Specification and Verification of real-time Systems with Timed Automata

Models, Logics, Verification

Luca Tesei
Reactive Systems

- Interact with an environment
- Perform internal or visible operations abstracted by actions \(a, b, \ldots\)
- Receive inputs from environment \((a, b, \ldots)\)
- Receive signals of events from environment \((a, b, \ldots)\)
- Typically run forever \((\text{Behaviours} \subseteq \{a, b, \ldots\}^\omega)\)
- Can stop and idle \((\text{Behaviours} \subseteq \{a, b, \ldots\}^*)\)
Reactive Systems examples

- Aircraft Control
- Washing Machine
- Cash Dispenser
- Queue, Stack, Buffer, ...
- Operating System
- Protein, Cell, ...
- Java objects
- ...

University of Camerino - Seminari @ cs.unicam – p.3/64
Untimed Models

- Automata recognising finite or infinite strings
- Process Algebras
- Petri Nets
- I/O Automata
- ...

ALL can be "compiled" into Finite or Infinite (Labeled) Transition Systems
Can express ONLY QUALITATIVE time information

"a occurs AFTER b", "every c is followed either by an a or a sequence of d", …
Real-time systems

- Their behaviour depends on hard time constraints

  "after an a a b occur within 5 time units", "c cannot be executed before 7 time units have elapsed from the previous occurrence of d"

- The constraints can also be punctual: "c occurs exactly after 3 time units from a"

- Classical untimed models CANNOT express these constraints
Timing the untimed

- New models have to be defined for real-time systems
- Hopefully by extending untimed models with time
- Hopefully obtaining similar verification tools: model checkers, bisimulation checkers, ...

- Some proposals:
  - Timed Petri Nets
  - Timed Process Algebras
  - Timed Automata
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
Clocks

- Introduction of time: clock variables
- Variables $x, y, z, \ldots$ measuring time
- Take values in $\mathbb{R}^{\geq 0}$
- All increase at the same rate: $\dot{x} = \dot{y} = \cdots = 1$
- Can be reset to 0 by transitions
- Every Timed Automaton has its own finite set of clock variables
- Clocks are used to express time constraints on transitions
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

\[ T1 \quad \delta \in \mathbb{R}^{>0} \quad (q, \nu) \xrightarrow{\delta} (q, \nu + \delta) \]

\[ T2 \quad (q, \psi, \gamma, \sigma, q') \in \mathcal{E}, \nu \models \psi \quad (q, \nu) \xrightarrow{\sigma} (q', \nu \setminus \gamma) \]

- \( \nu(x) \in \mathbb{R}^{\geq 0} \) 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

\[(0, x = 0) \xrightarrow{7.6}(0, x = 7.6) \xrightarrow{1.4}(0, x = 9.0)\]
\[a \xrightarrow{} (1, x = 0) \xrightarrow{0.5}(1, x = 0.5) \xrightarrow{b}(0, x = 0.5)\]
\[\xrightarrow{100.55}(0, x = 101.05) \xrightarrow{c}(2, x = 0) \xrightarrow{2}(2, x = 2)\]
\[c \xrightarrow{} (2, x = 0) \ldots\]

Corresponding timed trace:
\[a, 9)(b, 9.5)(c, 110.05)(c, 112.05) \ldots\]
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

- What kind of clock constraints can be put as guards on edges?
- This is a CRUCIAL point
- Choosing constraints that are too expressive can lead to UNDECIDABILITY of verification
- The maximum expressive power allowed to achieve decidability of model checking or language emptiness is precisely defined
- It is sufficiently expressive to specify typical behaviours of real-time systems
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

\[
\psi ::= \text{true} \mid \text{false} \mid x \# c \mid x - y \# c \mid \psi \land \psi
\]
Clocks constraints must contain natural number constants

However we can write arbitrarily precise constraints!

It is always possible to use rational numbers in clock constraints and then rewrite the automaton using natural constants.
Simply scale the time unit of $\frac{1}{m}$ where $m$ is the m.c.d. of all denominators of rational constants.

This is not free: the size of the state space for verification is exponential in the larger natural constant in clock constraints.
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.

