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

- Synergy between PAs and TAs in the context of Perfomance 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)
- ...

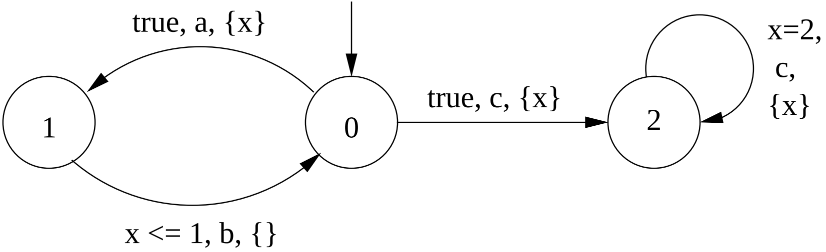
In this presentation

- 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 ω-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

Semantics

- The behaviour of a Timed Automaton is given by means of an LTS

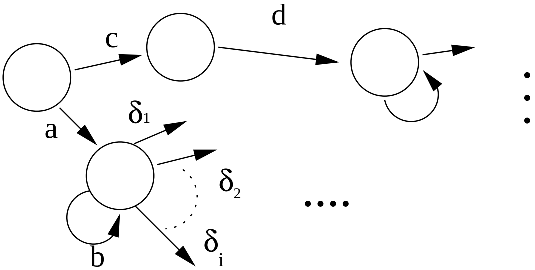
$$\text{T1} \quad \frac{\delta \in \mathbb{R}^{>0}}{(q,\nu) \xrightarrow{\delta} (q, \nu + \delta)} \qquad \text{T2} \quad \frac{(q,\psi,\gamma,\sigma,q') \in \mathcal{E}, \nu \models \psi}{(q,\nu) \xrightarrow{\sigma} (q', \nu \backslash \gamma)}$$

- $\nu(x) \in \mathbb{R}^{\geq 0}$ 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
- $(0, x = 0) \xrightarrow{7.6} (0, x = 7.6) \xrightarrow{1.4} (0, x = 9.0) \xrightarrow{a} (1, x = 0) \xrightarrow{0.5} (1, x = 0.5) \xrightarrow{b} (0, x = 0.5) \xrightarrow{100.55} (0, x = 101.05) \xrightarrow{c} (2, x = 0) \xrightarrow{2} (2, x = 2) \xrightarrow{c} (2, x = 0) \cdots$
- Corresponding timed trace: $(a, 9)(b, 9.5)(c, 110.05)(c, 112.05) \cdots$

Infinite Branching



- 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

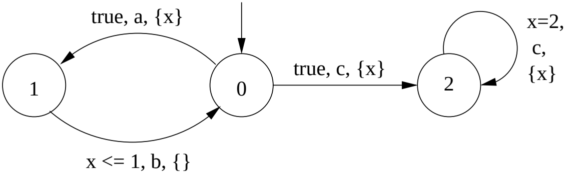
Clock Constraints

$$\psi \ ::= \ \mathit{true} \mid \mathit{false} \mid x \# c \mid x - y \# c \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi$$

- 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

$$\psi \ ::= \ \mathit{true} \mid \mathit{false} \mid x \# c \mid x - y \# c \mid \psi \wedge \psi$$

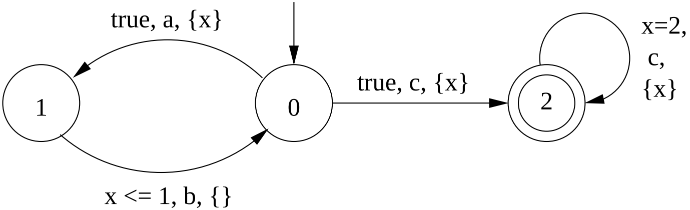
Fairness



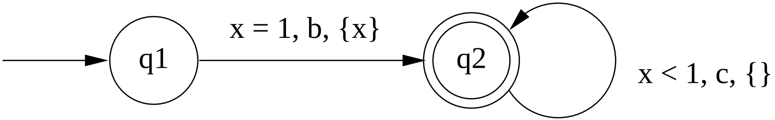
- Should express the real-time behaviour "after *a* a *b* occurs within 1 time unit"
- But the LTS could reach location 1 and stay there letting time to pass forever
- A notion of fairness is needed to exclude these traces

