SBMF 2024
27th Brazilian Symposium on Formal Methods

ETMF 2024

9TH SCHOOL OF THEORETICAL COMPUTER SCIENCE AND FORMAL METHODS

SERRA / BRAZIL
DECEMBER 3RD, 2024

Safety in Real-Time Systems. Modeling and Verification with Timed Automata

Prof. Ciprian Teodorov

ciprian.teodorov@ensta-bretagne.fr

Lab-STICC

ENSTA Bretagne

ensta-bretagne.fr/teodorov

Academia @ Lab-STICC, ENSTA Bretagne, Brest

[23-...] *Full Professor*

[15-23] *Associate professor*

Lead the OBP2 *Semantic Diagnosis & Formal Verification* Lab. (http://www.obpcdl.org/)

[13-15] *Postdoc*

Verification MBSE, Concurrent system modeling, and verification (https://gemoc.org/)

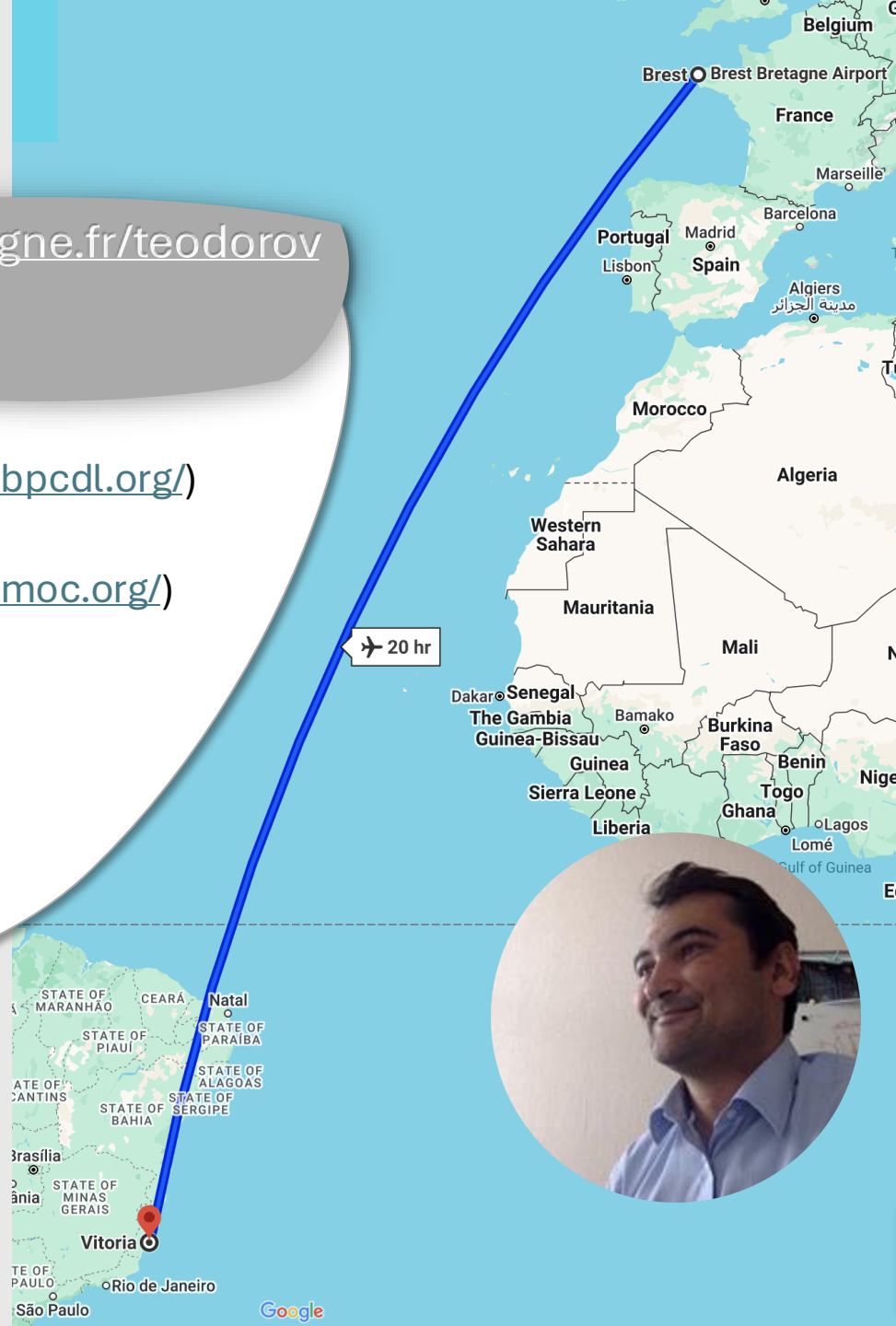
Industry @ Dolphin Integration - Grenoble Area

