Specification models and their analysis
Lecture 3: Timed Automata

Kai Lampka

December 20, 2010

Institut für Technische Informatik und Kommunikationsnetze

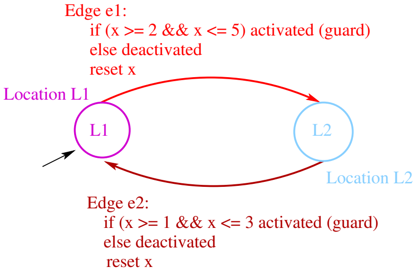
Part I

Basics – What is a Timed Automata?

Timed Automata

Intuition:

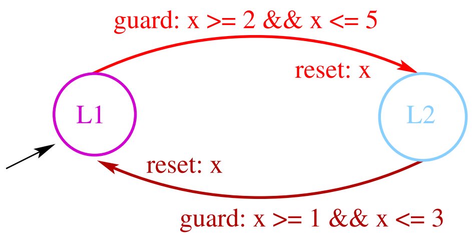
A Timed Automaton (Alur & Dill 90) is a finite state machine equipped with *clocks running all at the same speed*. Transitions (called edges) between states (called locations) are activated/deactivated according to the value currently held by each clock. When changing location (via traversal of an activated edge) clocks are reset.



Ingredients of TA:

- clocks and clock constants,
- locations equipped with labels, and
- edges equipped with labels, clock constraints (guards) and clock resets.

Clock Constraints



Atomic clock constraints:

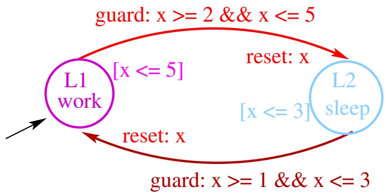
- An atomic clock constraint g_a has to be of the form $g_a := x \bowtie k$, where $\bowtie \in \{<, \leq, >, \geq, =\}$ and x is a clock and $k \in \mathbb{N}_0$ a (clock) constant.
- $\mathcal{CC}$ denotes the set of atomic clock constraints of a TA.
- E.g. $\mathcal{CC} := \{x \leq 1, x \leq 2, x \leq 3, x \leq 5\}$

- A complex clock constraint g_c of a TA is constructed by the following grammar:
$$g_c := g_a \in \mathcal{CC} | g_c \wedge g_c | \neg g_c$$
- A clock valuation is a function $\mu : \mathcal{C} \to \mathbb{R}_+$ which assigns a real-valued number to a clock yielding the respective satisfaction relation for clocks, and atomic, resp. complex clock constraints: $x < k \models \mathit{true} \Leftrightarrow \mu_x < k$, etc.

Formal Definition

A TA is a tuple $(Loc, l_0, \mathcal{A}ct, \mathcal{C}, \hookrightarrow, Inv, \mathcal{L})$, where

- Loc is the finite set of locations, with location l_0 is initial one.
- $\mathcal{A}ct$ is a set of event-labels.
- $\mathcal{C}$ is a finite set of real-valued clocks.
- $Inv : Loc \to \mathcal{CC}$
- $\mathcal{L} : Loc \to \Lambda$ is a mapping that assigns labels to locations.
- $\hookrightarrow \subseteq Loc \times \mathcal{CC}(\mathcal{C}) \times \mathcal{A}ct \times 2^{\mathcal{C}} \times Loc$ is an edge relation.



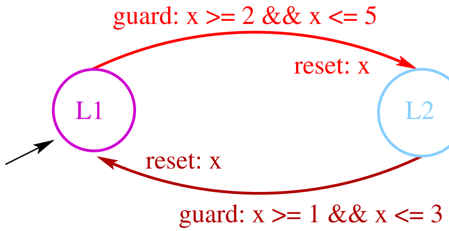
- $Loc = \{L_1, L_2\}$, $l_0 := L_1$, $\mathcal{A}ct := \{e_1, e_2\}$ $\mathcal{C} := \{x\}$
- $\mathcal{CC} := \{x \leq 1, x \leq 2, x \leq 3, x \leq 5\}$
- $Inv : \{L_1 \to (x \leq 5); L_2 \to (x \leq 3)\}$
- $\mathcal{L} := \{L_1 \to work; L_2 \to sleep\}$
- $\hookrightarrow := \{\xrightarrow{e_1}, \xrightarrow{e_2}\}$

Remarks:

- **Edge relation**: Elements of this relation are directed edges connecting pairs of locations. Commonly an edge carries a clock constraint $g_c \in \mathcal{CC}$. The edge-specific constraints, also denoted as guards, must evaluate to true once the edge should be traversed; in such cases we say that the respective edge is enabled. The power set $2^{\mathcal{C}}$ in the above definition refers to the fact that upon edge traversal a subset of clocks is reset to zero and clocks outside this subset maintain their values.
- **Reset of clocks upon edge traversals**: With $\mu : \mathcal{C} \to \mathbb{R}_+$ we refer to real-valued clock evaluations. Notation $\mu' = [\mathcal{R} \to 0]\mu$ denotes that the clocks of set $\mathcal{R} \subseteq \mathcal{C}$ are set to 0, and the remaining ones $(\mathcal{C} \setminus \mathcal{D})$ maintain their values.