Automata Theoretic Fairness

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

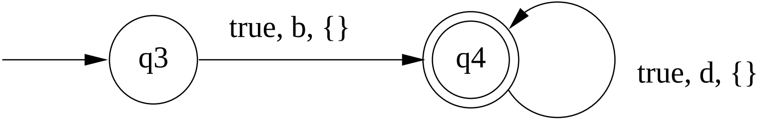


Parallel composition



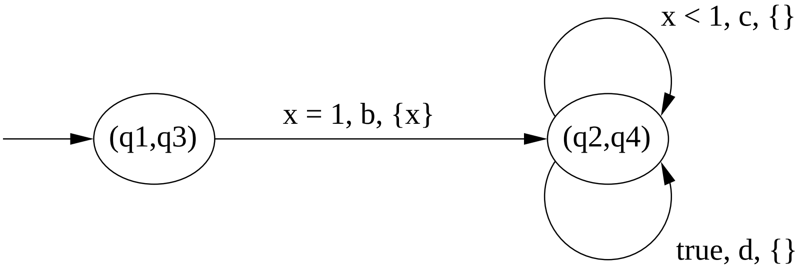
$$\mathcal{L}(T_1) = \{(b\, c^\omega, \overline{t}) \mid \exists \gamma \in \mathbb{R}^{>0} \colon\; 1 < \gamma < 2 \;\wedge\; \forall i \in \mathbb{N}.\; t_i < \gamma\}$$

Parallel composition



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

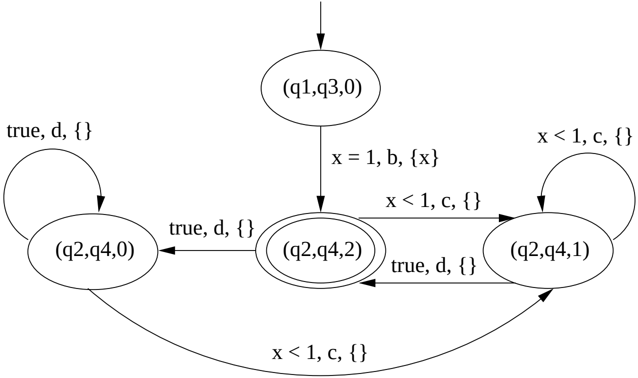
Parallel composition (simple product)



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

Do not respect fairness of components

Parallel composition (fair composition)



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

Fairness of components

Theorem

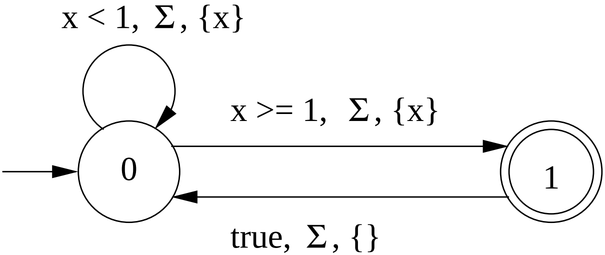
Let T_1 and T_2 be two timed automata with Büchi acceptance condition.
Let r be a run of the timed automaton $T_1 \parallel T_2$.
Then, the projection $r_{|1}$ is a run of T_1 and the projection $r_{|2}$ is a run of T_2.

Zeno runs

- Choosing a dense time domain like $\mathbb{R}^{\geq 0}$ lead to possible convergent time sequences
- $(0, x = 0) \xrightarrow{\frac{1}{4}} (0, x = \frac{1}{4}) \xrightarrow{\frac{1}{9}} (0, x = \frac{13}{36}) \cdots \xrightarrow{\frac{1}{n^2}}$
- 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

Throwing away Zeno runs

Put the automaton specifying the system in parallel with:



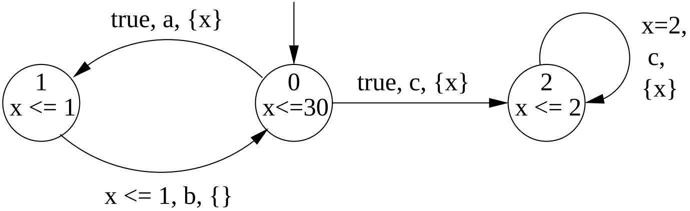
Timed Safety Automata (TSA)

