Timed CTL

VERIFICATION OF TA

Reachability (and beyond)

- The Region TS can be exploited to verify reachability of locations
- **Theorem**: Checking the reachability of a location in a timed automaton is PSPACE-complete.
  - The exponential part is due to the regions
  - The size of the region transition system is exponential in the number of clocks and the (binary representation of) constants with which clocks are compared.
- What about a temporal logic to express more interesting properties?
- Since bisimulation equivalence implies CTL equivalence (implication holds also for infinite state system) we could use CTL formulae over a TA, verifying them on the finite RTS imstead.
  - equivalence implies we get the same results
- However, CTL is somewhat poor to express timing constraints: no metric!
  - still useful!

CTL strengths/limitations

"the gate is always closed when the train is at the crossing"

- It does not contain any timing aspects, and can be described in CTL by:

$$\mathbf{AG(in \rightarrow down),}$$

- where in and down are locations in the timed automata Train and Gate, respectively.

- The (*timed liveness*) property "once the train is far, within 1 minute the gate is up for at least 1 minute" instead cannot be defined in CTL!

Timed CTL

- Extension of CTL with a metric Until Operator U^J

State formulae as usual, but allowing atomic clock constraints g (typically over the set of clocks in a timed automaton) :

- $\Phi ::= \text{true} \mid a \mid \mathbf{g} \mid \Phi \wedge \Phi \mid \neg\Phi \mid E\varphi \mid \forall\varphi$

with a in AP, **g in ACC(C)**

Timed CTL extends CTL with atomic clock constraints

- Path formulae: $\varphi ::= \Phi\ \mathbf{U}^J\ \Phi$
  where **J** is an interval whose bounds are natural numbers.

- Define $F^J\ \Phi = \text{true}\ U^J\ \Phi$: Φ holds sometimes during the interval J in the current path
- Derived operators:
- $EG^J\ \Phi = \neg AF^J\ \neg\Phi$ there exists a path s.t. Φ holds during the interval J.
- $AG^J\ \Phi = \neg EF^J\ \neg\Phi$ for all paths, Φ holds during the interval J.

Examples

"the light cannot be continuously switched on for more than 2 minutes"

$$AG(on \rightarrow AF_{\leq 2}\ \neg on).$$

"the light will stay on for at least 1 time unit and then switch off"

$$AG\ (on \wedge (x = 0)) \rightarrow (AG_{\geq 1}\ on \wedge AF_{>1}\ off\ )$$

(using a clock reset—test if 0-- to specify the time instant when the light is switched on, as in the TA for the Light Switch .

The train

"once the train is far, within 1 minute the gate is up for at least 1 minute"

$$AG(far \rightarrow AF^{\leq 1} AG^{\geq 1} up).$$

"the train needs at least 2 minutes to reach the crossing after transmitting the "approach" signal"

$$AG\ (near \wedge (y = 0)) \rightarrow AG^{\geq 2} \neg in$$

(the clock constr. y=0 denote the instant when the train signals its approach, see the TA for the train)

"the train needs at most five minutes to pass the crossing since its approach" :

$$AG\ (near \wedge (y = 0)) \rightarrow AF^{\leq 5} far .$$

Semantics if TCTL

- Path formulae are required to hold only on *time-divergent paths*
- If s = <l,η> is a state then

$$
\begin{array}{lll}
s \models \text{true} & & \\
s \models a & \text{iff} & a \in L(\ell) \\
s \models g & \text{iff} & \eta \models g \\
s \models \neg\Phi & \text{iff} & \text{not } s \models \Phi \\
s \models \Phi \wedge \Psi & \text{iff} & (s \models \Phi) \text{ and } (s \models \Psi) \\
s \models \exists\varphi & \text{iff} & \pi \models \varphi \text{ for some } \pi \in Paths_{div}(s) \\
s \models \forall\varphi & \text{iff} & \pi \models \varphi \text{ for all } \pi \in Paths_{div}(s).
\end{array}
$$

Eventually, Globally

for time-divergent path $\pi \in s_0 \xRightarrow{d_0} s_1 \xRightarrow{d_1} \ldots$

$$\pi \models \Diamond^J \Phi \quad \text{iff} \quad \exists\, i \geqslant 0.\, s_i{+}d \models \Phi \text{ for some } d \in [0, d_i] \text{ with } \sum_{k=0}^{i-1} d_k + d \in J.$$

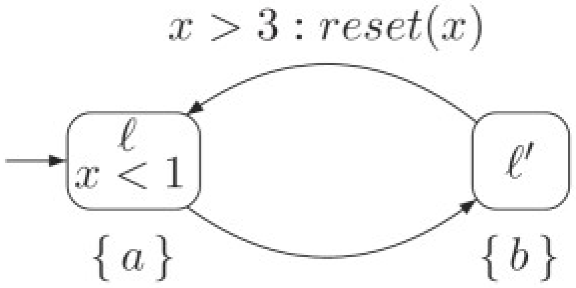
i.e., it is satisfied whenever a Φ-state is reached at some time instant t ∈ J.

$$\pi \models \Box^J \Phi \quad \text{iff} \quad \forall i \geqslant 0.\, s_i{+}d \models \Phi \text{ for any } d \in [0, d_i] \text{ with } \sum_{k=0}^{i-1} d_k + d \in J.$$

all states visited by π in the time interval J satisfy Φ.

