Chapter 7: Transition Systems

7.1 Definitions and notations

- The intuition is that a transition system consists of a set of possible states for the system and a set of transitions - or state changes - which the system can effect.

- When a state change is the result of an external event or of an action made by the system, then that transition is labeled with that event or action.

- Particular states or transitions in a transition system can be distinguished.
7.1.1 Transition systems

A \textit{transition systems} is a quadruple $\mathcal{A} = \langle S, T, \alpha, \beta \rangle$ where

- $S$ is a finite or infinite set of \textit{states},
- $T$ is a finite or infinite set of \textit{transitions},
- $\alpha$ and $\beta$ are two mapping from $T$ to $S$ which take each transition $t$ in $T$ to the two states $\alpha(t)$ and $\beta(t)$, respectively the \textit{source} and the \textit{target} of the transition $t$.

A transition $t$ with some source $s$ and target $s'$ is written $t : s \rightarrow s'$. Several transitions can have the same source and target.

A transition system is \textit{finite} if $S$ and $T$ are finite.
Paths

A path of length $n$, $n > 0$, in a transition system $A$ is a sequence of transitions $t_1, t_2, \ldots, t_n$ such that $\forall i : 1 \leq i < n$, $\beta(t_i) = \alpha(t_{i+1})$.

Similarly, an infinite path is an infinite sequence of transitions $t_1, t_2, \ldots, t_n, \ldots$ such that $\forall i : 1 \leq i < n$, $\beta(t_i) = \alpha(t_{i+1})$. 
Labeled transition systems

A transition system labeled by an alphabet \( A \) is a 5-tuple \( \mathcal{A} = \langle S, T, \alpha, \beta, \lambda \rangle \) where

- \( \langle S, T, \alpha, \beta \rangle \) is a transition system,
- \( \lambda \) is a mapping from \( T \) to \( A \) taking each transition \( t \) to its label \( \lambda(t) \).

Intuitively, the label of a transition indicates the action or event which triggers the transition. It is therefore logical to assume that two different transitions cannot have the same source, target and label.
Traces

If \( c = t_1 t_2 \ldots \) is a path in a labeled transition system, the sequence of actions \( \text{trace}(c) = \lambda(t_1)\lambda(t_2)\ldots \) is called the \textit{trace} of the path.
7.1.2 Transition system homomorphisms

**Definition**

Let \( A = \langle S, T, \alpha, \beta \rangle \) and \( A' = \langle S', T', \alpha', \beta' \rangle \) be two transition systems. A *homomorphism* \( h \) from \( A \) to \( A' \) is a pair \((h_\sigma, h_\tau)\) of mappings

\[
h_\sigma : S \rightarrow S', \quad h_\tau : T \rightarrow T',
\]

which satisfies, for every transition \( t \) of \( T \):

\[
\alpha'(h_\tau(t)) = h_\sigma(\alpha(t)), \quad \beta'(h_\tau(t)) = h_\sigma(\beta(t)).
\]
Labeled transition system homomorphisms

Let \( \mathcal{A} = \langle S, T, \alpha, \beta \rangle \) and \( \mathcal{A}' = \langle S', T', \alpha', \beta' \rangle \) be two transition systems labeled by the same alphabet. A labeled transition system homomorphism from \( \mathcal{A} \) to \( \mathcal{A}' \) is a homomorphism \( h \) which also satisfies the condition

\[
\lambda'(h_\tau(t)) = \lambda(t).
\]
The free product of transition systems

Consider $n$ transition systems $\mathcal{A}_i = \langle S_i, T_i, \alpha_i, \beta_i \rangle$, $i = 1, 2, \ldots, n$. The free product $\mathcal{A}_1 \times \mathcal{A}_2 \ldots \mathcal{A}_n$ of those $n$ transition systems is the transition system $\mathcal{A} = \langle S, T, \alpha, \beta \rangle$ defined by

- $S = S_1 \times \ldots \times S_n$,
- $T = T_1 \times \ldots \times T_n$,
- $\alpha(t_1, \ldots, t_n) = \langle \alpha_1(t_1), \ldots, \alpha_n(t_n) \rangle$,
- $\beta(t_1, \ldots, t_n) = \langle \beta_1(t_1), \ldots, \beta_n(t_n) \rangle$. 

