\}$  $L'(\langle x=1, r=1 \rangle) = \{x,y\}$
- We can still formalize the property that “the output bit $y$ is set infinitely often”.
This approach can be generalized for the sequential circuits with $n$ input $x_1, \ldots, x_n$, $m$ output bits $y_1, \ldots, y_m$ and $k$ registers $r_1, \ldots, r_k$:

- $S = \text{Eval}(x_1, \ldots, x_n, r_1, \ldots, r_k)$.
- $I = \{ a_1, \ldots, a_n, c_{0,1}, \ldots, c_{0,k} | a_1, \ldots, a_n \in \{0,1\} \}$ where $c_{0,i}$ is the initial value of $r_i$.
Sequential Hardware Circuits (Con.)

- $AP = \{x_1, \ldots, x_n, r_1, \ldots, r_k, y_1, \ldots, y_m\}$.
- $L(a_1, \ldots, a_n, c_1, \ldots, c_k) = \{x_i | a_i = 1\} \cup \{r_i | c_i = 1\} \cup \{y_i | s \models \lambda y_i(a_1, \ldots, a_n, c_1, \ldots, c_k) = 1\}$
- $(a_1, \ldots, a_n, c_1, \ldots, c_k) \xrightarrow{\tau} (a_1, \ldots, a_n, c'_1, \ldots, c'_k)$ if and only if $c'_j = \delta_{r_j} (a_1, \ldots, a_n, c_1, \ldots, c_k)$. 

Data-Dependant Systems

- **Example 2.12**: Consider an extension of BVM (Example 2.2) which
  - counts the number of coffee and tea left;
  - returns inserted coin if the machine is empty.

  For simplicity, BVM is modeled by two locations \textit{start} and \textit{select}. 
Data-Dependant Systems (Con.)

- We exploit conditional transitions $g: \alpha$ to model the data-depandanant behavior of the system, where $g$ is a condition and $\alpha$ is an action which is possible once $g$ holds.

- Hence below transitions model insertion of coin and refilling the machine:

\[
\begin{align*}
\text{start} & \xrightarrow{\text{true:coin}} \text{select} \\
\text{start} & \xrightarrow{\text{true:refill}} \text{start}
\end{align*}
\]
Data-Dependant Systems (Con.)

- Suppose the variables of ncof and ntea record the number of bottles of coffee and tea respectively.

\[
\text{select } \begin{array}{c}
\text{ncof > 0: cget} \\
\rightarrow \text{start}
\end{array} \quad \text{select } \begin{array}{c}
\text{ntea > 0: tget} \\
\rightarrow \text{start}
\end{array}
\]

\[
\text{select } \begin{array}{c}
\text{ncof = 0 \land ntea = 0: ret _coin} \\
\rightarrow \text{start}
\end{array}
\]
Data-Dependant Systems (Con.)

- The effect of each action on variables ncof and ntea is shown below:

<table>
<thead>
<tr>
<th>Action</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>refill</td>
<td>ncof:= max; ntea:=max</td>
</tr>
<tr>
<td>cget</td>
<td>ncof:=ncof-1</td>
</tr>
<tr>
<td>tget</td>
<td>ntea:=ntea-1</td>
</tr>
<tr>
<td>coin</td>
<td>No change</td>
</tr>
<tr>
<td>ret_coin</td>
<td>No change</td>
</tr>
</tbody>
</table>

- The graph consisting of locations as node and conditional transition as edge is not a TS.
Data-Dependant Systems (Con.)

- However a TS can be obtained by **unfolding** this graph.
This approach in modeling data-depandant systems can be formalized by program graphs over a set of Var of typed variables.

A program graph over a set of typed variables is a digraph whose edges are labeled by conditions on these variables and actions.
The effect of the actions is formalized by means of mapping
\[
\text{Effect: } \text{Act} \times \text{Eval}(\text{Var}) \rightarrow \text{Eval}(\text{Var})
\]
This indicates how the \textit{evaluation} \( \eta \) of variables is changed by performing an action.
**Definition**: A program graph overt set $\text{Var}$ of type variables is a tuple $(\text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0)$ where:

- $\text{Loc}$ is a set of locations and $\text{Act}$ is a set of actions;
- $\text{Effect} : \text{Act} \times \text{Eval}(\text{Var}) \rightarrow \text{Eval}(\text{Var})$;
- $\rightarrow \subseteq \text{Loc} \times \text{Cond}(\text{Var}) \times \text{Act} \times \text{Loc}$;
- $\text{Loc}_0 \subseteq \text{Loc}$ is a set of initial locations;
- $g \in \text{Cond}(\text{Var})$ is the initial condition.
Data-Dependant Systems (Con.)