Informal definition of Until

  - Formal Semantics of Until can be found in B&K
- A Time-divergent path π satisfies $\Phi U^J \Psi$ whenever at some time point in J, a state is reached satisfying Ψ and at all previous time instants $\Phi \vee \Psi$ holds.
  - NB: In LTL/CTL, Φ U Ψ is equivalent to $(\Phi \vee \Psi)$ U Ψ.



$$\Phi = \forall(a \,\mathsf{U}^{>1} b)$$

$$\pi \in \underbrace{\langle \ell, 0\rangle}_{s_0} \xRightarrow{0.5} \underbrace{\langle \ell', 0.5\rangle}_{s_1} \xRightarrow{2.5} \underbrace{\langle \ell, 0\rangle}_{s_2} \ldots .$$

$$\pi \models a \,\mathsf{U}^{>1} b$$

$$s_1{+}d \models b \text{ for some } d \in [0, 2.5] \text{ such that } 0.5{+}d > 1$$

$$s_0{+}d' \models a \text{ for all } d' \in [0, 0.5] \quad \text{and} \quad s_1{+}d' \models a \vee b \text{ for all } d' \in [0, d].$$

TCTL semantics for TA

Let TA be a timed automaton with clocks C and locations Loc. For TCTL state formula Φ, the *satisfaction set* $Sat(\Phi)$ is defined by:

$$Sat(\Phi) \; = \; \{ \, s \in Loc \times Eval(C) \mid s \models \Phi \, \}.$$

The timed automaton TA satisfies TCTL state formula Φ if and only if Φ holds in all initial states of TA:

$$TA \models \Phi \quad \text{if and only if} \quad \forall \ell_0 \in Loc_0. \, \langle \ell_0, \eta_0 \rangle \models \Phi$$

where $\eta_0(x) = 0$ for all $x \in C$. ■

Model checking?

- The construction of the RTS(TA) can be modified to add regions introduced by a TCTL formula Φ
  - When defining RTS we have to consider also clock constraints occurring in the formula Φ (i.e., the max constant c_x for each x)
- Result is a transition system RTS(TA, Φ)
  - Unfortunately, RTS(TA,Φ) is not bisimilar to TS(TA) (because of elimination of certain delay transitions)…but this still works since it is stutter-equivalent.
- There is a construction eliminating timing parameters from every TCTL subformula formula, by just adding new clocks
  - Ex: $\mathbf{EF}_{\leq 5}\, \varphi$ becomes $\mathbf{EF}((z \leq 5) \wedge \varphi)$

Given a state s, s |= $EF_{\leq 5}\, \varphi$ iff s[z:=0] |= $EF((z \leq 5) \wedge \varphi)$

  - Details are skipped here…
  - Result is a formula Φ’

Elimination of clocks

- The reason this elimination works is that a temporal logic extending CTL with clocks, rather than dense time operators, subsumes TCTL.
  - The idea of this logic, called **TCTLc**, is to consider a new set of clocks – the formula clocks – and to add atomic constraints “$x<c$, $x=c$” in the logic and a new operator (“in”) to reset a given clock to zero.
  - A TCTL formula Φ can always be transformed into an equivalent TCTLc formula Φ'
  - We can also define a Region TS, called RTS(A,Φ') introducing the formula clocks in RTS(A) and their constraints.

Model checking!

- TCTLc that can be interpreted over the Region TS(TA,Φ') as a CTL formula if clock constraints are considered just as atomic propositions and non-Zeno behaviors are excluded.

- Therefore, under a non-Zeno assumption:
- *TA |= Φ under the TCTL semantics is equivalent to RTS(TA,Φ) |= Φ' under the CTL semantics*
- Standard CTL model checking can thus be used
  - In practice, various optimizations are introduced

Final result

Theorem 9.68. Time Complexity of TCTL Model Checking

For timed automaton TA and TCTL formula Φ, the TCTL model-checking problem $TA \models \Phi$ can be determined in time $\mathcal{O}((N+K) \cdot |\Phi|)$, where N and K are the number of states and transitions in the region transition system $RTS(TA, \Phi)$, respectively.

- States of the RTS are exponential in the number of clocks and in the (binary repr. of) the constants:

Theorem 9.69. Complexity of TCTL Model Checking

The TCTL model-checking problem is PSPACE*-complete.*

- In practice, most often model checkers do not support full TCTL
  - Symbolic techniques improving efficiency of verification cannot be easily extended to full TCTL.