If, in addition, each $A_i$ is labeled by an alphabet $A_i$, the free product is a transition system labeled by the alphabet $A_1 \times \ldots A_n$; transitions are labeled as follows:

$$\lambda(t_1, \ldots, t_n) = \langle \lambda_1(t_1), \ldots \lambda(t_n) \rangle.$$
If the transition system $\mathcal{A}$ is in *global state* $s = \langle s_1, \ldots, s_n \rangle$, each component transition system $\mathcal{A}_i$ is in state $s_i$. Each $\mathcal{A}_i$ can independently effect transition $t_i$, changing to state $s'_i$. After having effected the *global transition* $t = \langle t_1, \ldots, t_n \rangle$, the transition system $\mathcal{A}$ changes to global state $s' = \langle s'_1, \ldots, s'_n \rangle$. In the case of labeled transition systems, the vector $\lambda(t)$ is the *global action* that triggered the *global transition* $t$. 
The free product assumes that in a global system, all component systems execute their transitions simultaneously, i.e., it is possible to divide time into intervals in such a way that during each of those intervals each component executes exactly one transition. In other words, the same ‘clock’ drives the different transition systems forming the product.
The synchronous product of transition systems

When processes interact, not all possible global actions are useful, since the interaction is subject to communication and synchronization constraints. The transition system associated with the system of processes must therefore be a subsystem of the free product of the component transition systems. The communication and synchronization constraints that define the subsystem can always be simply expressed by the synchronous product, formally defined below.
If $A_i$, $i = 1, \ldots, n$, $n$ transition systems labeled by alphabets $A_i$, and if $I \subseteq A_1 \times \ldots \times A_n$ is a synchronization constraint, the \textit{synchronous product} of the $A_i$ under $I$, written $\langle A_1, \ldots, A_n, I \rangle$, is the transition system of the free product of the $A_i$ containing only the global transitions $\langle t_1, \ldots, t_n \rangle$ whose vectors of labels $\langle \lambda(t_1), \ldots, \lambda(t_n) \rangle$ are elements of $I$.

In other words, the synchronous product allows only those global transitions corresponding to action vectors included in the synchronization constraint.
Chapter 8: Petri Nets

A Petri net is a four-tuple, $N = (P, T, F, \mu_0)$, where

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places;
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions ($P \cap T = \emptyset$);
- $F \subset (P \times T) \cup (T \times P)$ is the flow relation;
- $\mu_0 \subset P$ is the initial marking of the net.

A marking $\mu$ of $N$ is any subset of $P$. For any transition $t$,

- $\bullet t = \{p \in P | (p, t) \in F\}$ and $t^\bullet = \{p \in P | (t, p) \in F\}$ denote the preset and postset of $t$, respectively. A transition $t$ is enabled in a marking $\mu$ if $\bullet t \subseteq \mu$; otherwise, it is disabled. Let $\text{enabled}(\mu)$ be the set of transitions enabled in $\mu$. 

14
A transition $t$ may fire from marking $\mu$ if and only if the following two conditions hold:

- $t \in enabled(\mu)$, and
- $(\mu - \bullet t) \cap t^\bullet = \emptyset$.

The first condition is the normal firing condition for Petri nets. The second condition requires contact-freeness.
When transition $t$ fires from marking $\mu$, the new marking $\mu'$ is given as follows: $\mu' = (\mu - \bullet t) \cup t^\bullet$.

Note that since we assume contact-freeness, a self-loop will not be enabled.
A run of a Petri net is a finite or infinite sequence of markings and transitions

\[ \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \ldots \]

such that \( \mu_0 \) is the initial marking of the net, \( t_i \in enabled(\mu_i) \) for any \( i \ (i \geq 0) \), and that \( \mu_i = (\mu_{i-1} - \bullet t_{i-1}) \cup t_{i-1}^\bullet \) for any \( i \ (i \geq 1) \).
Time Petri Nets

Time Petri nets are classical Petri Nets where to each transition $t$ a time interval $[a, b]$ is associated. The times $a$ and $b$ are relative to the moment at which $t$ was last enabled. Assuming that $t$ was enabled at time $c$, then $t$ may fire only during the interval $[c + a, c + b]$ and must fire at the time $c + b$ at the latest, unless it is disabled before by the firing of another transition. Firing a transition takes no time.
Thus, the philosophy of this kind of time dependent Petri net is: when a transition becomes enabled it may not fire at once (in general) but during a certain time interval and at the end of the interval there is a force to fire. If the upper bound of the interval is at infinity, then the second characteristic, the obligation to fire, is lost. That is why we consider only time intervals whose upper bounds are finite numbers ($b \neq \infty$).
Let $\mathbb{N}$ be the set of natural numbers. A time Petri net is a six-tuple, $N = (P, T, F, Eft, Lft, \mu_0)$, where