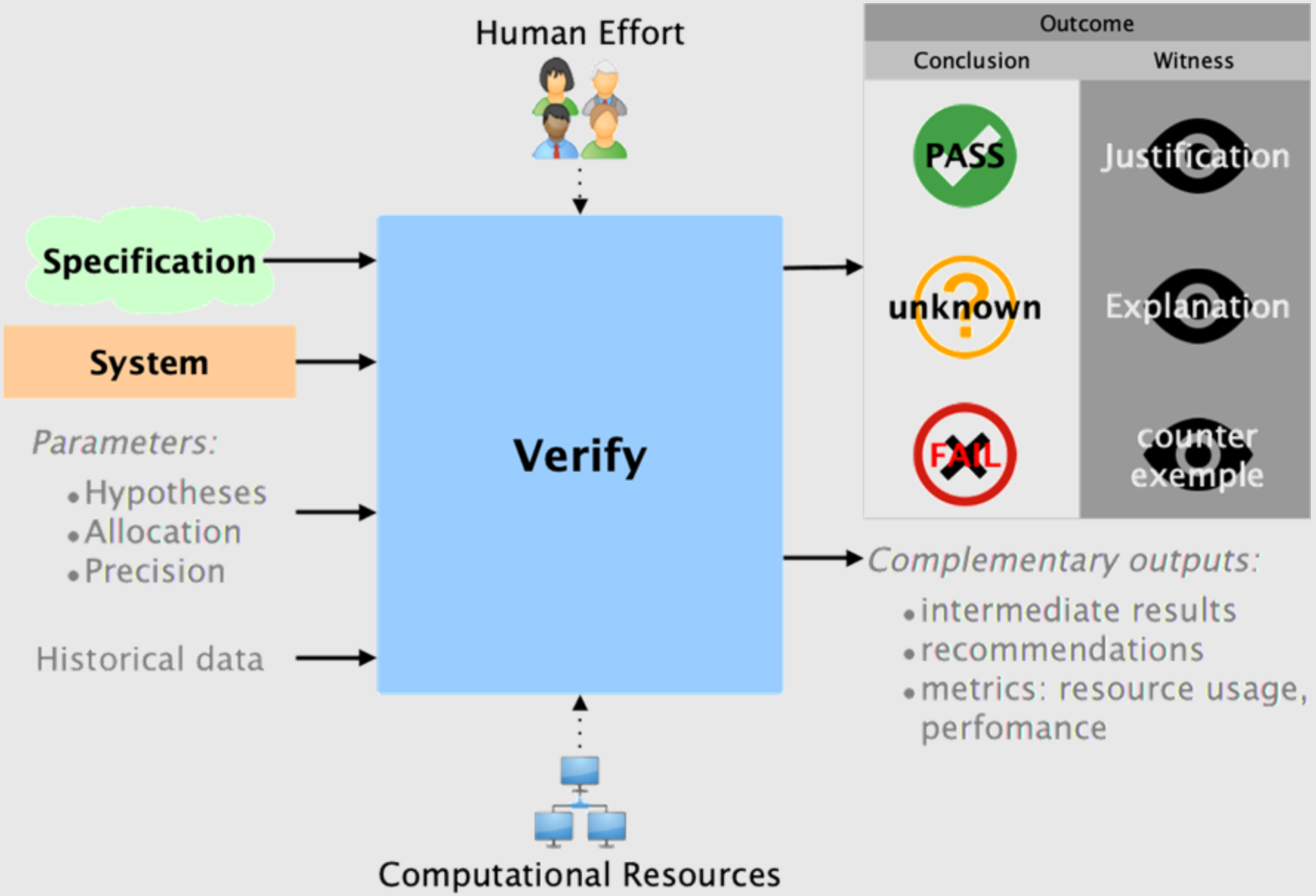
[11-13] *Electronics CAD engineer*

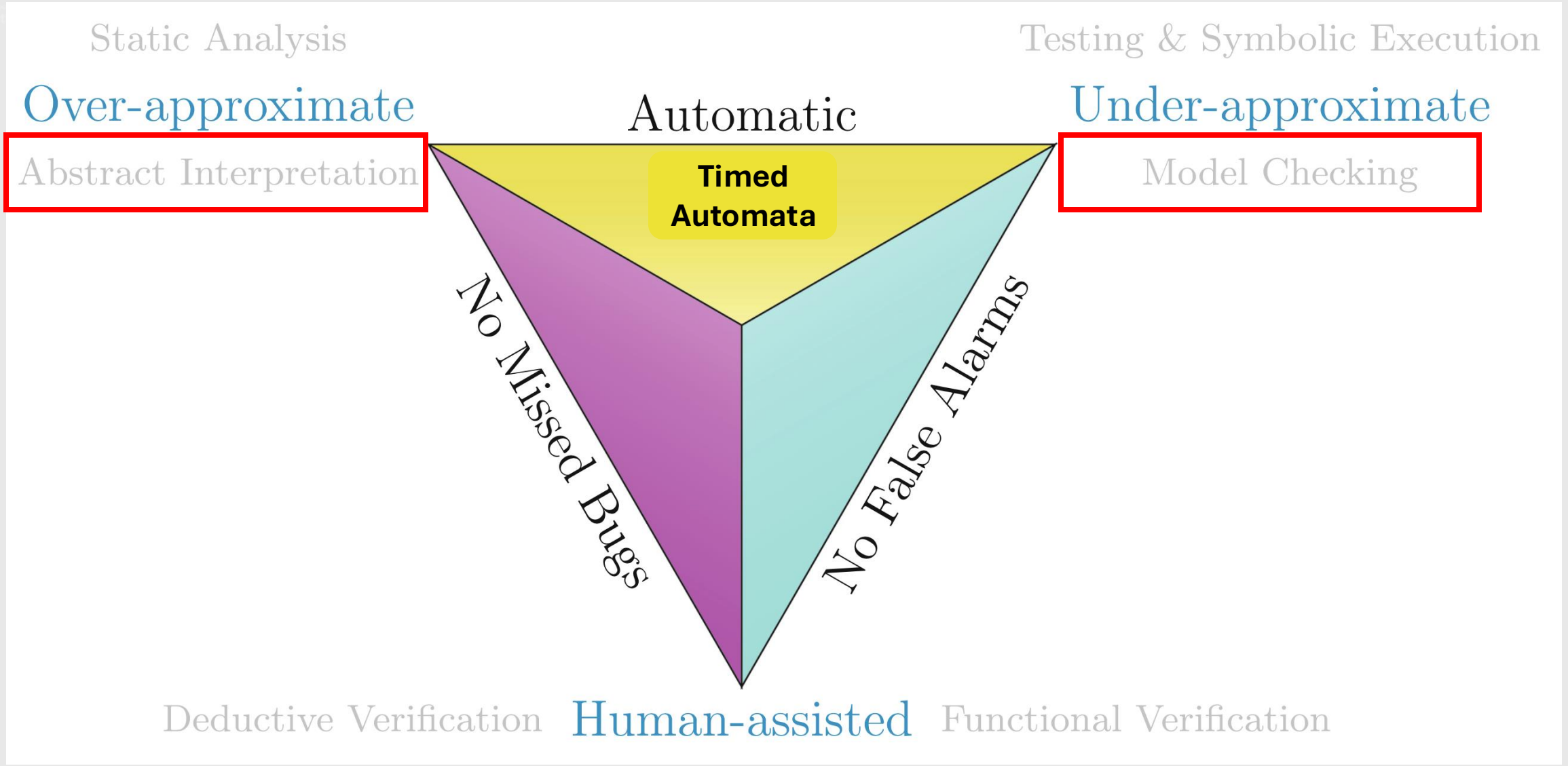
Compilation of VHDL, VHDL-AMS for mixed-signal simulation