Diagram:

- From state 1, with label $x \leq 1, b, \{\}$, transition to state 0.
- From state 0, with label true, c, \{x\}, transition to state 2.
- From state 2, with label $x = 2, c, \{x\}$, transition to state 2.
- From state 2, with label true, a, \{x\}, transition to state 0.
- From state 0, with label true, c, \{x\}, transition to state 2.
Timed Safety Automata (TSA)

- Practical simplification to reach fairness
- All tools are based on this model
- 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
- The model is less expressive than the automata theoretical one, but its implementation is feasible
"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.
Bounded inevitability can be expressed

Let’s use another clock $y$ to express that "eventually $c$ is executed within 200 time units"
Strange things called Zeno runs

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

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

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
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

As for the untimed case it is convenient to model small pieces of systems and then compose them to obtain the whole system.

As for the untimed case this potentially changes the stand-alone behaviour of components.

Deadlocks and Zeno states can be introduced!

The source of this trouble is the mechanism of synchronisation of timed automata.
Parallel Composition

If two or more timed automata are in parallel they:

- must have disjoint sets of clocks
- can share some action symbols, which constitute the **synchronisation actions**
- can perform independently non-synchronisation actions (interleaving)
- can perform a synchronisation action $a$ iff ALL components having $a$ in their alphabet can do $a$ (synchronisation)
A binary counter of $b$s

- State $0$: $x \leq 1$, $x=1$, display0, \{x\}
- State $1$: $x \leq 1$, $x=1$, display1, \{x\}

true, $b$, \{\}

Parallel Composition
Parallel Composition

An interference to free occurrence of $b_s$
Their composition

Parallel Composition

University of Camerino - Seminari @ cs.unicam – p.30/64
Verification

- Approach 1: automata theoretic verification
  - Based on automata theoretic results
  - Not often used: no tools

- 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 if $B$ is deterministic
Automata Theoretic Verification

Negative results for Verification:

- Non-deterministic TAs with Büchi acceptance condition are more expressive than deterministic ones
- 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
Train-Gate example: Train

true, idle_T, {}

\( x \geq 3 \), approach,
\{x\}

\( x < 8 \),
exit,
\{x\}

true, out, {}

\( x > 4 \),
in,
\{\}

\( x \geq 3 \), approach,
\{x\}
Train-Gate example: Gate
Train-Gate example: Controller

- **State 0**: True, idle_C, {}
- **State 1**: True, approach, {z}
- **State 2**: True, exit, {z}
- **State 3**: True, idle_C, {}