- 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

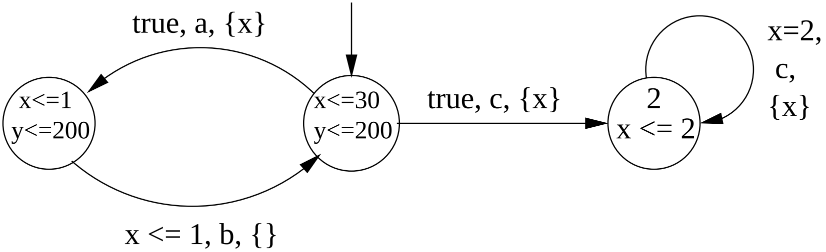
Timed Safety Automata (TSA)



- "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)

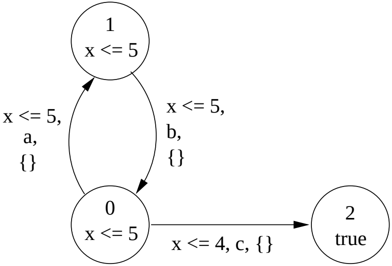
- Bounded inevitability can be expressed
- Let's use another clock y to express that "eventually c is executed within 200 time units"



Detecting Zeno states

- The model checking algorithm for TSA can detect all Zeno states
- Zeno states are all those states (q, ν) 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

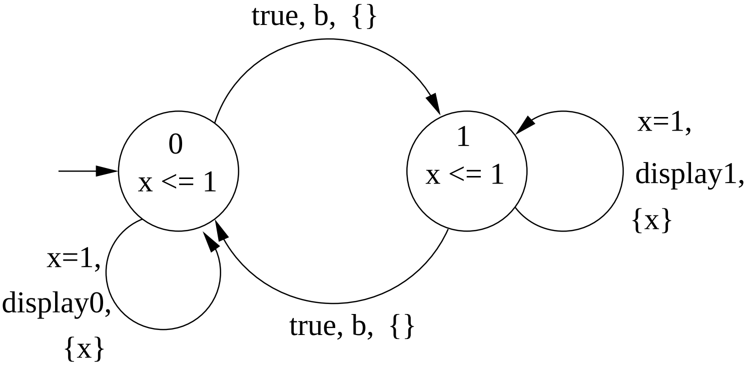
Detecting Zeno states



- (q, ν) 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 $x \leq 4$

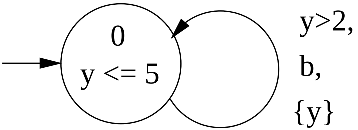
Parallel Composition of TSAs

A binary counter of *b*s



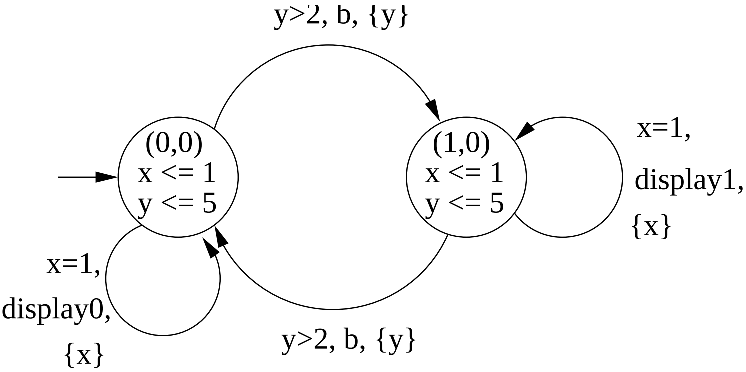
Parallel Composition of TSAs

An interference to free occurrence of *b*s



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 φ 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

Automata Theoretic Verification

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

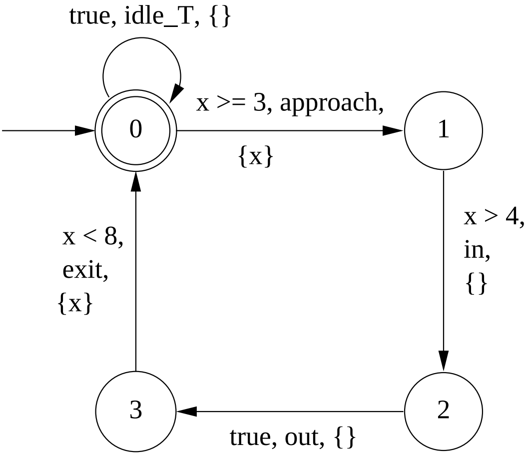
Minumum and Maximum Delay

- 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?