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places;
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions ($P \cap T = \emptyset$);
- $F \subset (P \times T) \cup (T \times P)$ is the flow relation;
- $Eft, Lft : T \to \mathbb{N}$ are functions for the earliest and latest firing times of transitions, satisfying that for any $t \in T$, $Eft(t) \leq Lft(t) < \infty$;
- $\mu \subset P$ is the initial marking of the net.
Let the domain of time values $\mathbf{T}$ be the set of nonnegative real numbers. A state of a time Petri net $N = (P, T, F, Eft, Lft, \mu_0)$ is a pair $s = (\mu, c)$, where $\mu$ is a marking of $N$, and $c : enabled(\mu) \rightarrow \mathbf{T}$ is called the clock function. The initial state of $N$ is $s_0 = (\mu_0, c_0)$ where $c_0(t) = 0$ for any $t \in enabled(\mu_0)$. 
A transition $t$ may fire from state $s = (\mu, c)$ after delay $\delta \in T$ if and only if the following four conditions hold:

- $t \in enabled(\mu)$,
- $(\mu - \bullet t) \cap t^* = \emptyset$,
- $Eft(t) \leq c(t) + \delta$, and
- $\forall t' \in enabled(\mu) : c(t') + \delta \leq Lft(t')$. 
When transition $t$ fires after delay $\delta$ from state $s = (\mu, c)$, the new state $s' = (\mu', c')$ is given as follows:

- $\mu' = (\mu - \cdot t) \cup t^\bullet$, and

- for any $t' \in enabled(\mu')$, if $t' \neq t$ and $t' \in enabled(\mu)$, then $c'(t') = c(t') + \delta$ else $c'(t') = 0$.

This is denoted by $s' = fire(s, (t, \delta))$. 
A run
\[ \rho = s_0 \xrightarrow{(t_0,\delta_0)} s_1 \xrightarrow{(t_1,\delta_1)} \ldots \xrightarrow{(t_{n-1},\delta_{n-1})} s_n \xrightarrow{(t_n,\delta_n)} \ldots \]

of a time Petri net is a finite or infinite sequence of states, transitions, and delays such that \( s_0 \) is the initial state, and for every \( i \geq 1 \), \( s_i \) is obtained from \( s_{i-1} \) by firing a transition \( t_{i-1} \) after delay \( \delta_{i-1} \) which satisfies that \( s_i = fire(s_{i-1}, (t_{i-1}, \delta_{i-1})) \).
Chapter 9: Timed Automata

To express system behaviors with timing constraints, we consider finite graph augmented with a finite set of (real-valued) clocks. The vertices of the graph are called locations, and edges are called switches. While the switches are instantaneous, time can elapse in a location. A clock can be reset to zero simultaneously with any switch. At any instant, the reading of a clock equals the time elapsed since the last time it was reset. With each switch we associate a clock constraint, and require that the switch may be taken only if the current values of the clocks satisfy this constraint. With each location we associate a clock constraint called its invariant, and require that time can elapse in a location only as long as its invariant stays true.
\[
\begin{align*}
    s_0 & \xrightarrow{a} s_1 & x := 0 & x < 1 \\
    s_1 & \xrightarrow{b} s_2 & y := 0 & x < 1 \\
    s_2 & \xrightarrow{c} s_3 \\
    s_0 & \xrightarrow{d, y > 2} s_3
\end{align*}
\]
Clock Constrains

To define timed automata formally, we need to say what type of clock constraints are allowed as invariants and enabling conditions. An atomic constraint compares a clock value with a time constant, and a clock constraint is a conjunction of atomic constraints. Any value from \( Q \), the set of nonnegative rationals, can used as a time constant.

Formally, for a set \( X \) of clock variables, the set \( \Phi(X) \) of clock constraints \( \phi \) is defined by the grammar

\[
\phi := x \leq c \mid c \leq x \mid x < c \mid c < x \mid \phi_1 \land \phi_2,
\]

where \( x \) is a clock in \( X \) and \( c \) is a constant in \( Q \).
Clock Interpretations

A *clock interpretation* \( v \) for a set \( X \) of clocks assigns a real value to each clock; that is, it is a mapping from \( X \) to the set \( R \) of nonnegative reals.

We say that a clock interpretation \( v \) for \( X \) satisfies a clock constraint \( \phi \) over \( X \) iff \( \phi \) evaluates to true according to the values given by \( v \).

For \( \delta \in R \), \( v + \delta \) denotes the clock interpretation which maps each clock \( x \) to the value \( v(x) + \delta \).