- We write $\ell \xrightarrow{g: \alpha} \ell'$ instead of $(\ell, g, \alpha, \ell') \in \rightarrow$.
- The condition $g$ is called guard of the conditional transition.
- The behavior in location $\ell \in \text{Loc}$ depends on the current variable evaluation $\eta$. A non-deterministic choice is made between transitions whose guard is satisfied in evaluation $\eta$ ($\eta \models g$).
Each program graph can be interpreted as a transition system which results from unfolding.

In other words, the semantics of a program graph is defined by a transition system.
The states of the corresponding TS consist of a location \( \ell \in \text{Loc} \) of program graph together with an evaluation \( \eta \) of the variables: \( \langle \ell, \eta \rangle \).

Initial states are initial locations that satisfy the initial condition \( g_0 \).

The set \( \text{AP} \) is comprised by locations and boolean conditions for the variables.
Data-Dependant Systems (Con.)

- The **labeling** of states is such that $\langle \ell, \eta \rangle$ is labeled with $\ell$ and with all conditions (over Var) that hold in $\eta$.

- The **transition** relation is defined as follows: whenever $\ell \xrightarrow{g;\alpha} \ell'$ and $g$ holds in the current evaluation $\eta$, then:

$$\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle$$
**Definition:** The transition system $TS(GP)$ of program graph $GP = (Loc, Act, Effect, \rightarrow, Loc_0, g_0)$ over set $Var$ is the tuple $(S, Act, \rightarrow, I, AP, L)$ where:

- $S = Loc \times \text{Eval}(Var)$;
- $\rightarrow \subseteq S \times Act \times S$ is defined by the following rule:

$$
\ell \xrightarrow{g;\alpha} \ell' \land \eta | = g \\
\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle
$$
Data-Dependant Systems (Con.)

- \( I = \{ \langle \ell, \eta \rangle \mid \ell \in \text{Loc}_0, \eta \models g_0 \} \);
- \( \text{AP} = \text{Loc} \cup \text{Cond}(\text{Val}) \);
- \( L(\langle \ell, \eta \rangle) = \{ \ell \} \cup \{ g \in \text{Cond}(\text{Var}) \mid \eta \models g \} \).

The definition TS(PG) determines a very large set of propositions AP. But generally only a small part of AP is necessary to formulate system properties.
The transition relation of TS(PG) is defined using the so-called SOS-notation:

\[
\text{premise} \quad \frac{\text{premise}}{\text{conclusion}}
\]

Which implies if the proposition above the “solid line” holds, then the proposition under the fraction bar holds as well.

If the premise is a tautology, the rule is called axiom.
The relations defined by SOS rules are the **smallest** relation satisfying the indicated axioms and rules.

The semantics of many modeling languages like process algebras are defined using SOS rules.
**Parallelism and Communication**

- In reality most hard- and software systems are not **sequential** but **parallel**.
- We describe several mechanisms to provide operational models for parallel systems by means of TSs.
  - These Mechanisms range from simple mechanism where no communication take place between TSs to message-passing one.
Parallelism and Communication (Con.)

- Assume the operational (stepwise) behavior of the processes that run in parallel are given by transition systems $TS_1, \ldots, TS_n$.

- Our goal is to define an operator $\|\|$ such that $TS = TS_1 \| TS_2 \| \ldots \| TS_n$ is a transition system that specifies the behavior of parallel composition of TSs.
Parallelism and Communication (Con.)

- It should be noted that each $TS_i$ may be a transition system that is composed of several transition systems:

  $$TS_i = TS_{i,1} || TS_{i,2} || \ldots || TS_{i,n}$$

- Hence complex systems can be described in a rather structured way using the parallel composition in a hierarchical way.
Parallelism and Communication: Interleaving

- In this model, it is assumed that a system is actually composed of a set of independent components.
- Hence, the global system state is composed of the current individual states of the components.
Parallelism and Communication: Interleaving (Con.)

- Actions of an independent components are merged or “interleaved” with actions from other components.
- Thus concurrency is presented by pure interleaving: nondeterministic choice between activities of parallel processes.
**Example**: Two independent traffic lights

- ||| denotes interleaving operator.

![Diagram](image)
Parallelism and Communication: Interleaving (Con.)

