Implementability of Timed Automata

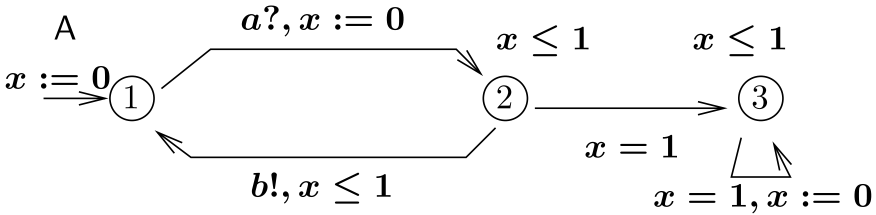
Karine Altisen (Verimag),

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

Context: Model-based Design

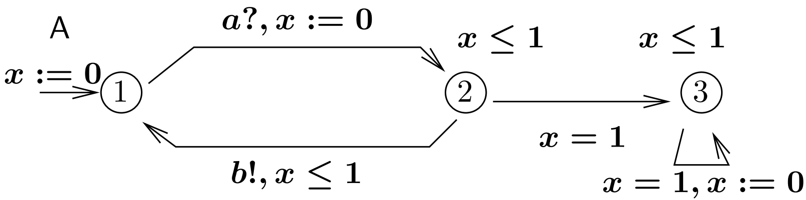
- Models are good for analysis:
  - simulation, testing, theorem prooving, verification...
- What about implementation
  - currently mostly an art/practice
- How to move from models to implementation?
  - as automatically as possible,
  - preserving as much as possible

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:

$$
\begin{array}{rcl}
 & (state = 1, x = 0) & \rightarrow \; (state = 1, x = 0.88) \\
\rightarrow ?a \rightarrow & (state = 2, x = 0) & \rightarrow \; (state = 2, x = 0.45) \\
\rightarrow b! \rightarrow & (state = 1, x = 0.45) & \rightarrow \; (state = 1, x = 54.3) \\
\rightarrow a? \rightarrow & (state = 2, x = 0) & \rightarrow \; (state = 2, x = 1) \\
\rightarrow & (state = 3, x = 1) & \ldots
\end{array}
$$

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)

$\longrightarrow$ a **model** with **ideal** semantics

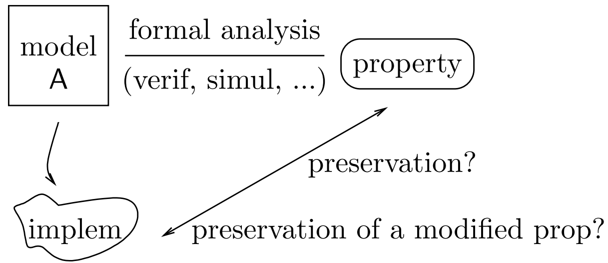
Towards a Realistic Platform

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:

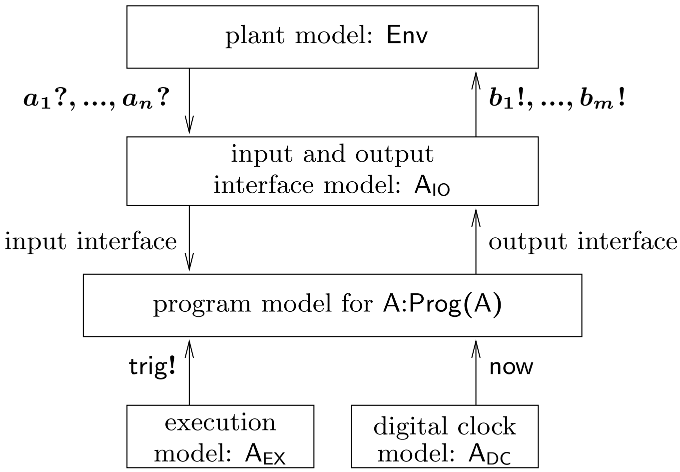
- Model it within a model of the execution platform
  KA+ST (Verimag)

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

Approach1: models the exec. platform

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

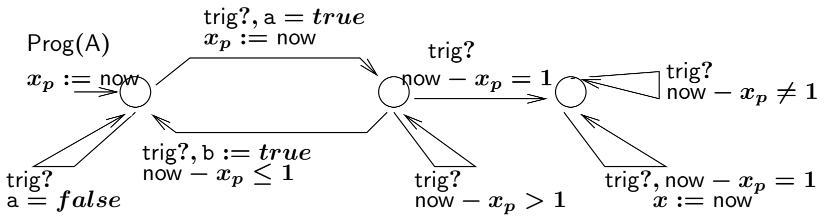
- global scheme



Approach1: the program implementing A

- translate A into Prog(A) an *untimed automaton*

  interface of Prog(A): inputs = {now, trig, *inputs*} outputs = {*outputs*}



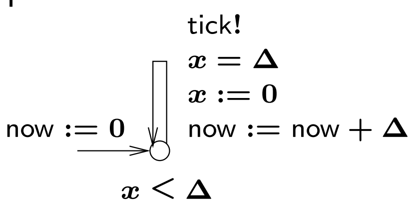
- program the implementation of A by interpreting Prog(A):