For \( Y \subseteq X \), \( v[Y := 0] \) denotes the clock interpretation for \( X \) which assigns 0 to each \( x \in Y \), and agree with \( v \) over the reset of the clocks.
A timed automaton is a tuple \( \langle L, L^0, \Sigma, X, I, E \rangle \), where

- \( L \) is a finite set of locations,
- \( L^0 \subseteq L \) is a set of initial locations,
- \( \Sigma \) is a finite set of labels,
- \( X \) is a finite set of clocks,
- \( I \) is a mapping that labels each location \( s \) in \( L \) with some clock constraint in \( \Phi(X) \), and
- \( E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L \) is a set of switches.
The semantics of a timed automaton $A$ is defined by associating a transition system $S_A$ with it.

A state of $S_A$ is pair $(s, v)$ such that $s$ is a location of $A$ and $v$ is a clock interpretation for $X$ such that $v$ satisfies the invariant $I(s)$. A state $(s, v)$ is an initial state if $s$ is an initial location of $A$ and $v(x) = 0$ for all clocks $x$.

There are two types of transitions in $S_A$:

- State can change due to elapse of time: for a state $(s, v)$ and a real-valued time increment $\delta \geq 0$, $(s, v) \xrightarrow{\delta_0} (s, v + \delta)$ if for all $0 \leq \delta' \leq \delta$, $v + \delta'$ satisfies the invariant $I(s)$.

- State can change due to a location-switch: for a switch $\langle s, a, \varphi, \lambda, s' \rangle$ such that $v$ satisfies $\varphi$, $(s, v) \xrightarrow{a} (s', v[\lambda := 0])$. 

32
Product constructions

Let $A_1 = \langle L_1, L_1^0, \Sigma_1, X_1, I_1, E_1 \rangle$ and $A_2 = \langle L_2, L_2^0, \Sigma_2, X_2, I_2, E_2 \rangle$ be two timed automata. Assume that the clock sets $X_1$ and $X_2$ are disjoint. Then, the product, denoted $A_1 \parallel A_2$, is the timed automaton

$\langle L_1 \times L_2, L_1^0 \times L_2^0, \Sigma_1 \cup \Sigma_2, X_1 \cup X_2, I, E \rangle$ where

$I(s_1, s_2) = I(s_1) \land I(s_2)$ and the switches are defined by:

1. for $a \in \Sigma_1 \cap \Sigma_2$, for every $\langle s_1, a, \varphi_1, \lambda_1, s_1' \rangle$ in $E_1$ and $\langle s_2, a, \varphi_2, \lambda_2, s_2' \rangle$ in $E_2$, $E$ contains $\langle (s_1, s_2), a, \varphi_1 \land \varphi_2, \lambda_1 \cup \lambda_2, (s_1', s_2') \rangle$.

2. for $a \in \Sigma_1 \setminus \Sigma_2$, for every $\langle s, a, \varphi, \lambda, s' \rangle$ in $E_1$ and every $t$ in $L_2$, $E$ contains $\langle (s, t), a, \varphi, \lambda, (s', t) \rangle$.

3. for $a \in \Sigma_2 \setminus \Sigma_1$, for every $\langle s, a, \varphi, \lambda, s' \rangle$ in $E_2$ and every $t$ in $L_1$, $E$ contains $\langle (t, s), a, \varphi, \lambda, (t, s') \rangle$. 


\[ (s, t) \quad a, \ x := 0 \quad (s', t) \quad c, \ y \geq 1 \]
\[ (s', t') \quad x \leq 2, \ y \leq 2 \quad (s, t') \quad a, \ x := 0 \]
\[ b, \ x \geq 1 \quad x \geq 1, \ b, \ y := 0 \]
\[ (s', t) \quad x \leq 2 \quad (s, t') \quad y \leq 2 \]
Chapter 10: Message Sequence Charts

Message sequence chart (MSC) is a graphical and textual language for the description and specification of the interactions between system components. The main area of application for MSCs is as overview specification of the communication behavior of real-time systems, in particular telecommunication switching systems.

MSCs represent typical execution scenarios, providing examples of either normal or exceptional executions of the proposed system. The MSC standard as defined by ITU-T in Recommendation Z.120 introduces two basic concepts: basic MSCs and High-Level MSCs.
Basic MSCs

A basic MSC describes exactly one scenario, which consists of a set of processes that run in parallel and exchange messages in a one to one, asynchronous fashion.
100 \leq e_{13} - e_1 < \infty
0 \leq 2(e_{13} - e_{12}) - (e_{13} - e_1) < \infty
The semantics of a bMSC essentially consists of sequences (of traces) of messages that are sent and received among the concurrent processes in the bMSC. The order of communication events (i.e. message sending or receiving) in a trace is deduced from the visual partial order determined by the flow of control within each process in the bMSC along with a causal dependency between the events of sending and receiving a message.
Each bMSC corresponds to a visual order for a pair of events $e_1$ and $e_2$ such that $e_1$ precedes $e_2$ in the following cases:

- **Causality**: A sending event $e_1$ and its corresponding receiving event $e_2$.