- An important justification for **interleaving** is the fact that the effect of currently executed, independent actions $\alpha$ and $\beta$ is identical to the effect when $\alpha$ and $\beta$ are **executed in arbitrary order**:
  - $\text{Effect}(\alpha || \beta, \eta) = \text{Effect}(\alpha; \beta) + \text{Effect}(\beta; \alpha)$
  - where $;$ stands for sequential execution and $+$ stands for non-deterministic choice.
**Definition:** The transition system $TS_1 || TS_2$ where $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$ is defined by

- $(S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$

Where the $TS \rightarrow$ is defined by :

- $s_1 \xrightarrow{\alpha} s_1'$
- $\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle$
- $s_2 \xrightarrow{\alpha} s_2'$
- $\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle$

- Labeling function is defined :
$\text{L}(\langle s_1, s_2 \rangle) = L(s_1) \cup L(s_2)$
Parallelism and Communication: Com. via shared variable

- Consider two independent value assignments $x := x + 1 \quad ||| \quad y := y - 2$.
- The order of their execution is not important, so they can be easily interleaved.
Parallelism and Communication: Com. via shared variable

- Consider two parallel assignments on a shared variable $x$, $x := 2 \cdot x$ $\mid\mid$ $x := x + 1$.
- The different order of execution results in different final value
Parallelism and Communication: Com. via shared variable

- Thus interleaving is not appropriate when two components compete on a shared variable.
- Hence to deal with parallel programs with shared variables, an interleaving operator will be defined on the level of program graphs $PG_1$ and $PG_2$.
- $TS(PG_1 || PG_2)$ describe a parallel system communicating via shared variables.
Parallelism and Communication: Com. via shared variable

**Definition:** Program graph $PG_1 ||| PG_2$ over $\text{Var}_1 \cup \text{Var}_2$, where for $i=1,2$, $PG_i = (\text{Loc}_i, \text{Act}_i, \text{Effect}_i, \rightarrow, \text{Loc}_0, i, g_{0,i})$, is defined by

- $PG_1 ||| PG_2 = (\text{Loc}_1 \times \text{Loc}_2, \text{Act}_1 \cup \text{Act}_2, \text{Effect}, \rightarrow, \text{Loc}_{0,1} \times \text{Loc}_{0,2}, g_{0,1} \land g_{0,2})$ where
- $\rightarrow$ is defined by the rules:
Parallelism and Communication: Com. via shared variable

Effect(\(\alpha, \eta\)) = Effect_1(\(\alpha, \eta\)) if \(\alpha \in \text{Act}_i\).

Program graphs PG_1 and PG_2 have the variables Var_1 \(\cap\) Var_2 in common. Variables in Var_1 \(\setminus\) Var_2 are local variables of PG_1.
Example 2.22: consider the program graphs PG₁ and PG₂ that correspond to assignments x:=x+1 and x:=2·x.
Parallelism and Communication: Com. via shared variable

- its underlying transition system $TS(PG_1 \parallel\parallel PG_2)$ is shown below:
  - The non-determinism in the initial state does not represent concurrency but just a possible resolution of contention on $x$.
  - $TS(PG_1 \parallel\parallel PG_2) \neq TS(PG_1) \parallel\parallel TS(PG_2)$. 
Parallelism and Communication: Com. via shared variable

- **Question:** which actions of program graphs can be interleaved arbitrary (i.e. executed arbitrary) and which one can not? (page 42)
- Read examples 2.24 and 2.25.
Parallelism and Communication: Handshaking

- Until now, we have introduced two mechanisms for parallel processes:
  - Interleaving: processes evolve completely autonomously.
  - Com. via shared variables: processes communicate via shared variables.
Parallelism and Communication: Handshaking (Con.)

- The third mechanism in which current processes that want to interact have to do this in a synchronous fashion.

- In other words, processes can interact if they are both participating in this interaction at the same time - they shake hands.
Parallelism and Communication: Handshaking (Con.)

- During handshaking, information can be exchanged.
- Here we abstract of information exchange (message contents) by considering the occurrence of handshake.
- A set $H$ of handshake actions is distinguished with $\tau \notin H$. 
Informally, if both parallel processes are ready to execute the same handshake action, can message passing take place.

All actions outside $H$ (Act\(\setminus\)H) are independent and therefore can be executed autonomously in an interleaved fashion.
Parallelism and Communication: Handshaking (Con.)