Other results

- Model checking safety, reachability, ω-regular properties, LTL and CTL for TA is PSPACE-complete
- *TCTL satisfiability is undecidable*
- The model-checking problem for Timed LTL is undecidable
  - Timed LTL is LTL extended with clock variables and clock constraints: G(warning → x.F(alarm ∧ x<10)
- Model checking Metric Temporal Logic (MTL) against TA is decidable, but only under finite runs (not primitive recursive...)
  - MTL has U_J operator, but no clocks. It is less expressive than Timed LTL.

TIMED LANGUAGES

Timed ω-word

- Intuition: an infinite sequence of "states", each with a timestamp
  - ⟨a, 0.3⟩ ⟨b, 1.6⟩ ⟨a, 4.1⟩ ⟨b, 4.25⟩ ⟨a, 9.8⟩...
  - ⟨up, 2.5⟩ ⟨up, 2.6⟩ ⟨down, 2.62⟩ ⟨up, 2.624⟩ ⟨up, 2.625⟩ ⟨up, 2.6252⟩...
  - ⟨{up, no_alarm}, 0.1⟩ ⟨{up, no_alarm}, 27.3⟩ ⟨{down, alarm}, 150.0⟩...
- *Timed* ω*-word* (or, more simply, *timed word*): a sequence of values (σ_1,τ_1) (σ_2,τ_2) … where each symbol σ_i is associated with a real-valued timestamp τ_i

Timed language

- Def: *time sequence* $\tau = \tau_1 \tau_2 \tau_3 \ldots$
  an infinite sequence of time values $\tau_i \geq 0 \in R_{\geq 0}$ s.t.
  - $\tau_i \leq \tau_{i+1}$ for all $i \geq 1$ (*monotonicity*)
    - if we impose $\tau_i < \tau_{i+1}$ then *strict monotonicity*
  - for all $t \in R$, there is some $i \geq 1$ s.t. $\tau_i > t$ (*progress*)
- Def: *timed word* σ over alphabet (i.e., finite set of symbols) $\mathscr{A}$ is a pair (σ, τ), where $\sigma = \sigma_1 \sigma_2 \ldots$ $(\in \mathscr{A}^\omega)$, is an ω-word and τ is a time sequence
- Def: *timed language* over $\mathscr{A}$ is a set of timed words over $\mathscr{A}$

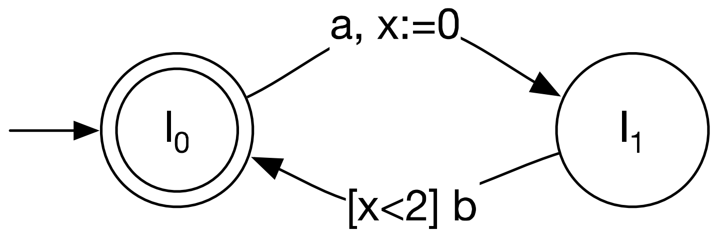
The Untime operation

- The *Untime* operation on a timed language discards the time values associated with symbols
- Def: for a timed language $\mathscr{L}$ over $\mathscr{A}$,
  Untime($\mathscr{L}$) = $\{\sigma \in \mathscr{A}^{\omega} \mid$ there is a τ s.t. $(\sigma, \tau) \in \mathscr{L}\}$
  - Untime($\mathcal{L}$) is a set of ω-words

Timed Automata as Recognizers of timed languages

- We consider timed words in the alphabet of the actions, called the *input alphabet*
- No atomic propositions in the locations
- *A set of final locations*
- A Timed Automaton is a tuple $\langle L, \Sigma, C, E, l_0, J, F \rangle$:
  - *L* is a finite set of locations
  - Σ is a finite set of input symbols
  - *C* is a finite set of clocks
  - $l_0 \in L$ is the initial location
  - *E* is a set of edges: $E \subseteq L \times B(C) \times \Sigma \times 2^C \times L$ where B(C) is the set of clock constraints
  - J: $L \rightarrow B(C)$ are the invariants
  - *F* is the set of *final locations*
- Everything else is as usual, but a timed word is accepted if there is a path leading infinitely often to a final location

Example of Timed Automaton accepting a Timed Language



- Accepts the timed language
  $\mathcal{L} = \{((ab)^{\omega}, \tau) \mid \text{for all } i, (\tau_{2i} < \tau_{2i-1} + 2)\}$

Run and acceptance for TA

- Def: a run $r_{(\sigma,\tau)}$ over a timed word (σ, τ), with $\sigma=\sigma_1\sigma_2\ldots$, and $\tau=\tau_1\tau_2\ldots$, is an infinite sequence $r_{(\sigma,\tau)}$: $\langle l_0, \eta_0\rangle \xrightarrow{\sigma_1,\tau_1} \langle l_1, \eta_1\rangle \xrightarrow{\sigma_2,\tau_2} \langle l_2, \eta_2\rangle \ldots$