PhD in Computer Science @ UBO – Brest

[08-11] Model-driven physical design for future nanoscale architectures




Human Effort
Specification
System
Parameters:
Hypotheses
Allocation
Precision
Historical data
Verify
Computational Resources
Outcome
Conclusion
Witness
PASS
Justification
unknown
Explanation
FAIL
counter exemple
Complementary outputs:
intermediate results
recommendations
metrics: resource usage, perfomance



Brain, M., Polgreen, E. (2025). **A Pyramid Of (Formal) Software Verification.** In: Platzer, A., Rozier, K.Y., Pradella, M., Rossi, M. (eds) Formal Methods. FM 2024. Lecture Notes in Computer Science, vol 14934. Springer, Cham. https://doi.org/10.1007/978-3-031-71177-0_24

Model checking is a formal verification technique that is highly accessible to engineers.

Many use cases:

- Circuit verification : **Intel** [1]
- Driver verification : **Microsoft** (SLAM [2])
- Distributed system verification : **Amazon** (TLA+ [3])

ETMF'24
Lecture 1

PROF. DR. JEFFERSON O. ANDRADE
FEDERAL INSTITUTE OF ESPÍRITO SANTO
INTRODUCTION TO SYSTEMS SPECIFICATION WITH TLA+

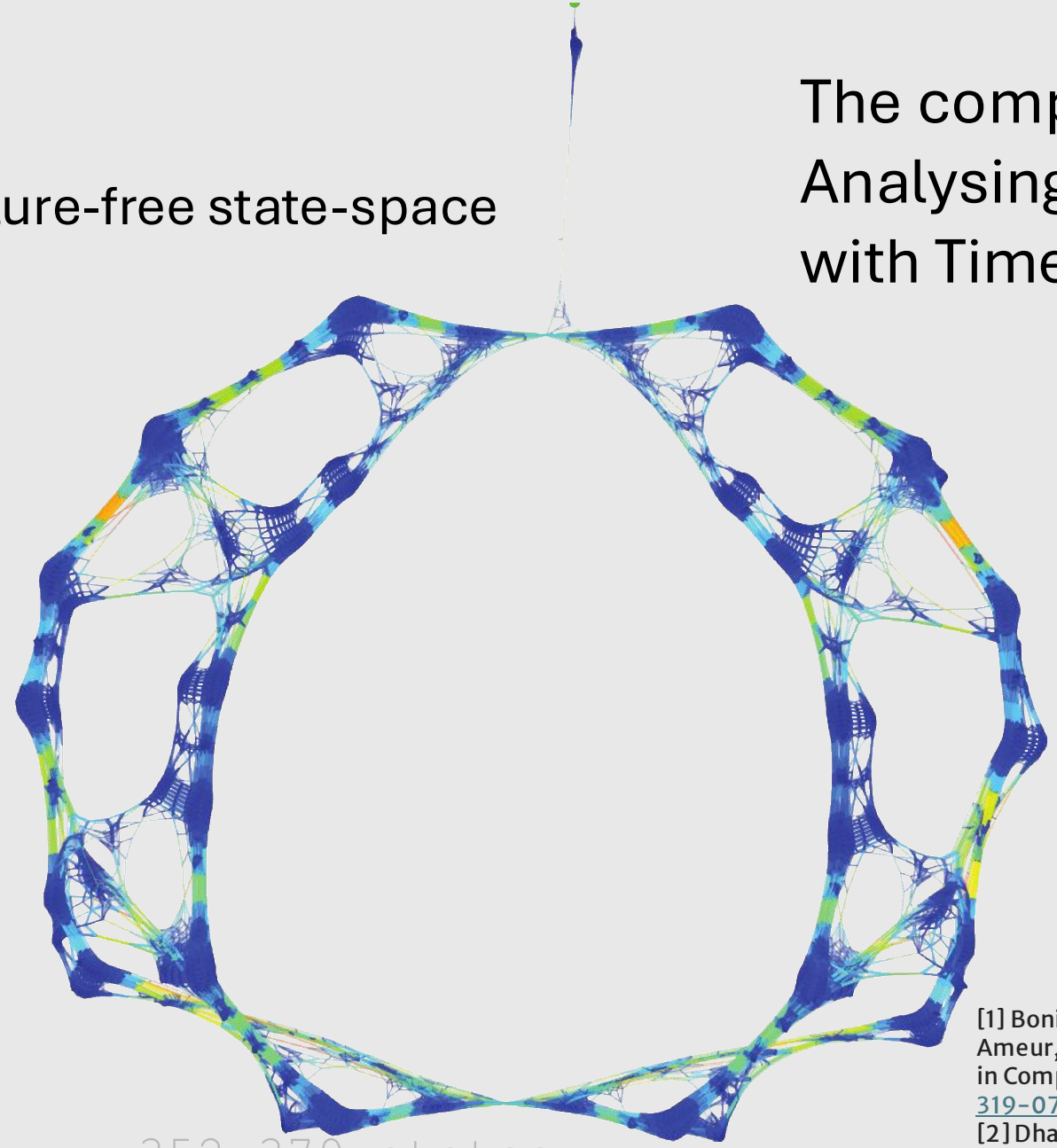
[1] Fix, L. (2008). **Fifteen Years of Formal Property Verification in Intel.** In: Grumberg, O., Veith, H. (eds) *25 Years of Model Checking. Lecture Notes in Computer Science*, vol 5000. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-69850-0_8