**Definition:** The transition system $TS_1 \|_H TS_2$, where $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$ for $i = 1, 2$ and $H \subseteq Act_1 \cap Act_2$ with $\tau \notin H$, is defined by:

- $(S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$
- $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$
- The transition relation $\rightarrow$ is defined by the below rules:
Parallelism and Communication: Handshaking (Con.)

- Interleaving for $\alpha \not\in H$:

  $s_1 \xrightarrow{\alpha} s_1'$

  $\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle$

- Handshaking for $\alpha \in H$:

  $s_1 \xrightarrow{\alpha} s_1'$

  $s_2 \xrightarrow{\alpha} s_2'$

  $\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2' \rangle$
Parallelism and Communication: Handshaking (Con.)

- We write $TS_1 || TS_2$ for $TS_1 ||_H TS_2$ for $H = \text{Act}_1 \cap \text{Act}_2$.
- When $H = \{\}$, then $TS_1 || \{\} TS_2$ reduces to $TS_1 || || TS_2$.
- The handshaking operator $||_H$ is communicative, but not associative:
  - $TS_1 ||_H TS_2 = TS_2 ||_H TS_1$.
  - $TS_1 ||_H (TS_2 ||_H' TS_3) \neq (TS_1 ||_H TS_2) ||_H' TS_3$. 
Parallelism and Communication: Handshaking (Con.)

- The (multiway) **handshaking** operator in $TS = TS_1 \parallel_H TS_2 \parallel_H ... \parallel_H TS_n$, where $H \subseteq \text{Act}_1 \cap ... \cap \text{Act}_n$, is appropriate to model broadcasting!

- The handshaking operator is similar to parallel operator in CSP algebra.
Parallelism and Communication: Handshaking (Con.)

- **Example 2.28**: consider two processes $P_1$ and $P_2$ of the form:

  \[
  P_i \text{ loop forever}
  \]

  ... 

  request 
  critical section 
  release 

  .... 

  end loop 

  Modeled by
Parallelism and Communication: Handshaking (Con.)

- $P_1$ and $P_2$ can communicate with a new process Arbiter to enter their critical sections:
  - Arbiter mimics a binary semaphore.
  - $TS_{Arb} = (TS_1 ||| TS_2) ||$ Arbiter guarantees mutual exclusion (why?).
Parallelism and Communication: Handshaking (Con.)
Parallelism and Communication: Handshaking (Con.)

- Read Example 2.30.
- It should be noted that the parallel operators introduced are time-abstract.
  - See example 2.31.
Parallelism and Communication: Channel Systems

- Channel systems are parallel systems where processes communicate via channels.
  - Channels are first-in, first-out buffers that may contain messages.
  - Processes are program graphs $PG_i$ extended with communication actions:
    - $c!v$ transmits the value $v$ along channel $c$.
    - $c?x$ receives a message via $c$ and assigns it to $x$. 
Definition: A program graph over \((\text{Var}, \text{Chan})\) is a tuple \((\text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0)\) as before with the only difference that

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

\[ \text{Comm} = \{ c!v, c?x \mid c \in \text{Chan}, v \in \text{dom}(c), x \in \text{Var} \text{ with } \text{dom}(c) \subseteq \text{dom}(x) \} \]
A channel system CS over (Var, Chan) consists of program graphs PG\textsubscript{i} over (Var, Chan) with Var=\(\bigcup_{1 \leq i \leq n} Var\textsubscript{i}\).

- We denote CS=[PG\textsubscript{1} | ... | PG\textsubscript{n}]

As before the conditional transitions labeled by \(\alpha: g\), \(\alpha \in Act\) can happen whenever g holds.
The occurrences of conditional transitions labeled by $g:c!v$ or $g:c?x$ depends on the current evaluation and the capacity and content of channel.

- Handshaking: if $\text{cap}(c)=0$, then process $P_i$ can perform $\ell_i \xrightarrow{c!v} \ell'_i$ only if another process $P_j$ receives $\ell_j \xrightarrow{c!x} \ell'_j$. 
Parallelism and Communication: Channel Systems (Con.)

- Asynchronous: if $\text{cap}(c) > 0$
  - $P_i$ can send if the channel $c$ is not full.
  - $P_j$ can receive if channel $c$ is not empty.
  - The Effect of actions are defined as below; the channel is treated as a buffer:

<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>

- Channel systems are used to model communication protocols.
Parallelism and Communication: Channel Systems (Con.)