- where $l_i \in L$, η_i are clock evaluations s.t.:
  - $\eta_0(x)=0$ for all $x\in C$ (initiation)
  - $\forall i\geq 1$, there is $e\in E$ s.t. $e=\langle l_{i-1}, \gamma_i, \sigma_i, \hat{C}_i, l_i\rangle$ and
    $(\eta_{i-1}+\tau_i-\tau_{i-1})$ satisfies both γ_i and $J(l_{i-1})$,
    $\eta_i =[\hat{C}_i \longmapsto 0](\eta_{i-1}+\tau_i-\tau_{i-1})$ and η_i satisfies $J(l_i)$
- *inf*($r_{(\sigma,\tau)}$) is the of locations visited *infinitely often* in the run

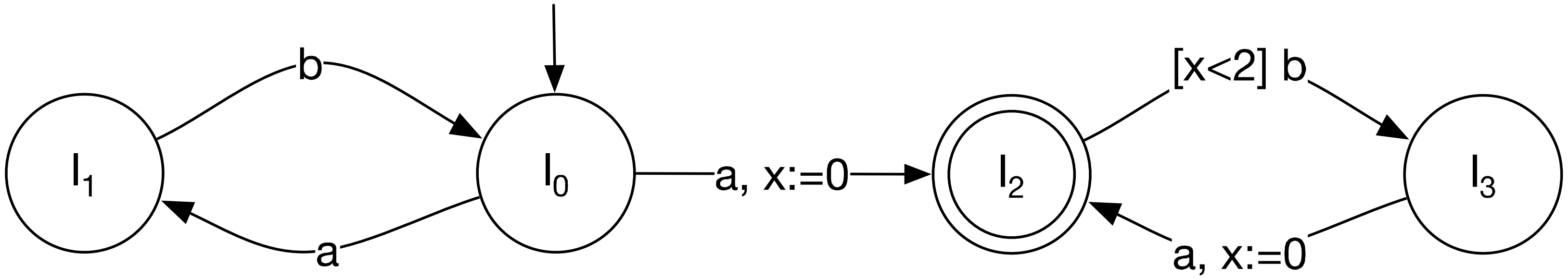
A timed word (σ, τ) is accepted by a TA $\langle L, \Sigma, C, E, l_0, J, F\rangle$

iff

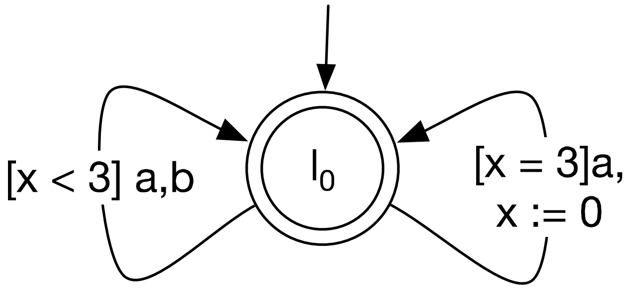
there exists a run $r_{(\sigma,\tau)}$ s.t. *inf*($r_{(\sigma,\tau)}$) $\cap$ F $\neq \emptyset$

More examples of TA

- TA that recognizes the timed language $\mathcal{L}_{crt} = \{((ab)^\omega, \tau) \mid \text{exists } i \text{ s.t. for all } j \geq i \ (\tau_{2j} - \tau_{2j-1} < 2)\}$



- TA that recognizes the timed language $\mathcal{L}_{per} = \{\{a,b\}^\omega, \tau) \mid \forall k \ \exists j \text{ s.t. } (\square_j = 3k \text{ and } \sigma_j = a)\}$

