Performance and other non-functional aspects of systems: an approach with PA and TA

Speakers: M.R. Di Berardini, L. Tesei
Dipartimento di Matematica e Informatica
Università di Camerino

Kick-off Meeting, Bertinoro, 23-24 ottobre 2008
Outline

PaCo Contributions

Timed Automata, Fairness, Composition

Timed Automata with Invariants

Verification and Performance Related Tools
PaCo Contributions

- Synergy between PAs and TAs in the context of Performance and other non-Functional aspects of systems
- Transformation functions:
  - Import PAFAS efficiency preorder into a Timed Automata context - and Back
  - Compare fairness (and liveness) notions and tools in PAs/PAFAS and Timed Automata
  - Interactions between Queuing Networks and Timed Automata
  - ...
- Introduce probability/stochasticity in our setting(s)
- ...

Synergy between PAs and TAs in the context of Performance and other non-Functional aspects of systems

Transformation functions:
- Import PAFAS efficiency preorder into a Timed Automata context - and Back
- Compare fairness (and liveness) notions and tools in PAs/PAFAS and Timed Automata
- Interactions between Queuing Networks and Timed Automata
- ...

Introduce probability/stochasticity in our setting(s)

...
In this presentation

- We give an overview of the “tools” (mainly in the Timed Automata setting) we think are useful to reach our objectives.
Timed Automata

- Defined in early 90s by R. Alur, D. Dill, T. Henzinger, et al.
- Defined on a good theoretical basis: classical automata and \( \omega \)-automata
- Relatively simple extension of classical automata
- Several classical results and techniques can be established for the timed version (with surprising exceptions)
- Success in the community of researchers in modelling/verification
Example of a Timed Automaton

- Transitions are guarded by constraints on clocks
- Transitions can reset sets of clocks
- Transitions are instantaneous
The behaviour of a Timed Automaton is given by means of an LTS:

\[ T1 \quad \frac{\delta \in \mathbb{R}^>0}{(q, \nu) \xrightarrow{\delta} (q, \nu + \delta)} \quad T2 \quad \frac{(q, \psi, \gamma, \sigma, q') \in \mathcal{E}, \nu \models \psi}{(q, \nu) \xrightarrow{\sigma} (q', \nu \setminus \gamma)} \]

- \( \nu(x) \in \mathbb{R}^{\geq0} \) for every clock \( x \) is the clock valuation
- \( q \) is the location in the automaton \( (0, 1, 2, \ldots) \)
- \( \mathcal{E} \) are automaton transitions
Timed traces

- A run of the LTS defining the semantics is a possible behaviour of the system
  
  $$(0, x = 0) \xrightarrow{7.6}(0, x = 7.6) \xrightarrow{1.4}(0, x = 9.0)$$
  
  $$\xrightarrow{a}(1, x = 0) \xrightarrow{0.5}(1, x = 0.5) \xrightarrow{b}(0, x = 0.5)$$
  
  $$\xrightarrow{100.55}(0, x = 101.05) \xrightarrow{c}(2, x = 0) \xrightarrow{2}(2, x = 2)$$
  
  $$\xrightarrow{c}(2, x = 0) \cdots$$

- Corresponding timed trace:
  
  $$(a, 9)(b, 9.5)(c, 110.05)(c, 112.05) \cdots$$
The LTS defining the semantics has infinite states and is also infinite branching.

Equivalences must be defined to reduce it to finite states and perform verification.
Clock Constraints

\[ \psi ::= \text{true} \mid \text{false} \mid x \# c \mid x - y \# c \mid \psi \land \psi \mid \psi \lor \psi \mid \neg \psi \]

- \( x, y \) clock variables, \( c \in \mathbb{N} \), and \( \# \) is a binary operator in \( \{<, >, \leq, \geq, =\} \)
- Usually an equivalent minimal grammar is used
- OR can be expressed by non-determinism and duplication of states

\[ \psi ::= \text{true} \mid \text{false} \mid x \# c \mid x - y \# c \mid \psi \land \psi \]
Fairness

- Should express the real-time behaviour "after a a b occurs within 1 time unit"
- But the LTS could reach location 1 and stay there letting time to pass forever
- A notion of fairness is needed to exclude these traces
Automata Theoretic Fairness