[2] Thomas Ball, Rupak Majumdar, Todd Millstein, and Sriram K. Rajamani. (2001). **Automatic predicate abstraction of C programs.** *PLDI '01. ACM*, New York, NY, USA, 203–213. https://doi.org/10.1145/378795.378846

[3] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. (2015). **How Amazon web services uses formal methods.** *Commun. ACM* 58, 4 (April 2015), 66–73. https://doi.org/10.1145/2699417

The complexity is here: Analysing a Landing Gear System [1] with Timed Automata [2]

Failure-free state-space



352 379 states
1 477 197 transitions

[1] Boniol, F., Wiels, V. (2014). **The Landing Gear System Case Study**. In: Boniol, F., Wiels, V., Ait Ameur, Y., Schewe, KD. (eds) ABZ 2014: The Landing Gear Case Study. ABZ 2014. Communications in Computer and Information Science, vol 433. Springer, Cham. https://doi.org/10.1007/978-3-319-07512-9_1

[2] Dhaussy, P., Teodorov, C. (2014). **Context-Aware Verification of a Landing Gear System**. In: Boniol, F., Wiels, V., Ait Ameur, Y., Schewe, KD. (eds) ABZ 2014: The Landing Gear Case Study. ABZ 2014. Communications in Computer and Information Science, vol 433. Springer, Cham. https://doi.org/10.1007/978-3-319-07512-9_4

Failure: Electro-valve blocked open

100X bigger state-space

35 223 747 states
164 661 264 transitions

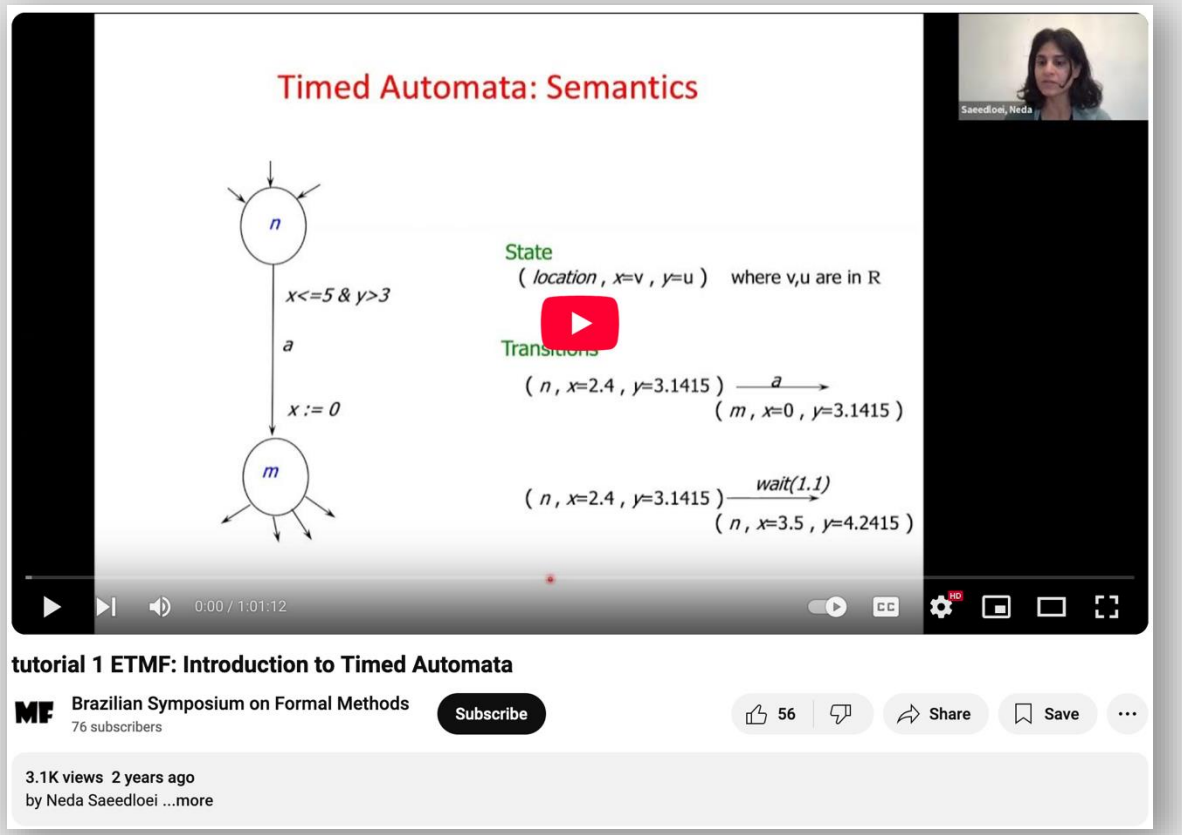
Modeling and Verification with Timed Automata

ETMF 2021:

Timed Automata

Prof. Neda Saeedloei (Towson University)

https://youtu.be/KJAZNXN3aiA



Example: Level Crossing

Goal: design a "Level Crossing Controller" that operates a barrier to stop road traffic when a train is passing.