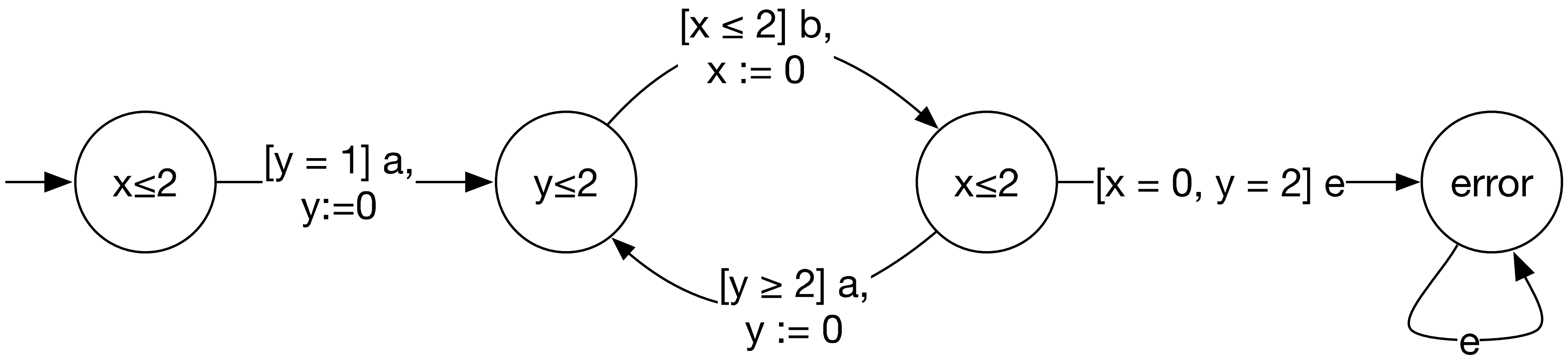


Timed regular languages

- Def: a timed regular language is a language that is accepted by some TA
  - $\mathcal{L}_{crt}$ and $\mathcal{L}_{per}$ are timed regular languages
  - $\mathcal{L}_{nr} = \{((ab)^\omega, \tau) \mid \forall j\ (\tau_{2j} - \tau_{2j-1} < \tau_{2j+2} - \tau_{2j+1})\}$
    is *not* regular
- If $\mathcal{L}$ is an ω-regular language $\subseteq \mathscr{A}^\omega$,
  $\mathcal{L}_t = \{(\sigma, \tau) \mid \sigma \in \mathcal{L}\}$ is a timed regular language
  - τ is any!

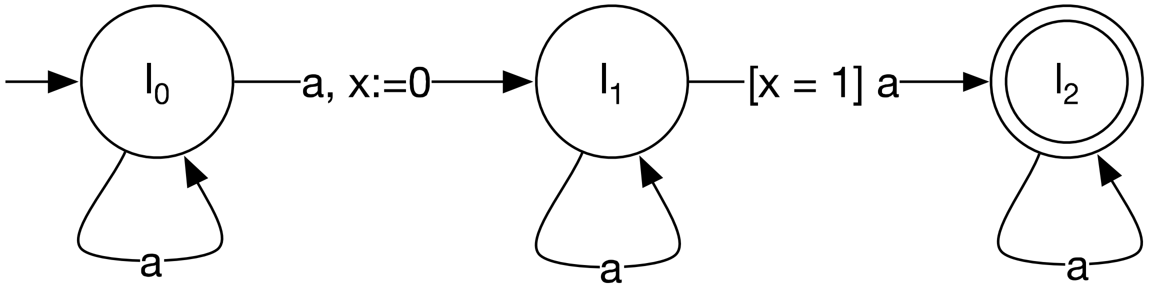
Untimed languages of TA are ω-regular

- Given a TA that accepts a language $\mathcal{L}$, there is a BA that accepts Untime($\mathcal{L}$)
  - that is, given a timed regular language $\mathcal{L}$, Untime($\mathcal{L}$) is ω-regular
  - Eliminating timing constraints from the transitions is not enough!
    - in the following TA, state *error* is not reachable due to the timing constraints; if we simply remove them, then it becomes easily reachable



Properties of TA

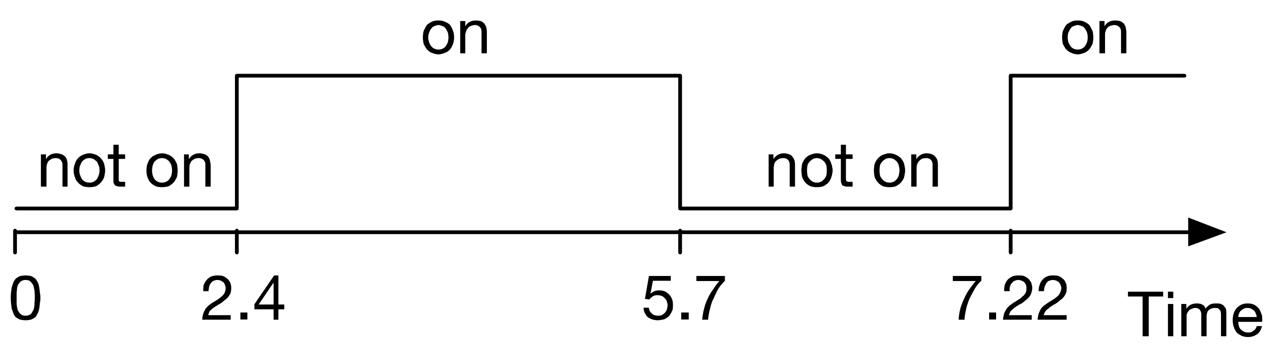
- TA are *closed* under union and intersection, but are *not closed* under complementation
  - A noncomplementable TA: $\mathcal{L}_{nc} = \{(a^\omega, \tau) \mid \exists i, \exists j>i\ (\tau_j = \tau_i + 1)\}$



  - An immediate consequence is that TA cannot be determinized
  - Nonclosure under complement strongly limits automata-based model checking
    - Language-inclusion and language-equivalence are indeed undecidable!
  - *Emptiness is decidable by verifying emptiness of Region Automaton (similar to the Region Transition System)*

Signals

- Each instant of the time domain is associated with the "state" of the system in that instant



Continuous time semantics

- It is possible to define TA (and temporal logic) semantics *over signals* rather than over Timed Words
  - The semantics in terms of Transition Systems is based on Timed Words
  - Linear-time TLs are more naturally interpreted over signals
- It is more realistic, but decision problems become harder to compute
  - For instance, the logic MTL becomes undecidable over continuous-time semantics
- NB: continuous-time semantics = signals
- pointwise semantics = timed words

Model Checking TA for LTL?