- Only infinite traces with infinite non-\(\delta\) labels are taken
- An acceptance condition (e.g. Büchi) specifies which infinite traces are the intended behaviours
Parallel composition

\[ \mathcal{L}(T_1) = \{(b \ c^\omega, \bar{t}) \mid \exists \gamma \in \mathbb{R}^>: 1 < \gamma < 2 \land \forall i \in \mathbb{N}. \ t_i < \gamma\} \]
Parallel composition

\[
\mathcal{L}(T_2) = \{(b d^\omega, \overline{t}) \mid \forall i \in \mathbb{N}. t_i \leq t_{i+1}\}
\]
Parallel composition (simple product)

\[ L_{err} = \{(b \{c, d\}^* d^\omega, t) \mid t \text{ diverges}\} \]

Do not respect fairness of components
Parallel composition (fair composition)

\[ \mathcal{L}(T_1 \parallel T_2) = \{ (b \{ c, d \}^\omega, \bar{t}) \mid t_0 = 1, \forall i \in \mathbb{N}^>0. \ t_i \leq t_{i+1} < 2 \} \]
Fairness of components

Theorem

Let $T_1$ and $T_2$ be two timed automata with Büchi acceptance condition.

Let $r$ be a run of the timed automaton $T_1 \parallel T_2$.

Then, the projection $r|_1$ is a run of $T_1$ and the projection $r|_2$ is a run of $T_2$. 
Zeno runs

- Choosing a dense time domain like $\mathbb{R}^\geq 0$ lead to possible convergent time sequences

  $$ (0, x = 0) \xrightarrow{\frac{1}{4}} (0, x = \frac{1}{4}) \xrightarrow{\frac{1}{9}} (0, x = \frac{13}{36}) \ldots \xrightarrow{\frac{1}{n^2}} $$

- This trace represents a convergent time behaviour: "infinite things take place in a finite amount of time"

- These traces cannot be considered as behaviours of a real-time systems and should be excluded from verification
Throwing away Zeno runs

Put the automaton specifying the system in parallel with:

\[ x < 1, \Sigma, \{x\} \]
Timed Safety Automata (TSA)

- Simplification to reach fairness without acceptance conditions, Sifakis et al. 1994
- States contain right-closed clock constraints called **invariants**
- Time can elapse in states if and only if the invariant is satisfied by the current clock valuation
- Fairness becomes: *timestops* (in a state time passing is not possible) are not allowed
Timed Safety Automata (TSA)

- The model is less expressive than the automata theoretical one, but its implementation is easier in both verification and simulation.
- In simulation the *next-state* function depends only on the current state.
Timed Safety Automata (TSA)

- "eventually c is executed" (unbounded inevitability) cannot be expressed, while it is possible with the acceptance condition
- Actions fairness is expressed by the invariant $x \geq 30$ in state 0
Timed Safety Automata (TSA)

- Bounded inevitability can be expressed
- Let’s use another clock $y$ to express that “eventually $c$ is executed within 200 time units”
Detecting Zeno states

- The model checking algorithm for TSA can detect all Zeno states.
- Zeno states are all those states \((q, \nu)\) from which ONLY Zeno runs start.
- It is sufficient to verify that "from every state there exists a path that eventually leads to a state in which 1 time unit has elapsed."
- This can be expressed as a TCTL formula and can be model checked.
- If the check fails we can find Zeno states from the diagnostic trace.
Detecting Zeno states

- \((q, \nu)\) such that \(q = 0, 1\) and \(4 < \nu(x) \leq 5\) are Zeno states
- The model checker shows the Zeno states that usually can be easily eliminated
- e.g. change invariants to \(x \leq 4\)
Parallel Composition of TSAs

A binary counter of $bs$

\[ x = 1, \text{ display0, } \{x\} \]
Parallel Composition of TSAs

An interference to free occurrence of $bs$

\[ y \leq 5 \quad 0 \quad y > 2, \quad b, \quad \{y\} \]
Parallel Composition of TSAs

Their composition
Parallel composition of TSAs

- Parallel composition of Timed Büchi automata must take into account also accepting states
- Construction is more complex than the simple product of transition tables
- In case of TSA the construction is simply the product of transition tables
Timed Automata with Deadlines

- Introduced by Bornot and Sifakis (1998)
- Every transition has a guard and a deadline (the deadline must imply the guard)
- Time can pass in a state as long as all deadlines do not hold
- Submodel of TSAs, but permits a compositional specification of timed systems
- Algebraic specification - a bridge with PAs
- A concept of urgency is defined which behaves well with composition
Verification

- **Approach 1**: automata theoretic verification
  - Based on automata theoretic results
  - One tool to perform verification

- **Approach 2**: model checking
  - Timed Temporal Logic formula $\varphi$ expressing property
  - Timed Safety Automaton $A$ with boolean variables in states expressing the system
  - $A \models \varphi$?
  - If $\models$ is verified the system fits the property, else we get diagnostic information
Positive results for Verification:

- Reachability problem is DECIDABLE
- Language Emptiness is DECIDABLE (PSPACE)
- TAs are closed w.r.t. intersection and union
- Language inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ is DECIDABLE only if $B$ is deterministic
Automata Theoretic Verification

Negative results for Verification:

- Non-deterministic TAs with Büchi acceptance condition are more expressive than deterministic ones (determinism takes into account actions and their time of enabling)
- Büchi non-deterministic TAs are not closed under complementation
- Language inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ is UNDECIDABLE if $B$ is non-deterministic
Minumum and Maximum Delay

- Courcoubetis and Yannakakis (CAV ’91)
- Compute the minimum and maximum delay to reach a target state starting from a source state
- Natural definition of **quantitative** best and worst case time complexity
- Probabilistic Timed Automata for the average case?
Priced (or Weighted) Timed Automata

- Timed Automata with costs on the transitions and on the states
- Optimal reachability problem: given sets of source states $S$ and target states $T$ determine, for each $s \in S$, the infimum cost over all the runs of the automaton from $s$ to a state in $T$
- Solved in exponential time
- Also timed games on this model (or variants) have been studied
Train-Gate example: Train

- true, idle_T, {}
- x >= 3, approach, {x}
- x < 8, exit, {x}
- x > 4, in, {}
- true, out, {}
Train-Gate example: Gate

true, idle_G, {}

true, lower, \{y\}

1 < y < 2,
up, \{}

y < 1,
down, \{}

true, raise, \{y\}

ture, idle_G, {}
Train-Gate example: Controller

true, idle_C, {}

0

true, approach, {z}

1

z < 1, raise, {}

3

z = 1, lower, {}

2

true, exit, {z}
Train-Gate example: Verification

- **Safety** property: "whenever the train is inside the gate, the gate must be closed"
- Can be expressed as a reachability problem on the System: Train|Gate|Controller
- Check that all states in which
  - the train is inside the gate (state 2)
  - the gate is *not* closed (states \{0, 1, 3\})
  cannot be reached
Train-Gate example: Verification

- **Liveness** property: The gate never remains closed for more than 11 minutes
- Technique:
  - Model the negation of property by a TA $N$

```
true, $\Sigma$, {}
true, $\Sigma \setminus \{\text{up}\}$, {}
true, $\Sigma$, {}
```

```
0 \rightarrow \text{true, down, \{x\}} \rightarrow 1 \rightarrow \text{x > 11, $\Sigma$, \{\}} \rightarrow 2
```
Train-Gate example: Verification

- **Technique:**
  - Construct the intersection automaton $I$ of the system $A$ and $N$
  - Check that the language of $I$ is empty
  - If it is empty then $A$ satisfies the original property (the negation of $N$)
  - Else the runs of $I$ give diagnostic information
Automata theoretic verification

- Language inclusion can be used only if the property to be verified can be expressed by a deterministic timed automaton:
  - Take the system $A$
  - Model the property by automaton $P$
  - $A$ satisfies the property if and only if $L(A) \subseteq L(P)$
- Not always possible
Automata theoretic verification

- Tool available: Open-KRONOS (Profoundus)
- Solve reachability problem on Timed Büchi automata
- Check language emptiness on Timed Büchi automata
Verification and Performance Related Tools

Model checking

- Classical paradigm of verification
- Extended to real-time case
- Logic: Timed Computational Tree Logic (TCTL)
- Model: Timed Safety Automata with boolean variables on the states
- Tool KRONOS (VERIMAG, France): almost all TCTL
- Tool UPPAAL (Univ. Uppsala, Sweden - Univ. Aalborg, Denmark): little fragment of TCTL
The Model

- Essentially Timed Safety Automaton $A$
- To facilitate formulas writing locations of $A$ must be labeled with boolean variables (State-based approach vs. Action-based approach)
- Let’s adapt the Train-Gate model used for automata theoretic verification
- We introduce new actions (initiation and termination of activities with duration)
- We introduce boolean variables to specify context-dependent information
TSA: Train

- **State 0**: IDLE_T
  - When `x >= 3`, transition to State 1 with condition `true`.
  - When `x < 8`, transition to State 3 with condition `exit, {x}`.

- **State 1**: ARRIVING
  - When `x <= 5`, transition to State 2 with condition `true`.
  - When `x > 4`, transition to State 3 with condition `i_crossing, {w}`.

- **State 2**: CROSSING
  - When `w <= 2`, transition to State 3 with condition `{w}`.
  - When `w > 1`, transition to State 0 with condition `c_crossing, {}`.