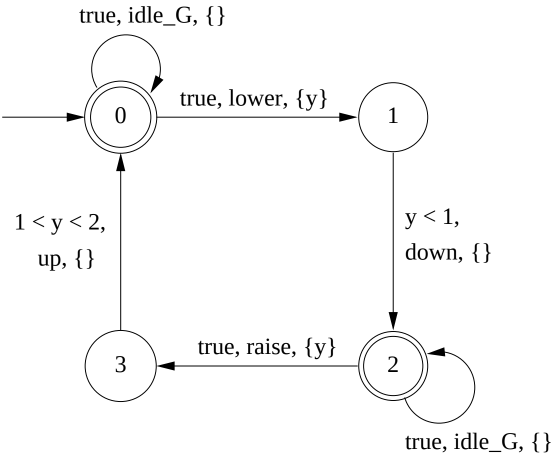
Priced (or Weighted) Timed Automata

- 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

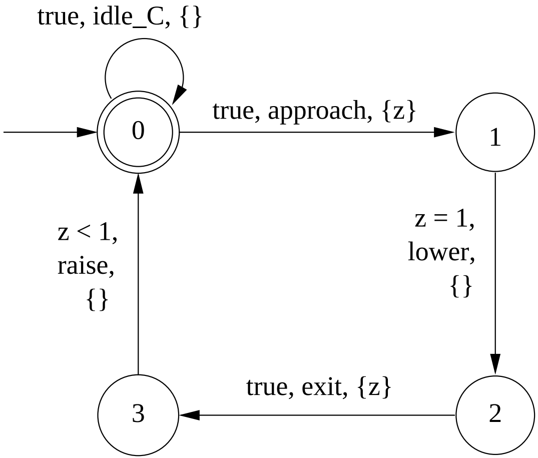
Train-Gate example: Train



Train-Gate example: Gate



Train-Gate example: Controller



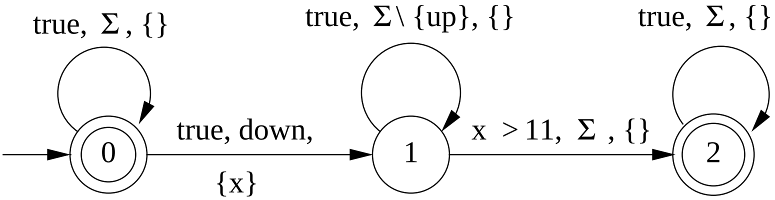
Train-Gate example: Verification

- **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



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

Automata theoretic verification

- Tool avilable: Open-KRONOS (Profoundus)
- Solve reachability problem on Timed Büchi automata
- Check language emptiness on Timed Büchi automata

