Embedded Systems Specification and Design

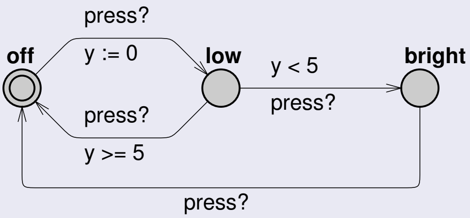
David Kendall

Introduction

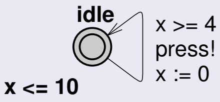
- Modelling a system as a *network* of timed automata
- Semantics of parallel composition
- Specifying real-time properties
  - Test automata
  - Uppaal's property specification language

Network of Timed Automata

Lamp



User



What behaviours is the system capable of?

Parallel Composition: Preliminaries

- Timed automata composed into a network of timed automata consisting of n TA's $A_i = (L_i, l_i^0, C, A, E_i, \mathcal{I}_i)$, $1 \leq i \leq n$.
- Assume a common set of clocks and actions
- A location vector is a vector $\bar{l} = (l_1, \ldots, l_n)$.
- We compose the invariant functions into a common function over location vectors $\mathcal{I}(\bar{l}) \mathrel{\widehat{=}} \wedge_i \mathcal{I}_i(l_i)$.
- We write $\bar{l}[l_i'/l_i]$ to denote the vector where the ith element l_i of $\bar{l}$ is replaced by l_i' .

Parallel Composition

Gives the meaning of a system comprising several components.

Definition (Network of TA Semantics)

Let $A_i = (L_i, l_i^0, C, A, E_i, \mathcal{I}_i)$ be a network of n timed automata. Let $\bar{l}_0 = (l_1^0, \ldots, l_n^0)$ be the initial location vector. The semantics is defined as a transition system $(S, s_0, \rightarrow)$, where $S = (L_1 \times \cdots \times L_n) \times \mathbb{R}^C$ is the set of states, $s_0 = (\bar{l}_0, u_0)$ is the initial state, 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)

as follows.

Transition Rules for Parallel Composition

TP $(\bar{l}, u) \xrightarrow{d} (\bar{l}, u + d)$, if $\forall\, d' : 0 \leq d' \leq d \Rightarrow u + d' \in \mathcal{I}(\bar{l})$

Transition Rules for Parallel Composition

TP $(\bar{l}, u) \xrightarrow{d} (\bar{l}, u + d)$, if $\forall\, d' : 0 \leq d' \leq d \Rightarrow u + d' \in \mathcal{I}(\bar{l})$

IA $(\bar{l}, u) \xrightarrow{\tau} (\bar{l}[l_i'/l_i], u')$ if there exists $(l_i, \tau, g, r, l_i') \in E_i$ such that $u \in g$, $u' = [r \mapsto 0]u$, and $u' \in \mathcal{I}(\bar{l}[l_i'/l_i])$

Transition Rules for Parallel Composition

TP $(\bar{l}, u) \xrightarrow{d} (\bar{l}, u + d)$, if $\forall\, d' : 0 \leq d' \leq d \Rightarrow u + d' \in \mathcal{I}(\bar{l})$

IA $(\bar{l}, u) \xrightarrow{\tau} (\bar{l}[l_i'/l_i], u')$ if there exists $(l_i, \tau, g, r, l_i') \in E_i$ such that $u \in g$, $u' = [r \mapsto 0]u$, and $u' \in \mathcal{I}(\bar{l}[l_i'/l_i])$

SA $(\bar{l}, u) \xrightarrow{c} (\bar{l}[l_i'/l_i, l_j'/l_j], u')$ if there exists $(l_i, c?, g_i, r_i, l_i') \in E_i$ and $(l_j, c!, g_j, r_j, l_j') \in E_j$ such that $u \in (g_i \wedge g_j)$, $u' = [r_i \cup r_j \mapsto 0]u$, and $u' \in \mathcal{I}(\bar{l}[l_i'/l_i, l_j'/l_j])$