- Model checking TA for LTL, over both the point-wise and continuous semantics, is PSPACE-complete.
  - Just build the BA equivalent to the negation of a formula and build the product with the TA
  - Result is still a TA (TA are closed under intersection)
  - Verify emptiness of the result
- Unfortunately, most metric extensions of LTL are undecidable
  - Metric Interval Temporal Logic is decidable, but in EXPTIME

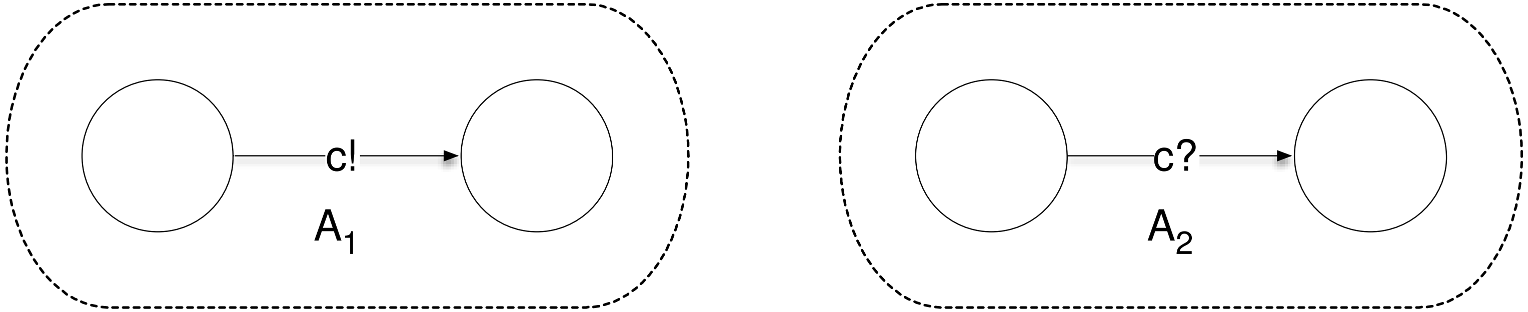
The Uppaal tool

Introduction

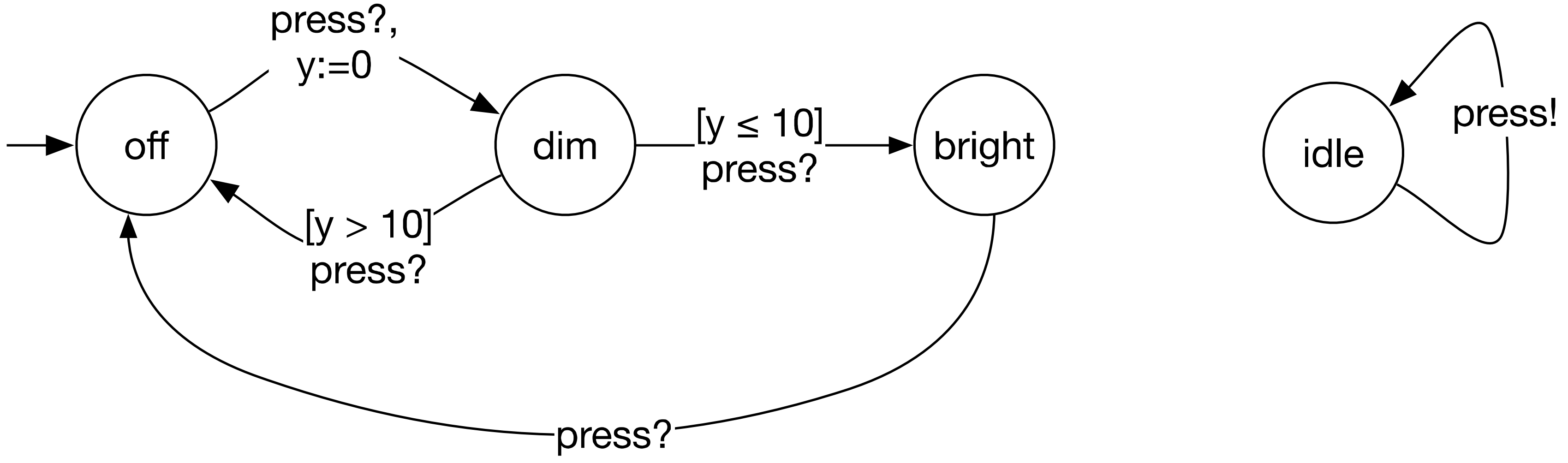
- Not the first tool for the verification of Timed Automata
  - KRONOS is the original one
- Developed by University of Uppsala and University of Aalborg
  - UPPsala + AALborg = UPPAAL
- www.uppaal.org
  - academic version
  - there is also a commercial version (www.uppaal.com)
- Some references (available from the Uppaal website):
  - Timed Automata: Semantics, Algorithms and Tools, Johan Bengtsson and Wang Yi. LNCS 3098, 2004
  - Tutorial on Uppaal