To detect an incoming train, the crossing is equipped with a "Detection Zone," which is implemented using a track circuit. This sensor not only detects when a train enters but also when the zone is clear.

For both safety and efficiency, once a train is detected in the zone, it must reach the crossing within 50 seconds at most. Once the train is on the crossing, it must clear it within a maximum of 40 seconds. However, the train's speed is limited, so it must remain on the crossing for at least 20 seconds.

The barrier, located at the intersection of the road and the railway, is operated by an electric motor, which is controlled by an electronic circuit with two inputs: "open" and "close."

Upon receiving a command, the barrier takes between 10 and 20 seconds to either fully open or close. If a train reaches the crossing while the barrier is raising, the sequence can be interrupted at any moment.

Timed Automata : Syntax

$\mathcal{A}$= (Q, ∑, X, G, L, δ, I)

Q – a finite set of states

I ⊆Q – un ensemble d'état initiaux

∑ – an alphabet

X – a finite set of clocks

G – a set of clock constraints

$L : Q \to \mathcal{P}(G)$ – A labeling function that associates each state with a set of clock constraints.

$\delta : Q \times \Sigma \times \mathcal{P}(G) \times \mathcal{P}(X) \times Q$ – a transition relation

Timed Automata : Semantics

For a timed automaton $\mathcal{A}$, its semantics is defined through an infinite transition system $S(\mathcal{A})$ as follows:

$$Q_{S(\mathcal{A})} = \{\langle s, v\rangle \mid s \in Q_{\mathcal{A}}, v \in \mathbb{R}^n, v \vdash L(s)\}$$

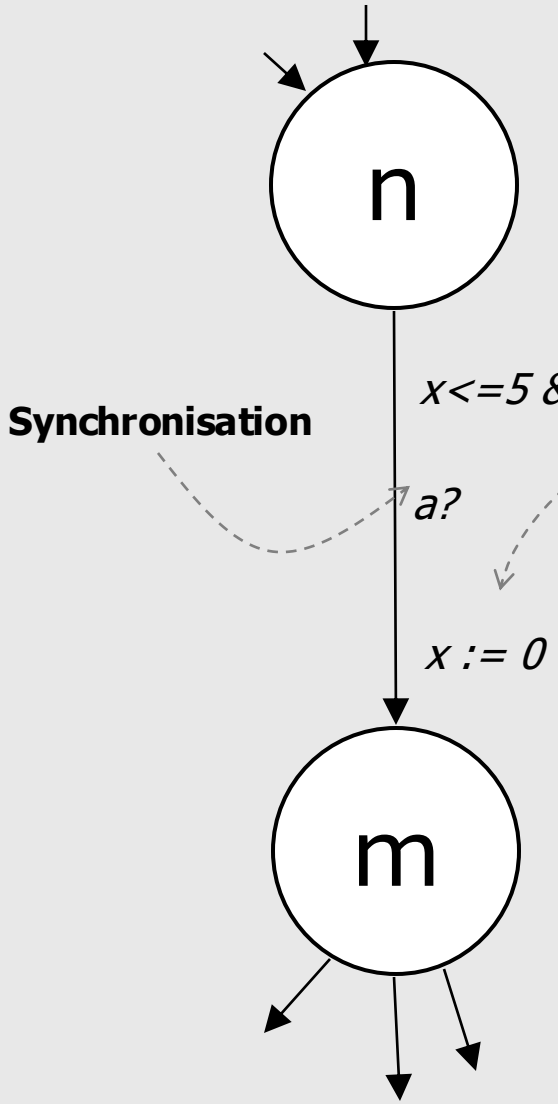
$$I_{S(\mathcal{A})} \subseteq Q_{S(\mathcal{A})} = \{\langle s, 0\rangle \mid s \in I_{\mathcal{A}}, 0 \vdash L(s)\}$$

Time transition

$$\delta_{S(\mathcal{A})}(\langle s, v\rangle) = \begin{cases} \langle s, v + d\rangle, & d \in [0, \infty) \quad \wedge v \vdash L(s) \quad \wedge v + d \vdash L(s) \\ \langle s', v'\rangle, & \delta(s, -, g, r) = s' \quad \wedge v \vdash g \quad \wedge v' = v[r/0] \end{cases}$$

Location switch transition

Timed Automata: From Syntax to Semantics



Clocks: x, y

Guard

Reset
The only action available on clocks`

Semantics

Configurations

$\langle$ *location*, $v = \langle x{=}r_1, y{=}r_2 \rangle\rangle$ where r_1, r_2 are in $\mathbb{R}$

