Implementability of Timed Automata

Karine Altisen (Verimag),

Joint work with:
Nicolas Markey (LSV),
Pierre-Alain Reynier (LSV),
Stavros Tripakis (Verimag)
Context: Model-based Design

- Models are good for analysis:
  - simulation, testing, theorem proving, verification...

- What about implementation
  - currently mostly an art/practice

- How to move from models to implementation?
  - as automatically as possible,
  - preserving as much as possible
Timed Automata: definition

- finite automaton
- real-valued clocks: $x$
- triggering conditions on transitions:
  - guards: $x = 1$ and resets: $x := 0$
  - inputs?: $a?$ + outputs!: $b!$
- condition on states: invariants: $x \leq 1$
Timed Automata: semantics

Example of trace:

$\text{(state } = 1, x = 0) \rightarrow (\text{state } = 1, x = 0.88)$

$\rightarrow ?a \rightarrow (\text{state } = 2, x = 0) \rightarrow (\text{state } = 2, x = 0.45)$

$\rightarrow b! \rightarrow (\text{state } = 1, x = 0.45) \rightarrow (\text{state } = 1, x = 54.3)$

$\rightarrow a? \rightarrow (\text{state } = 2, x = 0) \rightarrow (\text{state } = 2, x = 1)$

$\rightarrow (\text{state } = 3, x = 1) \ldots$
Timed Automata: semantics

Comments:

• the *clocks* are infinitely precise
  guards are tested against exact values

• the *computation* takes zero time
  (evaluation of guards, change of discrete states)

• the *communication* with outside takes zero time
  (inputs/outputs)

→ a *model* with *ideal* semantics
Towards a Realistic Platform

we consider that a realistic platform should specify:

• how precise are the clocks (they should be digital!) and how they are related

• speed, frequency and precision of computations

• how inputs and outputs are treated
  • w.r.t. environment and shared variables (if some)
  • w.r.t. time
Guaranties

- Property Preservation

- “Faster is better” property
  - “implem + platform” satisfies a property
  - change for a “more performant” platform,
  - is the property still satisfied?
Approaches

Two ways to take into account the imprecision due to implementation:

- Model it within a model of the execution platform KA+ST (Verimag)

- Adapt the semantics of timed automata to include imprecision Raskin et al. (ULB) and then PB+NM+PAR (LSV)
Approach 1: models the exec. platform

- Idea: translate the TA into a program and model the execution platform as timed automata

- Global scheme:

  - Plant model: Env
  - Input and output interface model: A_{IO}
  - Program model for A: \text{Prog}(A)
  - Execution model: A_{EX}
  - Digital clock model: A_{DC}

- Diagram:

Approach 1: the program implementing \( A \)

- translate \( A \) into \( \text{Prog}(A) \) an \textit{untimed automaton}
  
  interface of \( \text{Prog}(A) \): inputs = \{now, trig, inputs\}  outputs = \{outputs\}

- program the implementation of \( A \) by interpreting \( \text{Prog}(A) \):

  loop each trig --------------------------
  
  read now;  read inputs;  
  compute; update; write outputs;

endloop --------------------------
Approach 1: digital clock models

Digital clock model: $A_{DC}$

- provides now
- models that the clock of the CPU is digital (ie digitally updated)
- and may have some uncertainties

Examples

\[
x = \Delta \\
x := 0 \\
\text{tick!} \\
now := 0 \\
x \leq \Delta \\
now := \text{now} + \Delta
\]

\[
x \in [\Delta - \epsilon, \Delta + \epsilon] \\
x := 0 \\
\text{tick!} \\
now := 0 \\
x \leq \Delta \\
now := \text{now} + \Delta
\]
Approach 1: checking the implementation

A model around Prog(A) to check properties of the implementation

- A model of the execution platform: (timed automata)

  - digital clock: $A_{DC}$; $\rightarrow$ provides now
  - execution: $A_{EX}$; $\rightarrow$ provides trig!
  - communications: $A_{IO}$; $\rightarrow$ provides inputs/outputs