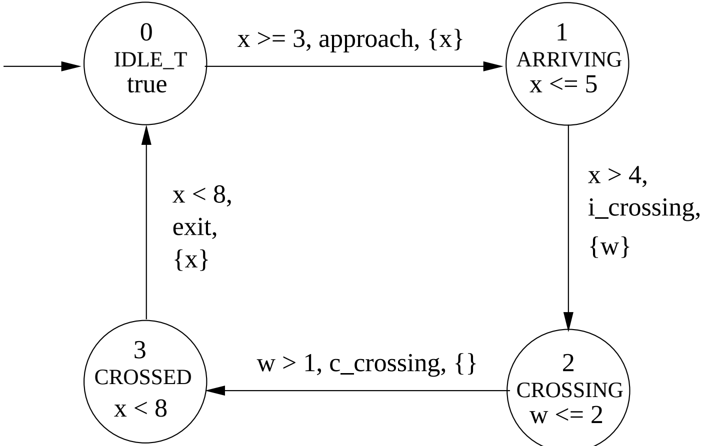
Model checking

- 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

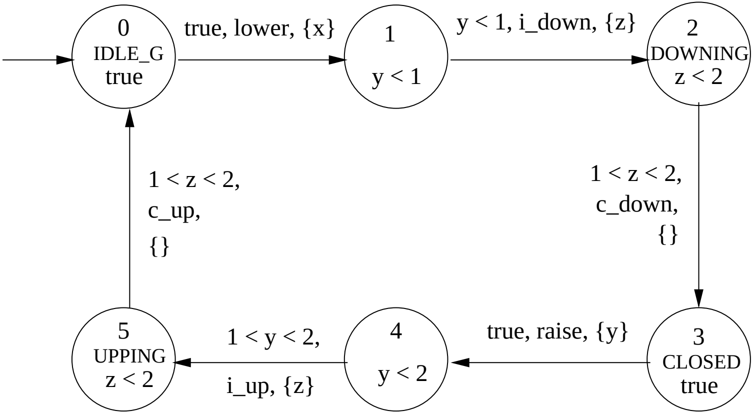
- Essentially Timed Safety Automaton A
- To facilitate formulas writing locations of A must be labeled with boolean variables (State-based approach vs. Action-based approach)
- Let's adapt the Train-Gate model used for automata theoretic verification
- We introduce new actions (initiation and termination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

TSA: Train

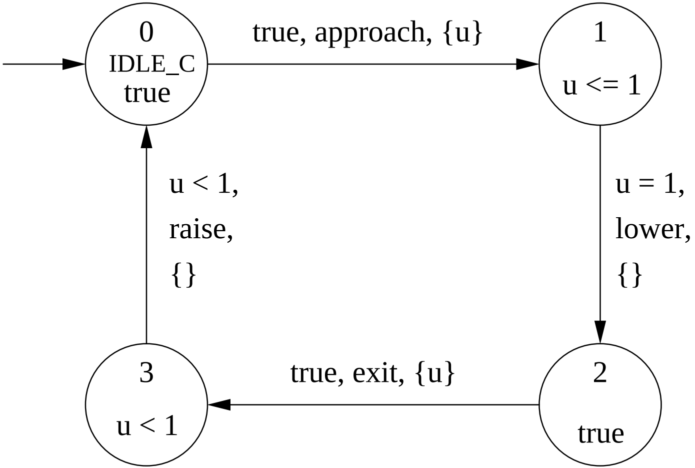


0
IDLE_T
true
x >= 3, approach, {x}
1
ARRIVING
x <= 5
x > 4,
i_crossing,
{w}
2
CROSSING
w <= 2
w > 1, c_crossing, {}
3
CROSSED
x < 8
x < 8,
exit,
{x}

TSA: Gate

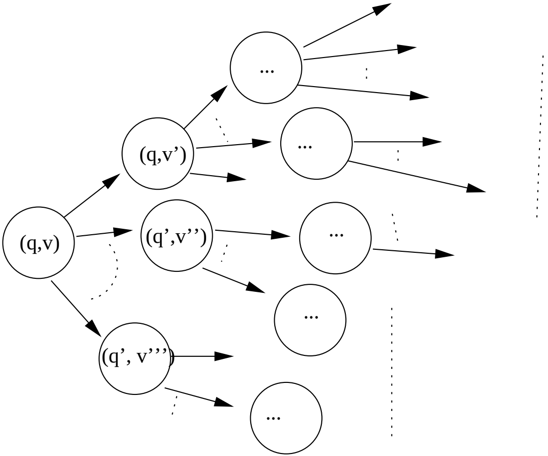

0
IDLE_G
true
true, lower, {x}
1
y < 1
y < 1, i_down, {z}
2
DOWNING
z < 2
1 < z < 2,
c_down,
{}
3
CLOSED
true
true, raise, {y}
4
y < 2
1 < y < 2,
i_up, {z}
5
UPPING
z < 2
1 < z < 2,
c_up,
{}

TSA: Controller



Logic TCTL

Think in terms of Computational Trees



TCTL: paths

- Consider a state (q, ν) 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, ν)
- If (q, ν) 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

