Embedded Systems Specification and Design
Model-based Design and Verification

David Kendall
What behaviours is the system capable of?
Timed automata composed into a network of timed automata consisting of $n$ TA's $A_i = (L_i, \ell_i^0, C, A, E_i, I_i)$, $1 \leq i \leq n$.

Assume a common set of clocks and actions.

A location vector is a vector $\vec{\ell} = (\ell_1, \ldots, \ell_n)$.

We compose the invariant functions into a common function over location vectors $I(\vec{\ell}) \equiv \bigwedge_i l_i(\ell_i)$.

We write $\vec{\ell}[\ell'_i/\ell_i]$ to denote the vector where the $i$th element $\ell_i$ of $\vec{\ell}$ is replaced by $\ell'_i$. 
Parallel Composition

Gives the meaning of a system comprising several components.

**Definition (Network of TA Semantics)**

Let $A_i = (L_i, \ell^0_i, C, A, E_i, I_i)$ be a network of $n$ timed automata. Let $\bar{\ell}^0 = (\ell^0_1, \ldots, \ell^0_n)$ be the initial location vector. The semantics is defined as a transition system $(S, s^0, L, \rightarrow)$, where

- $S = (L_1 \times \cdots \times L_n) \times \mathbb{R}_{\geq 0}^C$ is the set of states,
- $s^0 = (\bar{\ell}^0, 0_C)$ is the initial state,
- $L = \mathbb{R}_{\geq 0} \cup A$ is the set of labels, and
- $\rightarrow \subseteq S \times S$ is the transition relation defined by the rules for
  - Time Progress (TP)
  - Independent Action (IA), and
  - Synchronising Action (SA)
Transition Rules for Parallel Composition

\[ \text{TP} \ (\bar{\ell}, v) \xrightarrow{d} (\bar{\ell}, v + d), \ \text{if} \ \forall d' : 0 \leq d' \leq d \Rightarrow v + d' \models I(\bar{\ell}) \]
Transition Rules for Parallel Composition

TP \((\bar{\ell}, v) \xrightarrow{d}(\bar{\ell}, v + d), \) if \(\forall d': 0 \leq d' \leq d \Rightarrow v + d' \models I(\bar{\ell})\)

IA \((\bar{\ell}, v) \xrightarrow{\tau}(\bar{\ell}[\ell'_i/\ell_i], v')\) if there exists \((\ell_i, g, \tau, r, \ell'_i) \in E_i\) such that \(v \models g, v' = v[r],\) and \(v' \models I(\bar{\ell}[\ell'_i/\ell_i])\)
Transition Rules for Parallel Composition

**TP** \( (\ell, v) \xrightarrow{d} (\ell, v + d) \), if \( \forall d' : 0 \leq d' \leq d \Rightarrow v + d' \models I(\ell) \)

**IA** \( (\ell, v) \xrightarrow{\tau} (\ell[\ell'_i/\ell_i], v') \) if there exists \( (\ell_i, g, \tau, r, \ell'_i) \in E_i \) such that \( v \models g \), \( v' = v[r] \), and \( v' \models I(\ell[\ell'_i/\ell_i]) \)

**SA** \( (\ell, v) \xrightarrow{c} (\ell[\ell'_i/\ell_i, \ell'_j/\ell_j], v') \) if there exists \( (\ell_i, g_i, c?, r_i, \ell'_i) \in E_i \) and \( (\ell_j, g_j, c!, r_j, \ell'_j) \in E_j \) such that \( v \models (g_i \land g_j) \), \( v' = v[r_i \cup r_j] \), and \( v' \models I(\ell[\ell'_i/\ell_i, \ell'_j/\ell_j]) \)
Construct a (finite prefix of a) behaviour of the parallel composition of the *Lamp* | *User* system
Want to check formal model to see if it has specified properties.
Interested in both safety and liveness properties

Safety property
- Nothing bad ever happens
  E.g. the train is never in the crossing when the gate is open

Liveness property
- Something good eventually happens
  E.g. whenever the gate is closed, it is eventually opened again
How to specify properties of a TA

- **State properties**
  Simple boolean formulas that can be checked with respect to a single state

- **Test automata**

- **Real-time temporal logic**
  - Allow the expression of properties that concern executions (paths), i.e. sequences of states
Construct a TA $A_s$ that acts as an observer of the model $A_m$.
Usually the observer TA includes one special error location.
The property is tested by checking that the observer can never reach its error location in the composition $A_m | A_s$.

**Good:**
- Can use simple reachability analysis to test complex properties.

**Not so good:**
- May need to modify model in order to allow observation.
- Ad hoc specification may not be correct.
Test Automaton Example

- **Check that observer is never SAD**
- **Requires change to GATE model to allow observation**
- **Check the property for a variety of values of LIMIT**
Uppaal’s Specification Language

- A simple real-time temporal logic
- Like LTL but with path quantifiers and predicates on clock variables to capture real-time properties
- No nested temporal operators
- Kept simple deliberately so that properties can be decided by reachability testing
- Simple, efficient implementation of verification procedure
Definition of the Specification Language

- Simple state properties

  - Location assertions
    \[ P.\ell \]
  - Process \( P \) is in location \( \ell \)
    E.g. Gate.CLOSED, Train.IN, Observer.SAD, etc.
  - deadlock
  - Clock constraints
    \[ \text{ID REL NAT | ID REL ID | ID REL ID + NAT} \]
    \[ | \text{ID REL ID - NAT} \]
    E.g. \( x \geq 3, x > y, x \leq y + 4, x = y - 2 \)
Assume $AP$ is the set of simple state properties

The set $SP$ of state properties can be expressed as:

$$SP ::= AP \mid \text{not } SP \mid ( \text{SP} ) \mid \text{SP or SP} \mid \text{SP and SP} \mid \text{SP imply SP}$$

E.g. not Train.IN, Gate.CLOSED or Gate.OPEN, Train.OUT and $x \leq 5$, Gate.OPEN imply not TRAIN.IN, deadlock
Path properties

\[ \text{Prop ::= } A[\cdot] \quad SP \quad - \text{ all paths always} \]
\[ | \quad E<> \quad SP \quad - \text{ some path eventually} \]
\[ | \quad E[\cdot] \quad SP \quad - \text{ some path always} \]
\[ | \quad A<> \quad SP \quad - \text{ all paths eventually} \]
\[ | \quad SP \quad --> \quad SP \quad - \text{ leads to} \]

E.g. \( A[\cdot] \) not deadlock, \( E<> \) Gate.OPEN

Each property in UPPAAL must be expressed as a path property

N.B. \( P \rightarrow Q \) is equivalent to \( A[\cdot] \) (P imply A<> Q)
But UPPAAL doesn’t allow nested path quantifiers in general
All paths

\[ A[]_\phi \]

\[ A<>_\phi \]
Some path

$E[]\phi$

$E<>\phi$
Leads to

$\psi \sim \phi$
Example properties

- A[] not deadlock
  - On all executions, in every state, the property not deadlock is true

- E<> Train.In
  - On some execution, in some state, the train is in the crossing

- A[] (Train.In imply Gate.Closed)
  - On all executions, in every state, if the train is in the crossing, the gate is closed

- Gate.Closed $\rightarrow$ Gate.Open and (g $\leq$ 30)
  - Whenever the gate is closed, it is eventually opened within 30 time units (assumes g is global clock which is reset on entry to Gate.Closed)