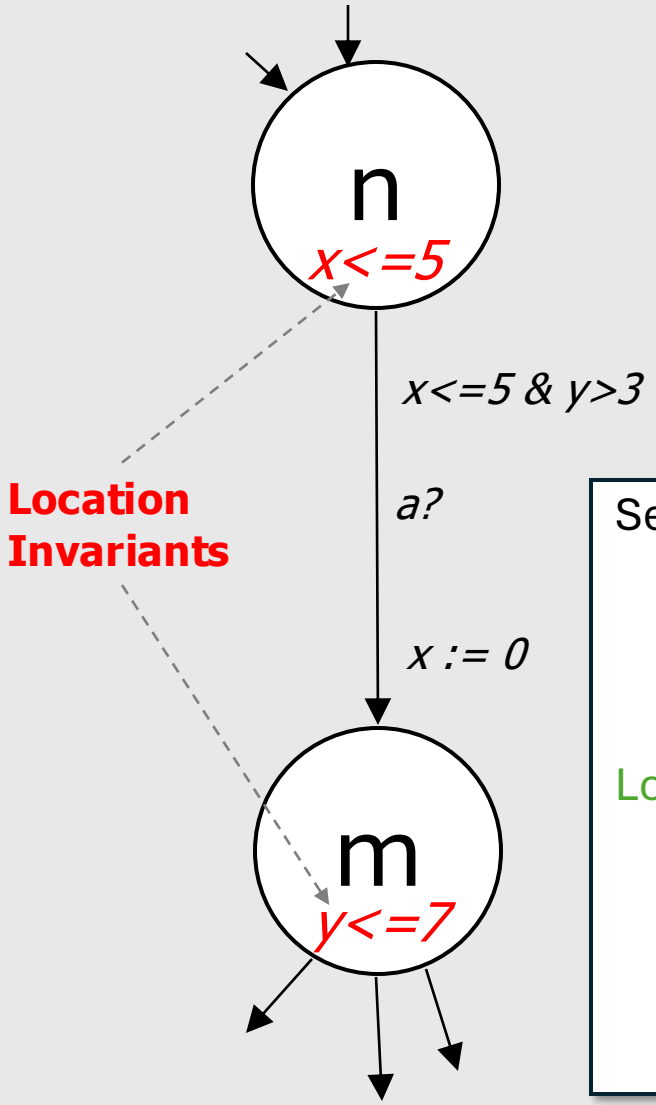
Transitions

Location switch transition $\langle n, \langle x{=}2.4 , y{=}3.1415\rangle\rangle \xrightarrow{a} \langle \mathbf{m}, \langle x{=}0 , y{=}3.1415\rangle\rangle$

Time transition $\langle n, \langle x{=}2.4 , y{=}3.1415\rangle\rangle \xrightarrow{\mathbf{wait(1.1)}} \langle n, \langle \mathbf{x{=}3.5 , y{=}4.2415}\rangle\rangle$

Time transition $\langle n, \langle x{=}2.4 , y{=}3.1415\rangle\rangle \xrightarrow{\mathbf{wait(3.2)}} \langle n, \langle \mathbf{x{=}5.6 , y{=}6.3415}\rangle\rangle$

Timed Automata: From Syntax to Semantics



Clocks: *x, y*

Invariants force the state to change!
Cannot stay in n infinitely

Semantics

Configurations

⟨ *location*, v = ⟨*x*=r_1, *y*=r_2⟩⟩ where r_1,r_2 are in $\mathbb{R}$

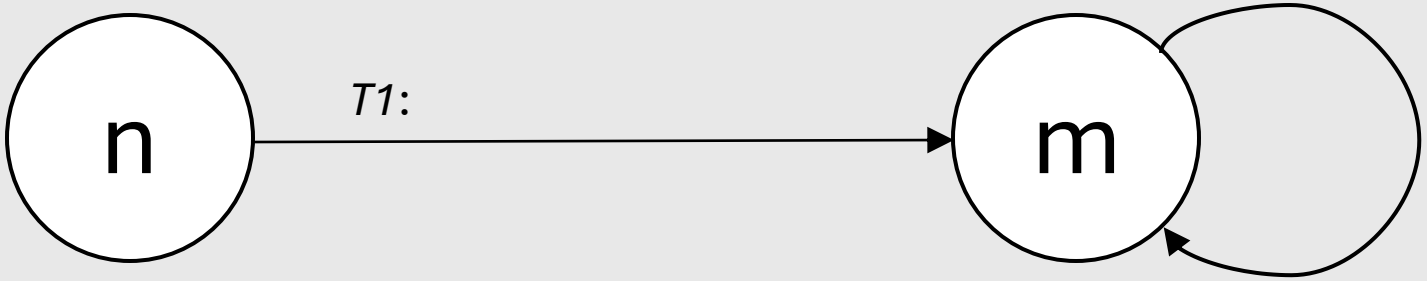
Transitions

Location switch transition ⟨*n*, ⟨*x*=2.4 , *y*=3.1415⟩⟩ —**a**→ ⟨***m***, ⟨*x*=0 , *y*=3.1415⟩⟩

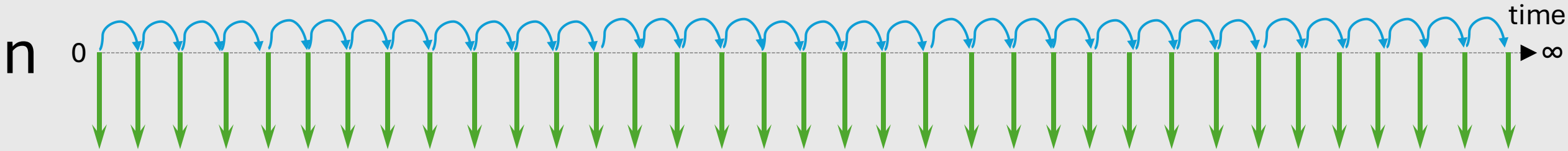
Time transition ⟨*n*, ⟨*x*=2.4 , *y*=3.1415⟩⟩ —**wait(1.1)**→ ⟨*n*, ⟨***x*=3.5 , *y*=4.2415**⟩⟩

Time transition ⟨*n*, ⟨*x*=2.4 , *y*=3.1415⟩⟩ —~~**wait(3.2)**~~→ ⟨*n*, ⟨***x*=5.6 , *y*=6.3415**⟩⟩