- **Example 2.23**: Alternating Bit Protocol
  - The Sender $S$ sends its messages along an unreliable channel $c$ to a receiver $R$, who sends its acknowledges along a reliable channel $d$ to the sender.
Parallelism and Communication: Channel Systems (Con.)

- S and R uses two bits 0,1 to distinguish retransmissions of m from transmissions of subsequent (and previous) messages:
  - When S sends message m tagged with 0 <m,0>, it waits to receive <0> (sent by R), otherwise retransmits the <m,0>.
  - When S receives <0>, it transmits the new message tagged by 1.
  - When S receives <1>, it retransmit its message <m,0>.
Parallelism and Communication: Channel Systems (Con.)

- The program graph of the sender S:
Parallelism and Communication: Channel Systems (Con.)

- The program graph of the receiver R:
Parallelism and Communication: Channel Systems (Con.)

- The complete system is presented as a channel system over chan = \{c, d, tmr_on, tmr_off, timeout\} and Var = \{x, y, m_i\} : ABP = [S|Timer|R].

- The behavior of a channel system can be given by a transition system, similar to the mapping from program graphs to transition systems.
Parallelism and Communication: Channel Systems (Con.)

- The states of TS(CS), where CS=\([PG_1|...|PG_n]\) over (Chan,Var), are tuples \(\langle \ell_1, ..., \ell_n, \eta, \xi \rangle\):
  - \(\ell_i\) indicates the current location of component PG\(_i\);
  - \(\eta(\in\text{Eval(Var)})\) keeps track of the current values of the variables;
  - \(\xi, \xi(c)\in\text{dom(c)}^*,\) records the current content of the various channels.
Definition: The transition system of CS, denoted by TS(CS), where CS=[PG_1|...|PG_n] over (Chan, Var) and PG_i= (Loc_i, Act_i, Effect_i, \rightarrow_i, \text{Loc}_{0,i}, g_{0,i}), is defined by TS(CS)=(S, Act, \rightarrow, I, \text{AP}, L):

- S=(Loc_1 \times ... \times Loc_n) \times \text{Eval}(Var) \times \text{Eval}(Chan);
- Act=\bigcup_{0 \leq i \leq n} Act_i \cup \{\tau\};
- I=\{\langle \ell_1, ..., \ell_n, \eta, \xi \rangle \mid \forall 1 \leq i \leq n. (\ell_i \in \text{Loc}_{0,i} \land \eta = g_{0,i})\};
Parallelism and Communication: Channel Systems (Con.)

- \( AP = \bigcup_{1 \leq i \leq n} \text{Loc}_i \cup \text{Cond}(\text{Var}) \);
- \( L(\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle) = \{\ell_1, \ldots, \ell_n\} \cup \{g \in \text{Cond}(\text{Var}) | \eta| = g\} \);
- \( \rightarrow \) is defined by the following rules:
  - **Interleaving for** \( \alpha \in \text{Act}_i \):
    \[
    \ell_i \xrightarrow{g:\alpha} i \ell_i' \]
    \[
    \langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{i} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi \rangle
    \]
    \[
    \eta' = \text{Effect}(\alpha, \eta)
    \]
Parallelism and Communication: Channel Systems (Con.)

- **Asynchronous** message passing for $c \in \text{Chan}$, $\text{cap}(c) > 0$:
  - Receive a value along channel $c$ and assign it to $x$:
    
    \[
    \ell_i \xrightarrow{g:c?} l_i' \land \eta = g \land \text{len}(\xi)(c) = k > 0 \land \xi(c) = v_1\ldots v_k
    \]
    \[
    \langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi' \rangle
    \]
    
    where $\eta' = \eta[x:=v_1]$ and $\xi' = \xi[c:=v_2\ldots v_k]$  
  - Transmit value $v \in \text{dom}(c)$ over channel $c$:
    
    \[
    \ell_i \xrightarrow{g:c!} l_i' \land \eta = g \land \text{len}(\xi)(c) = k < \text{cap}(c) \land \xi(c) = v_1\ldots v_k
    \]
    \[
    \langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta, \xi' \rangle
    \]
    
    where $\xi' = \xi[c:=v_1v_2\ldots v_kv]$
Parallelism and Communication: Channel Systems (Con.)

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

$$
\ell_i \xrightarrow{g_i : c?x} \ell_i' \land \eta\models g_1 \land \eta\models g_2 \land \ell_j \xrightarrow{g_2 : c!v} \ell_j' \land i \neq j
$$

$$
\langle \ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_j', \ldots, \ell_n, \eta', \xi \rangle
$$

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

- See example 2.34.
Parallelism and Communication: Synchronous Parallelism

- In **synchronous** systems components evolve in a **lock step** fashion.