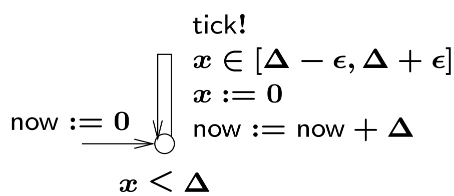
```
loop each trig -------------------
   read now;         read inputs;
   compute; update; write outputs;
endloop --------------------------
```

Approach1: 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





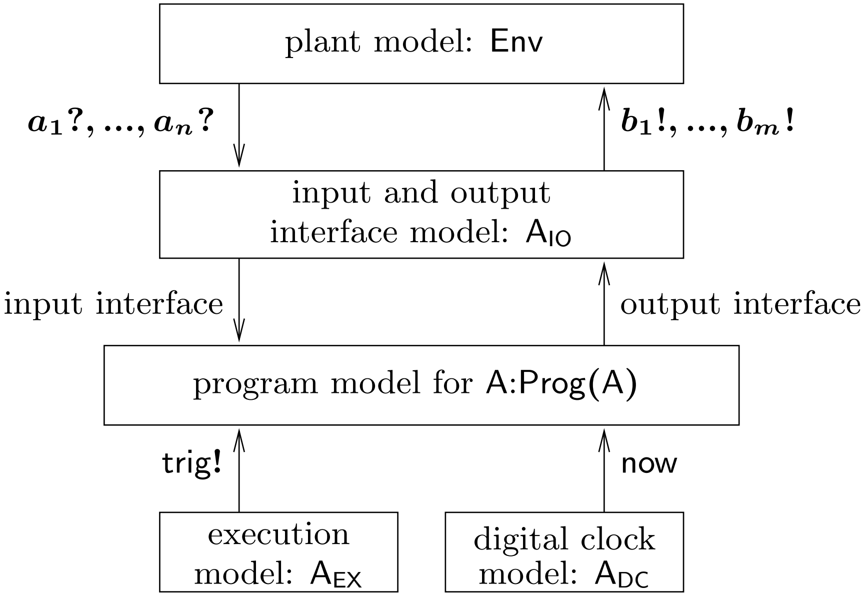
Approach1: checking the implementation

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

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

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

$\longrightarrow$ model of the platform: $\boldsymbol{P} = \mathsf{A}_{\mathsf{EX}} || \mathsf{A}_{\mathsf{DC}} || \mathsf{A}_{\mathsf{IO}}$


plant model: Env
$a_1?, ..., a_n?$
$b_1!, ..., b_m!$
input and output interface model: A_{IO}
input interface
output interface
program model for A:Prog(A)
trig!
now
execution model: A_{EX}
digital clock model: A_{DC}

Approach1: checking the implementation

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

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

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

$\boldsymbol{M} = \mathsf{Env}||\mathsf{Prog(A)}||\boldsymbol{P}$

Approach1: 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?

Approach2: 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 $\boldsymbol{\Delta_P}$)
- execution: one cycle of computation takes at most $\boldsymbol{\Delta_L}$
- communications: one shared buffer of size 1 per input/output

```
loop -----------------------------
   read now;        read inputs;
   compute; update; write outputs;
endloop --------------------------
```

Approach2: results – Raskin et al. (ULB)

Definitions of new semantics:

- $[\text{A}]_{\Delta_L,\Delta_P}$: sem. of the program of A executing on the platform
- $[\text{A}_\Delta]$: new sem. for A, approximation by Δ of the ideal sem.
  — enlargement: $x \in [a, b] \longrightarrow x \in [a - \Delta, b + \Delta]$

Theorems:

- if $\Delta > 4\Delta_P + 3\Delta_L$, then $[\text{A}]_{\Delta_L,\Delta_P}$ refines $[\text{A}_\Delta]$
- if $\Delta' < \Delta$, then $[\text{A}'_\Delta]$ refines $[\text{A}_\Delta]$

Robustness: A is *robust* wrt a property φ
iff $\exists\Delta$ st the semantics $[\text{A}_\Delta]$ satisfies φ

Approach2: robust verification

- Verifies: $\exists \boldsymbol{\Delta}$ st the semantics of $\mathsf{A}_{\boldsymbol{\Delta}}$ satifies φ

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

- 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 verfication 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