- **Controlability**: The event $e_1$ appears above the event $e_2$ on the same process line, and $e_2$ is a sending event. This order reflects the fact that a sending event can wait for other events to occur. On the other hand, we sometimes have less control on the order in which receiving events occur.

- **Fifo order**: The receiving event $e_1$ appears above the receiving event $e_2$ on the same process line, and the corresponding sending events $e'_1$ and $e'_2$ appear on a mutual process line where $e'_1$ is above $e'_2$. 
A bMSC is a tuple $D = (P, E, M, L, V)$ where

- $P$ is a finite set of processes;
- $E$ is a finite set of events corresponding to sending a message and receiving a message.
- $M$ is a finite set of messages. Each message in $M$ is of the form $(e, g, e')$ where $e, e' \in E$ corresponds to sending and receiving the message respectively, and $g$ is the message name which is a character string.
- $L : E \rightarrow P$ is labelling function which maps each event $e \in E$ to a process $L(e) \in P$.
- $V$ is a finite set whose elements are a pair $(e, e')$ where $e, e' \in E$ and $e$ precedes $e'$, which is corresponding to a visual order;
We use *event sequences* to represent the *traces* of bMSCs which are corresponding to the untimed behavior of bMSCs. Any event sequence is of the form $e_0 \hat{e} e_1 \hat{e} \ldots \hat{e} e_m$, which represents that $e_{i+1}$ takes place after $e_i$ for any $i$ ($0 \leq i \leq m - 1$).

Let $D = (P, E, M, L, V, C)$ be a bMSC. An event sequence $e_0 \hat{e} e_1 \hat{e} \ldots \hat{e} e_m$ is a *trace* of $D$ if and only if the following conditions hold:

- all events in $E$ occur in the sequence, and each event occurs only once, i.e. $\{e_0, e_1, \ldots, e_m\} = E$ and $e_i \neq e_j$ for any $i, j$ ($0 \leq i < j \leq m$); and
- $e_1, e_2, \ldots, e_m$ satisfy the visual order defined by $V$, i.e. for any $e_i$ and $e_j$, if $(e_i, e_j) \in V$, then $0 \leq i < j \leq m$. 


MSC Specifications

For describing multiple scenarios, MSC specifications are introduced, which are a combination of a set of basic MSCs and a high-Level MSC describing their compositions.
A MSC specification (MSS) is a tuple $S = (U, N, \text{succ}, \text{ref})$ where

- $U$ is a finite set of bMSCs satisfying that for any $D = (P, E, M, L, V)$ and $D' = (P', E', M', L', V')$ in $U$, if $D \neq D'$, then $E \cap E' = \emptyset$;

- $N = \{\top\} \cup I \cup \{\bot\}$ is a finite set of nodes partitioned into the three sets: the singleton-set of start node, the set of intermediate nodes, and the singleton-set of end node, respectively;

- $\text{succ} \subset N \times N$ is the relation which reflects the connectivity of the nodes in $N$ such that any node in $N$ is reachable from the start node; and

- $\text{ref} : I \mapsto U$ is a function that maps each intermediate node to a bMSC in $U$. 

According to the synchronous interpretation of the concatenation of two bMSCs in a MSS, we define the behavior of a MSS $S$ as the event sequences which are the concatenation of the event sequences representing the behavior of the bMSCs which make up $S$.

The synchronous interpretation of the concatenation of two bMSCs in a MSS means that when moving one node to the other, all events in the previous bMSC finish before any event in the following bMSC occurs, which is closed to the visual structure of the MSS and may be closed to the behavior of the system that the designer of the MSS has in mind.
Part I: Introduction
Chapter 0: Preliminaries
Chapter 1: Language and Computation

Part II: Models
Chapter 2: Finite Automata
Chapter 3: Regular Expressions
Chapter 4: Context-Free Grammars
Chapter 5: Pushdown Automata
Chapter 6: Turing Machines
Part III: Modelling

Chapter 7: Transition Systems

Chapter 8: Petri Nets, Time Petri Nets

Chapter 9: Timed Automata, Hybrid Automata

Chapter 10: Message Sequence Charts