  - Synchronous hardware circuits in which all components are connected to a **central clock** and all perform a step on each clock pulse.
Parallelism and Communication: Synchronous Parallelism (Con.)

**Definition:** The synchronous product $TS_1 \otimes TS_2$, where

- $TS_i = (S_i, \text{Act}, \rightarrow_i, I_i, \text{AP}_i, L_i)$ and
- $*: \text{Act} \times \text{Act} \rightarrow \text{Act}$ be a mapping that assigns $\alpha*\beta$ to each pair of actions $\alpha$ and $\beta$

is given by $(S_1 \times S_2, \text{Act}, \rightarrow, I_1 \times I_2, \text{AP}_1 \cup \text{AP}_2, L)$ where
Parallelism and Communication: Synchronous Parallelism (Con.)

- The transition relation is defined by:

\[
\begin{aligned}
\text{s}_1 &\xrightarrow{\alpha} s_1' \\
\text{s}_2 &\xrightarrow{\beta} s_2'
\end{aligned}
\]

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

- \(\alpha \ast \beta\) denotes the synchronous execution of action \(\alpha\) and \(\beta\).

- \(L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)\).

- It should be noted that compared to parallel operator \(||\), there is no autonomous transition of \(TS_1\) or \(TS_2\).
The state-space Explosion Problem

- Program graphs and channel systems introduced to model data-dependant and communication systems.

- In following we discuss on the cardinality of resulting transition systems, since verification techniques are based on analyzing the TSs.
The state-space Explosion Problem (Con.)

- Program graph representation:
  - Recall that the states of the unfolded TS are of the form $\langle \ell, \eta \rangle$, where location $\ell$ and variable evaluation $\eta$.
  - Assume all variables in Var have finite domain.
  - The number of states in the resulting TS is:
    $$|\text{Loc}| \cdot \prod_{x \in \text{Var}} |\text{dom}(x)|$$
The number of states grows exponentially in the number of variables in the program graph.

Parallelism

In all variant of parallel operators, the state space of the complete system is built of the local state space $S_i$ of the components.
The state-space Explosion Problem (Con.)

- Hence the number of states in $S$ is growing exponentially in the number of components.
- Additionally the size of local state of each component grows exponentially with number of variable.
- Thus the "exponential blowup" in the number of parallel components and number of variables explains the problem.
The state-space Explosion Problem (Con.)

- Channel systems
  - For the size of TS of channel systems, similar observation can be made as for the representation of program graphs.
  - Additionally, the size of channels affects on the number of states.
The state-space Explosion Problem (Con.)

- Thus for $\text{CS}=\left[\text{PG}_1 \mid \ldots \mid \text{PG}_n \right]$ over $\text{Var}=\text{Var}_1 \cup \ldots \cup \text{Var}_n$ and channels $\text{Chan}$, the cardinality of state space is:

\[
\prod_{i=1}^{n} |\text{PG}_i| \cdot \prod_{c \in \text{Chan}} |\text{dom}(c)|^{\text{cap}(c)}
\]

which can be rewritten as:

\[
\prod_{i=1}^{n} \left( |\text{Loc}_i| \cdot \prod_{x \in \text{Var}_i} |\text{dom}(x)| \right) \cdot \prod_{c \in \text{Chan}} |\text{dom}(c)|^{\text{cap}(c)}
\]
Summery

- Transition systems are a fundamental model for modeling software and hardware systems.
- An execution of a TS is an alternating sequence of states and actions that starts in an initial state and that cannot be prolonged.
Interleaving amounts to represent the evolvement of “simultaneous” activities of independent concurrent processes by the nondeterministic choice between these activities.

In case of shared variable communication, parallel composition on the level of transition systems does not faithfully reflect the system’s behavior. Instead, composition on program graphs has to be considered.
Concurrent processes that communicate via handshaking on the set H of actions execute actions outside H autonomously whereas they execute actions in H synchronously.

In channel systems, concurrent processes communicate via FIFO-buffers. Handshaking communication is obtained when channels have capacity 0. For channels with a positive capacity, communication takes place asynchronously.
The size of transition system representations grows exponentially in various components, such as the number of variables in a program graph or the number of components in a concurrent system. This is known as the state-space explosion problem.