- **State 3**: CROSSED
  - When `x < 8`, transition to State 0 with condition `{x}`.
TSA: Gate

0
IDLE_G
true

1
y < 1

2
DOWNING
z < 2

3
CLOSED
true

4
y < 2

5
UPPING
z < 2

1 < z < 2,
c_up,
{}

1 < z < 2,
c_down,
{}

1 < y < 2,
i_up, {z}

true, raise, {y}
TSA: Controller

0
IDLE_C
true

true, approach, \{u\}

1
u \leq 1

2
true

3
u < 1

u < 1, raise, \{}

u = 1, lower, \{}

true, exit, \{u\}
Logic TCTL

Think in terms of Computational Trees
Consider a state \((q, \nu)\) of the LTS defining the semantics of the TSA. This state is the root of a computational tree with infinite depth and infinite branching. Every infinite path of this tree is a suffix of a run of the TSA that reach \((q, \nu)\). If \((q, \nu)\) is the initial state of the TSA, then the paths of the computational tree represents the set of all runs of the TSA.
TCTL: basic syntax

\[ \varphi ::= \psi \quad \text{Clock constraint} \]
\[ \mid b \quad \text{Boolean variable} \]
\[ \mid z.\varphi \quad \text{Freeze clock} \]
\[ \mid \neg \varphi \quad \text{Negation} \]
\[ \mid \varphi_1 \lor \varphi_2 \quad \text{Disjunction} \]
\[ \mid \varphi_1 \exists U \varphi_2 \quad \text{Exist Path Until} \]
\[ \mid \varphi_1 \forall U \varphi_2 \quad \text{Forall Paths Until} \]
TCTL: basic semantics

\[(q, \nu, \zeta) \models \psi \iff \nu \cup \zeta \models \psi\]

\[(q, \nu, \zeta) \models b \iff b \in \mathcal{P}(q)\]

\[(q, \nu, \zeta) \models z \cdot \varphi \iff (q, \nu, \zeta \setminus \{z\}) \models \varphi\]

\[(q, \nu, \zeta) \models \neg \varphi \iff (q, \nu, \zeta) \not\models \varphi\]

\[(q, \nu, \zeta) \models \varphi_1 \lor \varphi_2 \iff (q, \nu, \zeta) \models \varphi_1 \text{ or } (q, \nu, \zeta) \models \varphi_2\]

\[(q, \nu, \zeta) \models \exists U \varphi_2 \iff \exists \pi(q, \nu) \in \prod_{\infty}(q, \nu): \exists p = (i, \delta) \in \text{Pos}(\pi(q, \nu)):\]

\[s_i = (q_i, \nu_i) \land (q_i, \nu_i + \delta, \zeta + \Delta(p)) \models \varphi_2\]

\[\land \forall p' = (j, \delta') \in \text{Pos}(\pi(q, \nu)). (p' \prec p \land s_j = (q_j, \nu_j))\]

\[\Rightarrow (q_j, \nu_j + \delta', \zeta + \Delta(p')) \models \varphi_1 \lor \varphi_2\]

...
TCTL: useful syntactic sugar

$\exists^\diamond \varphi$ is the formula to express *reachability*. It is satisfied by a state $(q, \nu)$ iff there exists a $(q, \nu)$-path in which eventually a state satisfying $\varphi$ is reached. The translation is $\text{true} \exists^\bigcup \varphi$

$\forall \Box \varphi$ expresses *invariance*. It is satisfied by a state $(q, \nu)$ iff $\varphi$ is satisfied in all states reachable along all $(q, \nu)$-paths. The translation, as usual, is $\neg \exists^\diamond \neg \varphi$
TCTL: useful syntactic sugar

\( \forall \diamond \varphi \) expresses *inevitability*. It is satisfied by a state \((q, \nu)\) iff in all \((q, \nu)\)-paths a state in which \(\varphi\) is satisfied is reachable. The translation is \(true \forall U \varphi\)

\( \exists \Box \varphi \) expresses *possible invariance*. A state \((q, \nu)\) satisfies it iff there exists a \((q, \nu)\)-path along which the formula \(\varphi\) is satisfied in all reachable states. The translation is \(\neg \forall \diamond \neg \varphi\)
TCTL: useful syntactic sugar

- $\exists \diamond \leq c \varphi$ is **bounded reachability**. A state $(q, \nu)$ satisfies it iff there exists a $(q, \nu)$-path along which a state satisfying $\varphi$ is reachable within $c$ time units. The translation uses the freeze quantifier: $z.\exists \diamond (\varphi \land z \leq c)$