Building models from components

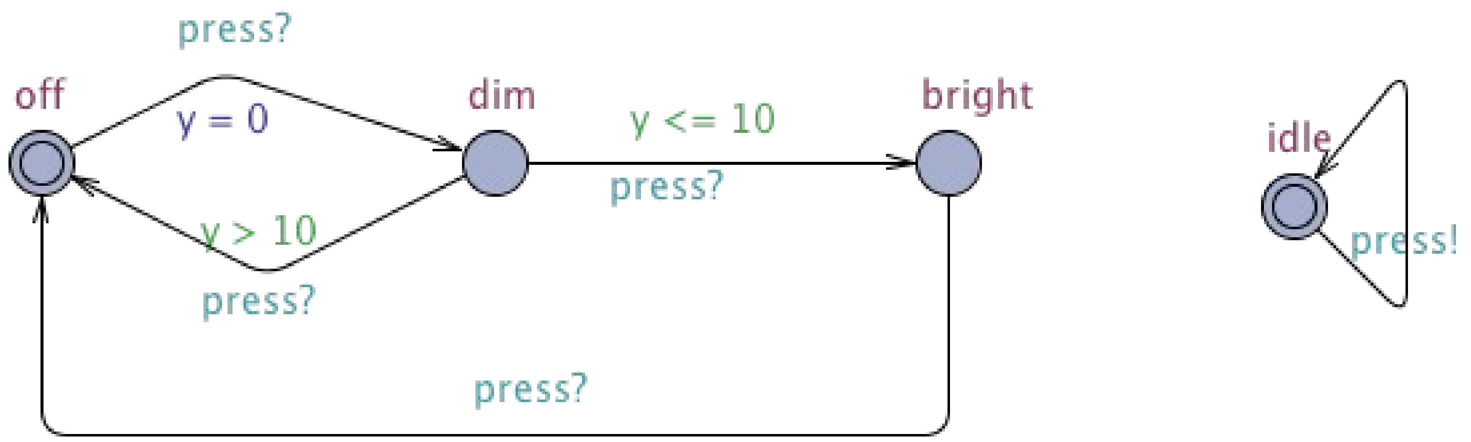
- Uppaal allows users to build complex models as *networks of Timed Automata*
  - the model is broken down into interacting components
- The interaction between components is achieved through *channels*
  - one component issues a message through a channel, the other receives it
- If H is the set of channels, the symbol on a transition is now of type $\Sigma \cup H! \cup H?$
  - c!: component sends a message through channel *c*
  - c?: component receives a message through channel *c*