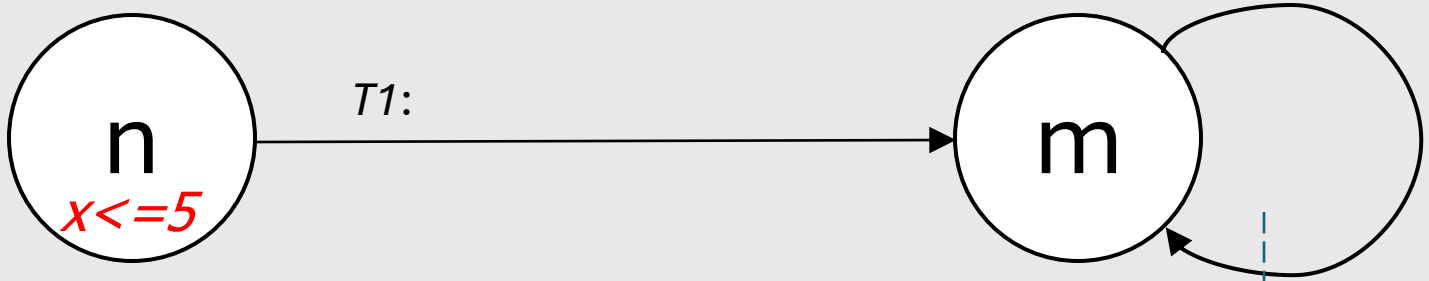
Deriving The Most Useful Pattern in Timed Automata



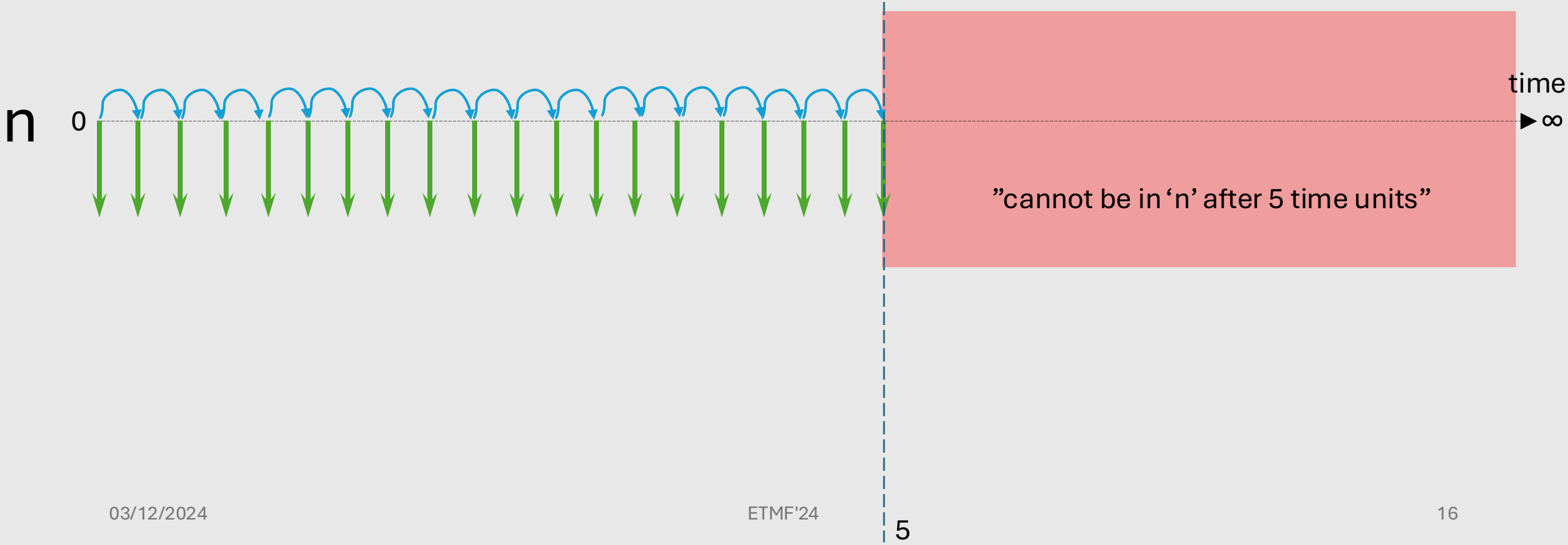
A<>Process.m -- Fails



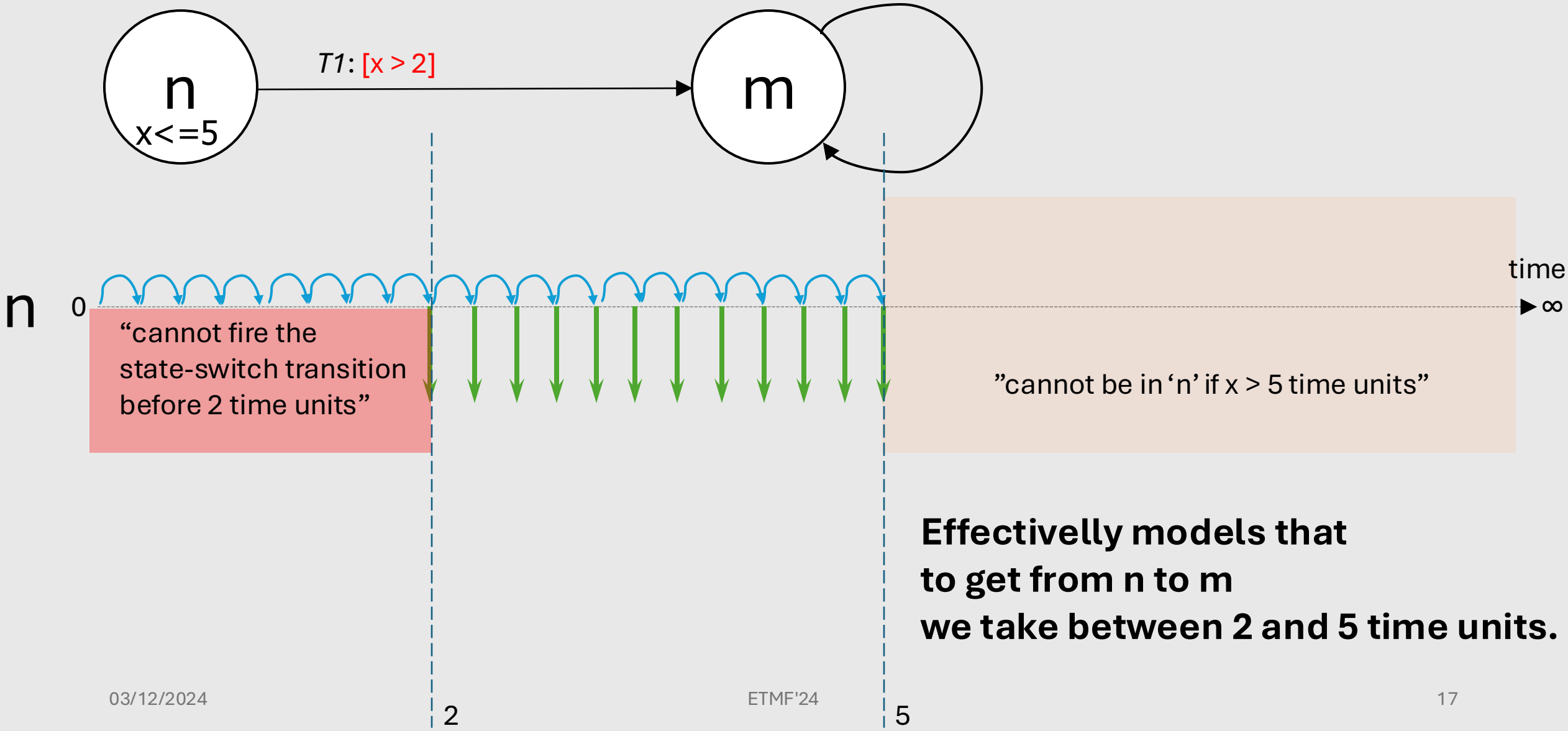
Deriving The Most Useful Pattern in Timed Automata