  $\rightarrow$ model of the platform: $P = A_{EX} || A_{DC} || A_{IO}$
plant model: $\text{Env}$

$\mathbf{a}_1 \?, \ldots, \mathbf{a}_n \?$

input and output interface model: $\mathbf{A}_{\text{IO}}$

output interface

input interface

program model for $\mathbf{A}$: $\text{Prog}(\mathbf{A})$

trig!

execution model: $\mathbf{A}_{\text{EX}}$

digital clock model: $\mathbf{A}_{\text{DC}}$

$\mathbf{b}_1 \!, \ldots, \mathbf{b}_m \!$
**Approach 1: checking the implementation**

A model around $\text{Prog}(A)$ to check properties of the implementation

- A model of the “real” execution of $A$:
  - execution platform: $P = A_{\text{EX}} || A_{\text{DC}} || A_{\text{IO}}$
  - reasonable assumptions on the environment: $\text{Env}$

$\Rightarrow$ model of the execution of the program that implements $A$
on the execution platform modeled by $P$
when executing uppon the environment $\text{Env}$

$M = \text{Env} || \text{Prog}(A) || P$
Approach 1: checking the implementation

Formal analysis of $M$

- verification (model-checking)
- controller synthesis
- preservation and "faster is better" properties are FALSE with no assumptions
  try to prove them under some restrictive hypothesis?
Approach 2: adapt the semantics

Context:
fix the assumptions under which executing the timed automaton, so as to ensure properties

→ fix a given platform
  ● digital clock of the CPU: periodically updated (period $\Delta_P$)
  ● execution: one cycle of computation takes at most $\Delta_L$
  ● communications: one shared buffer of size 1 per input/output

loop -----------------------------
  read now;       read inputs;
  compute; update; write outputs;
endloop --------------------------
Approach 2: results – Raskin et al. (ULB)

Definitions of new semantics:

• $[\{A\}]_{\Delta_L, \Delta_P}$: sem. of the program of A executing on the platform

• $[\{A\}_{\Delta}]$: new sem. for A, approximation by $\Delta$ of the ideal sem.
  — enlargement: $x \in [a, b] \rightarrow x \in [a - \Delta, b + \Delta]$

Theorems:

• if $\Delta > 4\Delta_P + 3\Delta_L$, then $[\{A\}]_{\Delta_L, \Delta_P}$ refines $[\{A\}_{\Delta}]$

• if $\Delta' < \Delta$, then $[\{A\}_{\Delta'}]$ refines $[\{A\}_{\Delta}]$

Robustness: A is robust wrt a property $\varphi$
iff $\exists \Delta$ st the semantics $[\{A\}_{\Delta}]$ satisfies $\varphi$
Approach 2: robust verification

- Verifies: $\exists \Delta$ st the semantics of $A_\Delta$ satifies $\varphi$

- Algo (idea): fix-point computation
  - $\text{Reach}(A_\Delta)$: the set of reachable states
  - computes: $\text{Reach}^*(A) = \cap_{\Delta > 0} \text{Reach}(A_\Delta)$

- Properties:
  - safety (ULB)
  - LTL (LSV)
  - bounded time properties (LSV)
Conclusion: Modeling vs Semantics

Modeling:
- uses classical timed automata, their semantics and algorithms
- allows changing the program type/execution platform by modularly changing the model
- offers possibilities for verification and synthesis
  BUT results are difficult to obtain

Semantics:
- introduces new semantics
- fixes the execution platform
- offers possibilities for robust verification
  + “Faster is better” property is true
Conclusion – Perspectives

Modeling:

● results: implementation framework using standard semantics + modeling

● to be continued: platform refinement and preservation

Semantics:

● results: implementability result on a given platform, for some properties

● to be continued: MTL properties
Related Work

- The tool TIMES [Uppsala]:
  - Timed automata that spawn tasks (multi-threaded programs)
  - Focus: schedulability analysis

- Timed Triggered Automata [Mokrushin, Krcal, Yi, Thiagarajan]:
  - Essentially discrete-time automata

- Digitization, robustness for timed automata [many]:
  - Focus: verification
  - Relation to code generation needs to be better understood