Property Specification

- 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

Test Automata

- 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 $A_m \mid A_s$
- 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

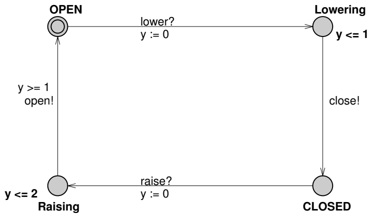


Figure: GATE Automaton

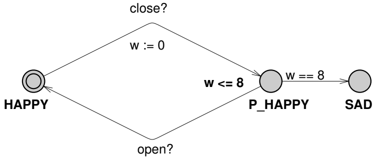


Figure: OBSERVER Automaton

- Check that observer is never SAD
- Requires change to GATE model to allow observation

Uppaal's Specification Language

- A simple real-time temporal logic
- Like LTL but with real-time values and path quantifiers
- 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
    $P.l$
  - Process P is in location l
    E.g. Gate.CLOSED, Train.IN, etc.
  - deadlock
  - Clock constraints
    ID REL NAT | ID REL ID | ID REL ID + NAT
    | ID REL ID - NAT
    E.g. x >= 3, x > y, x <= y + 4, x == y - 2

Definition of the Specification Language ctd

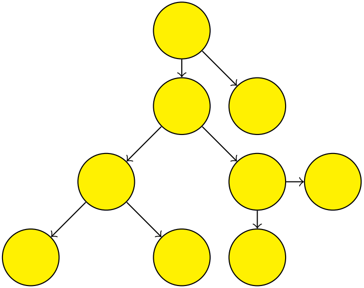
- Assume *AP* is the set of simple state properties
- The set *SP* of state properties can be expressed as
  SP ::= AP | not SP | (SP) | SP or SP | SP and SP
  | SP imply SP
  E.g. not Train.IN, Gate.CLOSED or Gate.OPEN, Train.OUT and x
  <= 5, Gate.OPEN imply not TRAIN.IN, deadlock

Definition of the Specification Language ctd

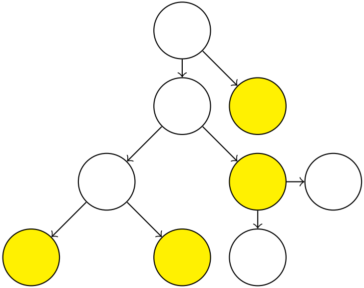
- Path properties
- ```
  Prop ::= A[] SP        - all paths always
         | E<> SP        - some path eventually
         | E[] SP        - some path always
         | A<> SP        - all paths eventually
         | SP --> SP     - leads to
  ```
- E.g. A[] not deadlock, E<> Gate.OPEN
- Each property in UPPAAL must be expressed as a path property
- N.B. P –> Q is equivalent to A[] (P imply A<> Q)
  But UPPAAL doesn't allow nested path quantifiers in general
  ```

All paths

A[]ϕ

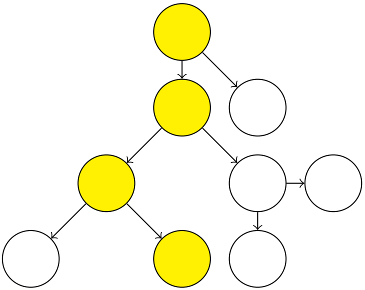


A<>ϕ



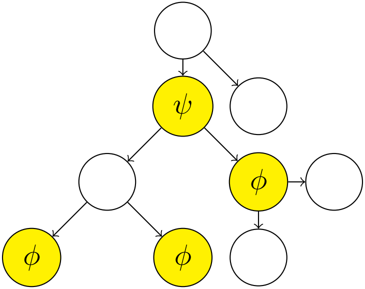
Some path

$E[]\phi$ $E<>\phi$



Leads to

$$\psi \leadsto \phi$$



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 –> Gate.Open and (g <= 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)