- **Transition 1**: If z < 1, raise, {}
- **Transition 2**: If z = 1, lower, {}

Diagram showing the states and transitions with conditions for each.
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 $N$

```
0 -> true, down, \{x\}
true, \Sigma, \{}       true, \Sigma \setminus \{\text{up}\}, \{}       true, \Sigma, \{}
1 -> x > 11, \Sigma, \{}
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 $\mathcal{L}(A) \subseteq \mathcal{L}(P)$

- Not always possible
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

0
IDLE_T
true

1
ARRIVING
x <= 5

3
CROSSED
x < 8

2
CROSSING
w <= 2

x >= 3, approach, {x}

x < 8, exit, {x}

x > 4, i_crossing, {w}

w > 1, c_crossing, {}
TSA: Gate

0
IDLE_G
true

1
y < 1

2
DOWNING
z < 2

3
CLOSED
true

4
y < 2

5
UPPING
z < 2

\[ \begin{align*}
\text{true, lower, } \{x\} \\
1 < z < 2, \\
c_{\text{up}}, \\
\{} \\
\end{align*} \]

\[ \begin{align*}
y < 1, \ i_{\text{down}}, \ \{z\} \\
\end{align*} \]

\[ \begin{align*}
1 < z < 2, \\
c_{\text{down}}, \\
\{} \\
\end{align*} \]

\[ \begin{align*}
1 < y < 2, \\
i_{\text{up}}, \ \{z\} \\
\end{align*} \]

\[ \begin{align*}
\text{true, raise, } \{y\} \\
\end{align*} \]
TSA: Controller

0
IDLE_C
true

1
u <= 1

3
u < 1
raise,
{}

2
true

true, approach, \{u\}

u < 1,
raise,
{}
u = 1,
lower,
{}

true, exit, \{u\}
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 represent the set of all runs of the TSA.
TCTL: basic syntax

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

\[(q, \nu, \zeta) \models \psi \quad \text{iff} \quad \nu \cup \zeta \models \psi\]

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

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

\[(q, \nu, \zeta) \models \neg \varphi \quad \text{iff} \quad (q, \nu, \zeta) \not\models \varphi\]

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

\[(q, \nu, \zeta) \models \varphi_1 \exists U \varphi_2 \quad \text{iff} \quad \exists \pi_{(q, \nu)} \in \Pi_A^{\infty}(q, \nu): \exists p = (i, \delta) \in \text{Pos}(\pi_{(q, \nu)}):
\]

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

\[\land \forall p' = (j, \delta') \in \text{Pos}(\pi_{(q, \nu)}). (p' < p \land s_j = (q_j, \nu_j)) \Rightarrow (q_j, \nu_j + \delta', \zeta + \Delta(p')) \models \varphi_1 \lor \varphi_2\]

\[\cdots\]
∃◊ϕ is the formula to express reachability. It is satisfied by a state (q, ν) iff there exists a (q, ν)-path in which eventually a state satisfying ϕ is reached. The translation is true∃Uϕ.

∀□ϕ expresses invariance. It is satisfied by a state (q, ν) iff ϕ is satisfied in all states reachable along all (q, ν)-paths. The translation, as usual, is ¬∃◊¬ϕ.
\( \forall \Diamond \varphi \) 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 model checks almost all TCTL

KRONOS can construct the Parallel Composition on the fly while verifying a formula

Train-Gate safety property:

\[(\text{IDLE}_T \land \text{IDLE}_G \land \text{IDLE}_C) \rightarrow \forall \square (\text{CROSSING} \rightarrow \text{CLOSED})\]
Train-Gate liveness property:

\[(\text{IDLE}_T \land \text{IDLE}_G \land \text{IDLE}_C) \rightarrow \forall \Box (\text{CLOSED} \rightarrow \forall \Diamond \leq_{11} \text{IDLE}_G)\]

together with

\[(\text{IDLE}_T \land \text{IDLE}_G \land \text{IDLE}_C) \rightarrow \forall \Box (\text{CROSSING} \rightarrow \neg \text{UPPING})\]
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 rephrased in term of reachability
- UPPAAL verification engine is optimised and efficient
\[ \phi ::= \exists \Diamond \text{Expr} \]

\[ \mid \forall \Box \text{Expr} \]

\[ \mid \forall \Diamond \text{Expr} \]

\[ \mid \exists \Box \text{Expr} \]

\[ \mid \exists \Box (\text{Expr} \rightarrow \exists \Diamond \text{Expr}) \]

\text{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

idle

trying

y \leq b

waiting

y \leq b

critical

y \leq ucs

x == 0

ac?

y := 0

ac?

y := 0

x := 1

ac?

y := 0, x := 1

x := 0

ac?

y == b, x != 1

ac?

y == b, x == 1

x := 0

ac?

y := 0

ac?

y := 0

y == b, x == 1

y == b, x != 1

University of Camerino - Seminari @ cs.unicam – p.58/64
Fischer’s Mutex: Process 2

idle

trying
y <= b

waiting
y <= b

x == 0

bc?

y := 0

bc?

y <= b

x == 0

bc?

y := 0, x := 2

bc?

y == b, x != 2

bc?

y == b, x == 2

bc?

y := 0

bc?

y <= ucs

critical

y := 0

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

\[ y \geq acc \quad ac! \quad y := 0 \]

\[ bc! \]

\[ y := 0 \]

\[ y \geq acc \quad cc! \]

\[ y := 0 \]
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

\[
\begin{align*}
\text{y >= n} & \quad \text{cc?} & \quad \text{y := 0, x := 0} \\
\text{y >= n} & \quad \text{cc?} & \quad \text{y := 0, x := 2} \\
\text{y := 0, x := 1} & & \\
\end{align*}
\]
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?