$\varphi ::=$	ψ	Clock constraint
	$\mid b$	Boolean variable
	$\mid z.\varphi$	Freeze clock
	$\mid \neg\varphi$	Negation
	$\mid \varphi_1 \vee \varphi_2$	Disjunction
	$\mid \varphi_1 \, \exists\mathcal{U} \, \varphi_2$	Exist Path Until
	$\mid \varphi_1 \, \forall\mathcal{U} \, \varphi_2$	Forall Paths Until

TCTL: basic semantics

$$
\begin{array}{lll}
(q,\nu,\zeta) \models \psi & \text{iff} & \nu \cup \zeta \models \psi \\
(q,\nu,\zeta) \models b & \text{iff} & b \in \mathcal{P}(q) \\
(q,\nu,\zeta) \models z.\varphi & \text{iff} & (q,\nu,\zeta \backslash \{z\}) \models \varphi \\
(q,\nu,\zeta) \models \neg\varphi & \text{iff} & (q,\nu,\zeta) \not\models \varphi \\
(q,\nu,\zeta) \models \varphi_1 \vee \varphi_2 & \text{iff} & (q,\nu,\zeta) \models \varphi_1 \text{ or } (q,\nu,\zeta) \models \varphi_2 \\
(q,\nu,\zeta) \models \varphi_1 \exists\mathcal{U}\, \varphi_2 & \text{iff} & \exists\, \pi_{(q,\nu)} \in \Pi_A^{\infty}(q,\nu) \colon\ \exists p = (i,\delta) \in \mathtt{Pos}(\pi_{(q,\nu)}) \colon \\
 & & \quad s_i = (q_i,\nu_i) \wedge (q_i,\nu_i+\delta,\zeta+\Delta(p)) \models \varphi_2 \\
 & & \quad \wedge\, \forall p' = (j,\delta') \in \mathtt{Pos}(\pi_{(q,\nu)}).\ (p' \prec p \wedge s_j = (q_j,\nu_j)) \\
 & & \qquad\qquad \Rightarrow (q_j,\nu_j+\delta',\zeta+\Delta(p')) \models \varphi_1 \vee \varphi_2
\end{array}
$$

...

TCTL: useful syntactic sugar

$\exists\Diamond\varphi$ 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 $\mathit{true}\exists\mathcal{U}\varphi$

$\forall\Box\varphi$ 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 $\neg\exists\Diamond\neg\varphi$

TCTL: useful syntactic sugar

$\forall\Diamond\varphi$ expresses *inevitability*. It is satisfied by a state (q, ν) iff in all (q, ν)-paths a state in which φ is satisfied is reachable. The translation is *true*$\forall\mathcal{U}\varphi$

$\exists\Box\varphi$ expresses *possible invariance*. A state (q, ν) satisfies it iff there exists a (q, ν)-path along which the formula φ 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, ν) satisfies it iff there exists a (q, ν)-path along which a state satisfying φ is reachable within c time units. The translation uses the freeze quantifier: $z.\exists\Diamond(\varphi \wedge z \leq c)$
- $\forall\Diamond_{\leq c}\varphi$ is *bounded inevitability*. A state (q, ν) satisfies it iff in all (q, ν)-paths a state satisfying φ is reachable within c time units. The translation is $z.\forall\Diamond(\varphi \wedge z \leq c)$

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:

$$(\mathtt{IDLE\_T} \wedge \mathtt{IDLE\_G} \wedge \mathtt{IDLE\_C})$$

$$\rightarrow \forall\Box(\mathtt{CROSSING} \rightarrow \mathtt{CLOSED})$$

KRONOS: Train-Gate verification

Train-Gate liveness property:

$$(\mathtt{IDLE\_T} \wedge \mathtt{IDLE\_G} \wedge \mathtt{IDLE\_C}) \rightarrow$$
$$\forall\Box(\mathtt{CLOSED} \rightarrow \forall\Diamond_{\leq 11}\mathtt{IDLE\_G})$$