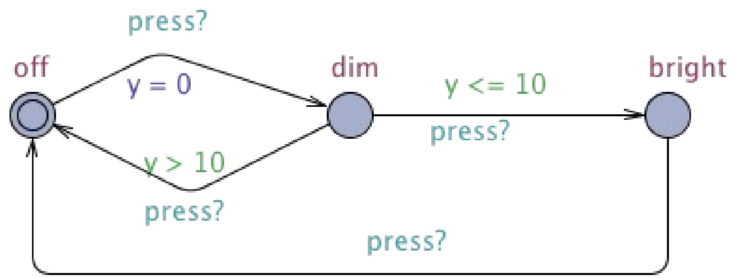
Example of network of TA



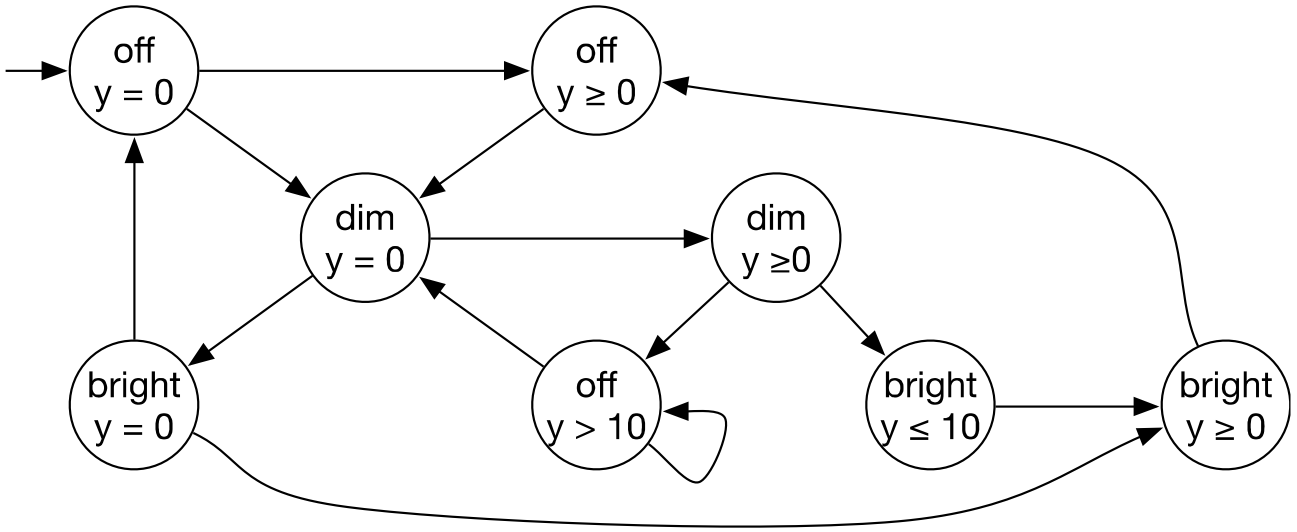
Example in Uppaal



Regions vs. zones



- The region abstraction is very fine, this automaton has over 50 states
- Clock *zones* are a coarser abstraction
  - a zone corresponds to a clock constraint, and covers many regions
- Zone graph for the previous automaton:

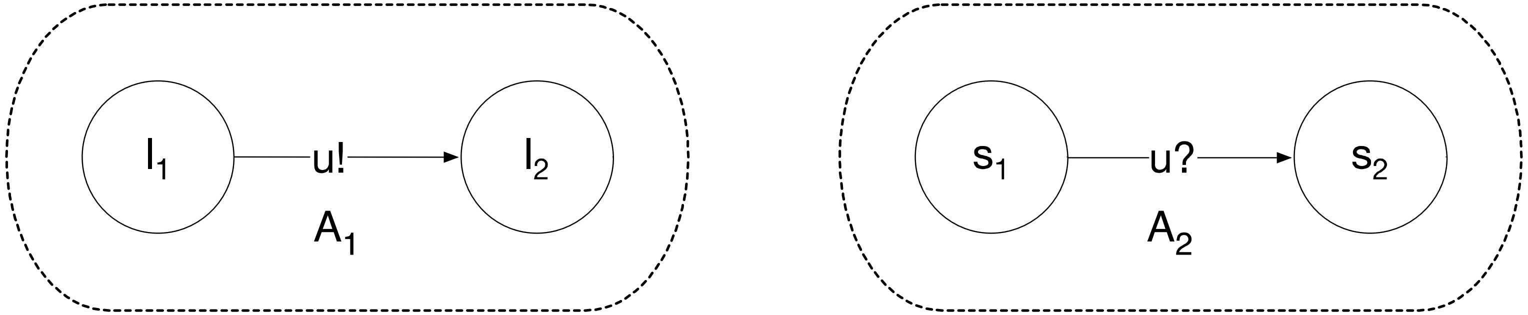


Committed and urgent locations

- Time cannot pass in locations that are *committed* or *urgent*
- When a location is *urgent*, this is like resetting a clock y upon entering the location, and adding invariant $y \leq 0$ to the location
  - but interleaving with normal locations is possible
- When a location is *committed*, not only time *cannot* pass, but in addition the only transition that can be taken must exit a committed location
  - i.e., only interleaving with other committed locations is possible
- Useful to reduce interleaving and the number of clocks

Urgent channels

- If a synchronization channel is declared as *urgent*, the synchronization occurs as soon as it is enabled



  - as soon as the two automata are in locations l_1 and s_1, the synchronization occurs
  - transitions with urgent channels cannot have clock constraints, for efficiency reasons

The query language

- Properties to be checked in Uppaal can be expressed using a subset of (T)CTL
  - A[] Expression
    E<> Expression
    E[] Expression
    A<> Expression
    Expression --> Expression
    - where Expression is a formula describing a (combination of) location, guard on variables, clock constraint
      - for example, error, or Lamp.bright or Lamp.dim
      - deadlock is a special keyword to describe deadlocked states
      - clock constraints allow some TCTL-like constraints: the query language can be considered a “safety subset” of TCTL.

The query language (2)

- E<>P “P is reachable”
  - there is a path (i.e., a run) from the initial state that leads to a state in which P holds
    - i.e., a state in which P holds is reachable from the initial state
- A<>P “P is inevitable”
  - all paths from the initial state are such that eventually a state in which P holds is reached
- A[]P   “P is an invariant”
  - all states that are reachable from the initial state are such that P holds
- E[]P “P is potentially always true”
  - there is a path from the initial state such that P holds in all states that are along that path ()
- P --> Q   “P leads to Q”
  - along all paths from the initial state, all states in which P holds are such that eventually Q holds
  - this is an abbreviation for the CTL property
    A[] (P imply A<> Q)