- $\forall \diamond \leq c \varphi$ is **bounded inevitability**. A state $(q, \nu)$ satisfies it iff in all $(q, \nu)$-paths a state satisfying $\varphi$ is reachable within $c$ time units. The translation is $z.\forall \diamond (\varphi \land z \leq c)$
KRONOS: Train-Gate verification

- KRONOS model checks almost all TCTL
- KRONOS can construct the Parallel Composition on the fly while verifying a formula
- Train-Gate safety property:

\[(\text{IDLE}_T \land \text{IDLE}_G \land \text{IDLE}_C) \rightarrow \forall \Box (\text{CROSSING} \rightarrow \text{CLOSED})\]
KRONOS: Train-Gate verification

Train-Gate liveness property:

\[(\text{IDLE}_T \land \text{IDLE}_G \land \text{IDLE}_C) \rightarrow \forall \Box (\text{CLOSED} \rightarrow \forall \Diamond_{\leq 11} \text{IDLE}_G)\]
UPPAAL Verification

- UPPAAL has a graphical interface for drawing automata
- UPPAAL has a simulator to run some traces of automata
- UPPAAL synchronisation is based on the concept of Network of Automata
- UPPAAL model checker can check only properties rephrasable in term of reachability
- UPPAAL verification engine is optimised and efficient
UPPAAL supported logic

\( \varphi ::= \exists \lozenge \text{Expr} \)
\( | \forall \square \text{Expr} \)
\( | \forall \lozenge \text{Expr} \)
\( | \exists \square \text{Expr} \)
\( | \forall \square (\text{Expr} \rightarrow \exists \lozenge \text{Expr}) \)

Expr can be a boolean expression involving variables or a dot expression of the form \( P.s \) that is satisfied only if the component \( P \) is in state \( s \)
Fischer’s Mutual excl. algorithm

- Uses time to guarantee mutual exclusion
- Shared integer variable $x \in \{0, 1, 2\}$
- Process $P_i$ checks if $x == 0$, then set $x = i$ within $b$ time units
- Then, it waits $b$ time units and enters critical section iff $x$ still equals $i$
- It stays in the critical section for a limited time ($ucs$)
Fischer’s Mutex: Process 1

```
Verification and Performance Related Tools

Fischer's Mutex: Process 1

idle
  x := 0
  ac?
  y := 0
  ac?
  y <= b
trying
  y <= b
  y := 0
  ac?
  x == 0
  ac?
  x := 1
waiting
  y <= b
  y := 0, x := 1
  ac?
  y == b, x == 1
  ac?
  y == b, x != 1
critical
  y := 0
  ac?
```
Fischer’s Mutex: Process 2

- Verification and Performance Related Tools

### Fischer’s Mutex: Process 2

**States:**
- **Idle**
- **Critical**
- **Waiting**

**Transitions:**
- **Idle** to **Trying:** $y \leq b$
- **Trying** to **Idle:** $x = 0$
- **Idle** to **Critical:** $bc$?
- **Trying** to **Waiting:** $y = 0$
- **Waiting** to **Trying:** $y = 0, x = 2$
- **Waiting** to **Critical:** $y = b, x \neq 2$
- **Trying** to **Critical:** $y = b, x = 2$
- **Critical** to **Waiting:** $y = 0$
- **Waiting** to **Critical:** $bc$?
Serialisation

- Access to variable $x$ must be serialised in order to consider it atomic
Fischer’s Mutex: Serialiser

\[
\begin{align*}
&y \geq acc \\
&ac! \\
&y := 0 \\

&bc! \\
&y := 0 \\
&cc! \\
\end{align*}
\]
Fischer’s Mutex: Attacker

- An attacker can access the variable \( x \) and set it at any value.
- The attacker has a limited power.
- Every attack can take place iff at least \( n \) time units have elapsed from the previous attack.
- How much power the attacker need to break the protocol safety?
- Safety: ”\( P_1 \) and \( P_2 \) never access the critical section simultaneously”
Fischer’s Mutex: Attacker

- $y \geq n \quad cc? \quad y := 0, x := 0$
- $y \geq n \quad cc? \quad y := 0, x := 1$
- $y \geq n \quad cc? \quad y := 0, x := 2$
Fischer’s Mutex: Verification

- If \( n > b \) then the protocol maintains safety
- The following symmetric properties can be checked by UPPAAL:

\[
\forall \Box (P_1.\text{critical} \rightarrow \neg P_2.\text{critical})
\]

\[
\forall \Box (P_2.\text{critical} \rightarrow \neg P_1.\text{critical})
\]

What about protocol liveness?
Verification and Performance Related Tools

University of Camerino

Thanks!