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

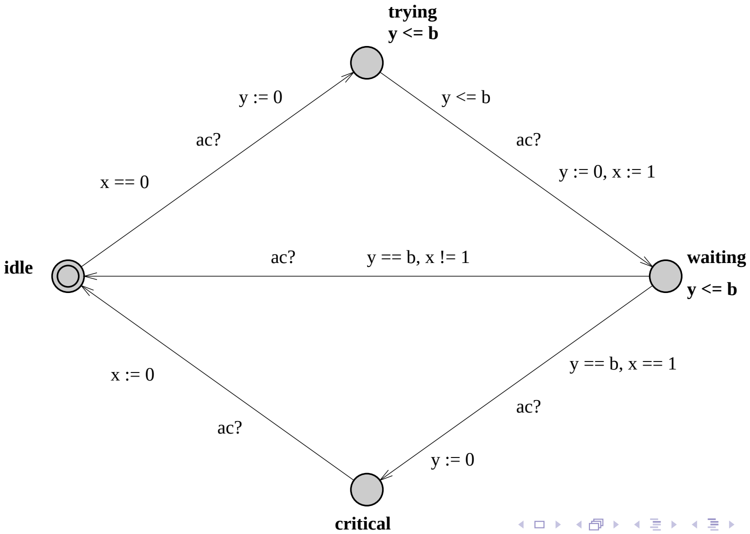
$$
\begin{aligned}
\varphi ::= \quad & \exists\Diamond\texttt{Expr} \\
& \mid \forall\Box\texttt{Expr} \\
& \mid \forall\Diamond\texttt{Expr} \\
& \mid \exists\Box\texttt{Expr} \\
& \mid \forall\Box(\texttt{Expr} \longrightarrow \exists\Diamond\texttt{Expr})
\end{aligned}
$$

`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

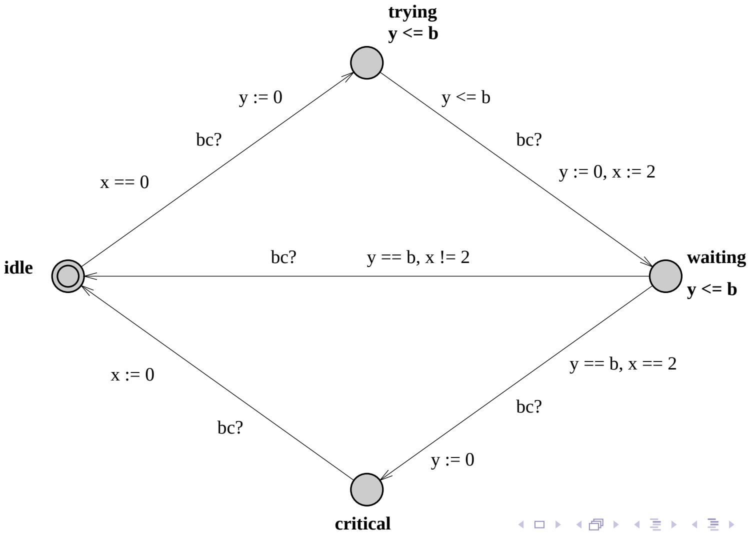
Fischer's Mutual excl. algorithm

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

Fischer's Mutex: Process 1



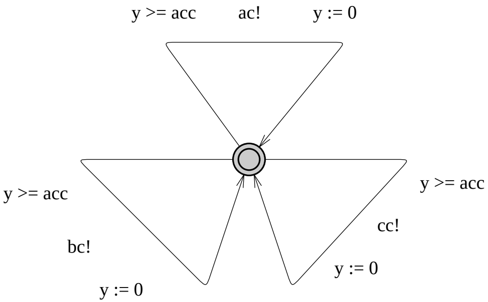
Fischer's Mutex: Process 2



Serialisation

- Access to variable `x` must be serialised in order to consider it atomic

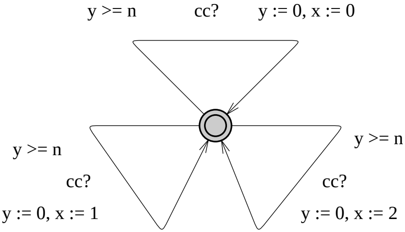
Fischer's Mutex: Serialiser



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



Fischer's Mutex: Verification

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

$$\forall\Box(P_1.\mathtt{critical} \rightarrow \neg P_2.\mathtt{critical})$$

$$\forall\Box(P_2.\mathtt{critical} \rightarrow \neg P_1.\mathtt{critical})$$

What about protocol liveness?


University of Camerino
UNICAM
Università di Camerino
1336
Thanks!