Timed Automata

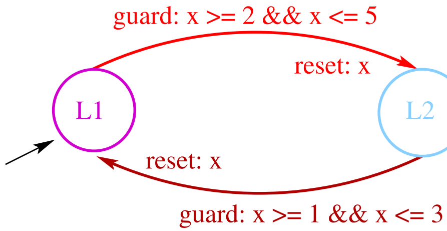
How do we interpret a TA?



Note: Activated edges can be executed, i. e., they do not have to be executed!

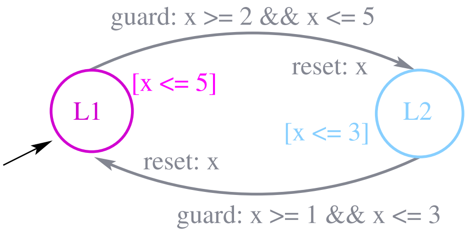
Timed Automata

How do we interpret a TA?



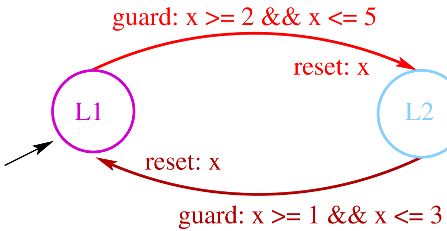
Note: Activated edges can be executed, i. e., they do not have to be executed!

For enforcing behavior we employ location invariants. This are clock constraints defined for locations.



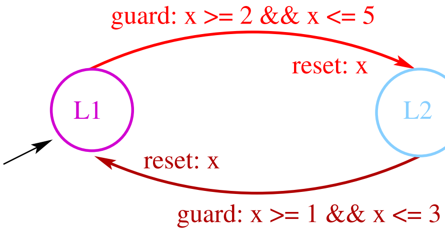
Timed Automata

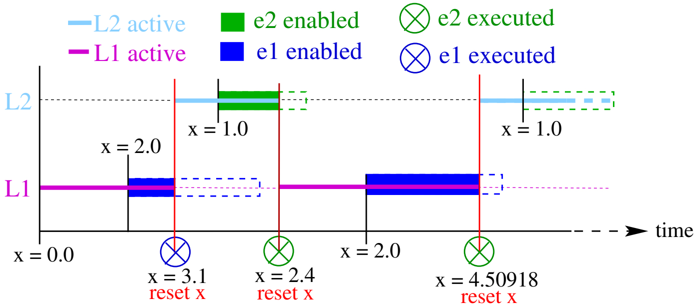
How do we interpret a TA?



1. Progress of time (delay transitions) : we are allowed to increase the values of the clocks, as long as no location invariant is violated.
2. Location change (discrete transitions): An enabled edge can be traversed at any time, i. e., as long as it guards evaluated to true and the invariants of the target location(s) are satisfied. When changing locations some clocks might be reset, i. e., they evaluate to 0 right after.

Timed Automata





Timed Automata: Operational semantic

Remark: State layout:

- Let a state of a TA be the tuple $\langle l, \mu, \rangle$ where l is the currently active location, and
- μ is a valuation of **all** clocks.

Delay transition (progress of time):

$$\frac{\textit{empty}}{\langle l, \mu\rangle \xrightarrow{\delta} \langle l, \mu + \delta\rangle} \quad \left(\begin{array}{c} \delta, \delta' \in \mathbb{R}_+ \text{ with } 0 \leq \delta' \leq \delta \text{and} \\ l \;:\; \mu + \delta' \models \mathit{Inv}(l) \end{array} \right)$$

Informal: One may advance the clock values as long as the location invariants of the active locations are satisfied at all time up to the new time $\mu + \delta$.

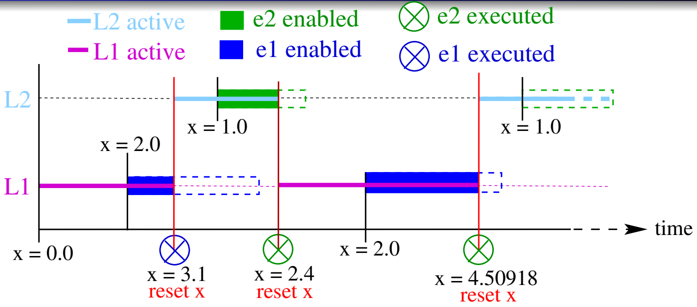
Timed Automata: Operational semantic

Discrete transitions (location change):

$$\frac{\left(l_i \xhookrightarrow{\mathcal{G}_a, a, \mathcal{R}} l_k\right) \in TA}{\langle l, \mu\rangle \xrightarrow{a} \langle l', \mu'\rangle} \quad \left(\begin{array}{ll} l_i \text{ contained in } l & (1) \\ \mu \models \mathcal{G}_a, & (2) \\ \mu' = [\mathcal{R} \to 0]\mu, & (3) \\ l' : \mu' \models Inv(l') & (4) \end{array}\right)$$

Informal: The side conditions from above refer to the fact that an edge is enabled. Informally this means: edge a is enabled in a state $\langle l, \mu\rangle$ if its preceding location l_i is marked active (cond. (1)), the clock evaluation μ satisfy the clock constraints (guard) $\mathcal{G}_a$ (cond. (2)), the successor state $\langle l', \mu'\rangle$ contains the clock resets of the clocks of $\mathcal{R}$ (cond. (3)), and the invariants of newly activated locations (l') hold for μ' (cond. (4)) .

Timed Automata



$$\langle L_1, 0\rangle \xrightarrow{3.1} \langle L_1, 3.1\rangle \xrightarrow{e_1} \langle L_2, 0\rangle \xrightarrow{2.4} \langle L_2, 2.4\rangle \xrightarrow{e_1} \langle L_1, 0\rangle \cdots$$

where such a sequence is called execution trace.

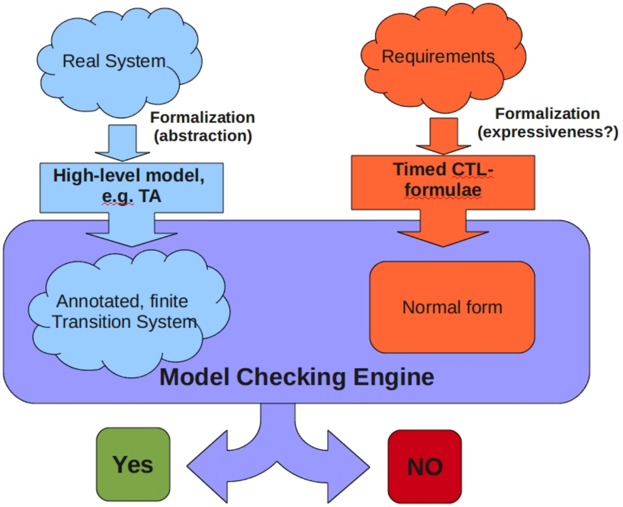
With positive guard-clock-evaluations from dense intervals $[a, b]$ one may construct infinitely many system states, here all states of the kind $\langle L_1, x \in \mathcal{I}_x\rangle$ and $\langle L_2, x \in \mathcal{I}_x)\rangle$, where $\mathcal{I}_x = [0, \infty)$ (Why up to ∞?)

Timed Automata

- A TA can be expanded into a **finite** transition system, denoted as region graph.
- The region graph is a symbolic representation of a TA's behavior; symbolic means that it does not consists of individual system states $\langle L, \text{time stamp}\rangle$, but groups these states into finitely many representatives, i. e., equivalence classes.
- As the region graph is finite and a complete representation of a TA behavior timed CTL-model checking on TA is decidable.

In this lecture we will not elaborate on these very sophisticated details.

Summary


Real System
Formalization (abstraction)
High-level model, e.g. TA
Requirements
Formalization (expressiveness?)
Timed CTL-formulae
Annotated, finite Transition System
Normal form
Model Checking Engine
Yes
NO

Part II

Stating Properties

Stating properties: Queries

There exists a timed version of CTL, but this will not be part of this lecture. We simply employ reachability queries as known from CTL, where we stick to the syntax of the Uppaal timed model checker:

1. **Possibly p**: The statement $\boxed{E <> p}$ evaluates to true for a timed transition system of a TA $\mathcal{T}$ *iff* there is a path $\Pi(s_0) := s_0 \xrightarrow{\mathcal{T}} s_1 \xrightarrow{\mathcal{T}} \ldots s_n$, of alternated delay and action transitions where system configuration s_n satisfies state property p. E.g. $\mathcal{T} \models E <> (buffer > B)$ is true *iff* we reach a system configuration where variable *buffer* is larger than constant B.

2. **Invariantly p**: The statement $\boxed{A[\,]p}$ evaluates to true *iff* every reachable state as contained in the timed transition system of a TA $\mathcal{T}$ satisfy state property p. E.g. $\mathcal{T} \models A[\,](buffer < B)$ is true *iff* in all system configurations variable *buffer* is smaller than constant B.

Stating properties: Observers

With so called observer TA and a respective reachability query one may validated complex properties of a modelled system.

Observer:
An observer is a TA which is executed in parallel for flagging the validity or violation of a property. By explicitly exploiting the non-deterministic choice between edge execution one is enabled to validate complex properties.

$\longrightarrow$ Example 2.1: Observer