A<>Process.m -- Pass

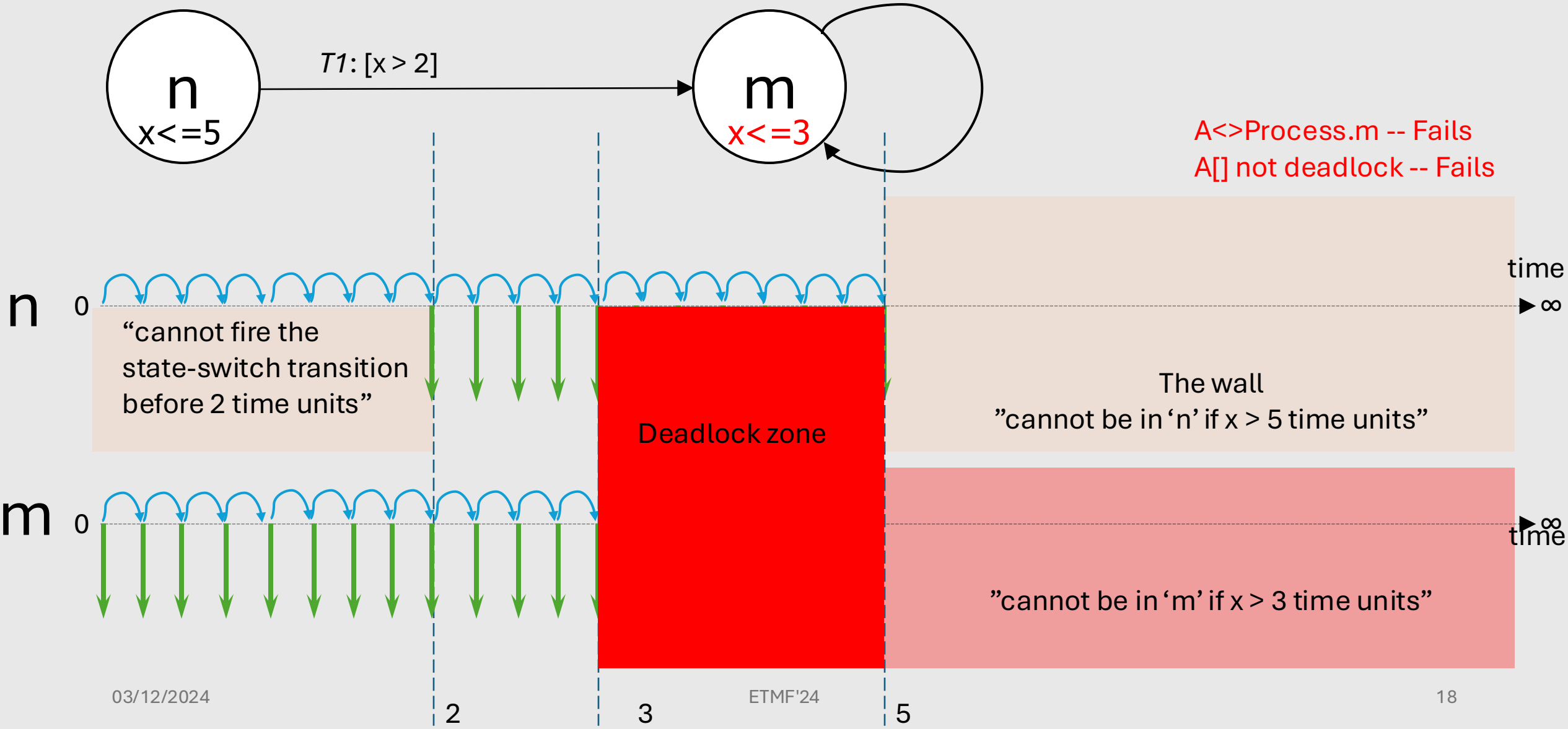


Deriving The Most Useful Pattern in Timed Automata

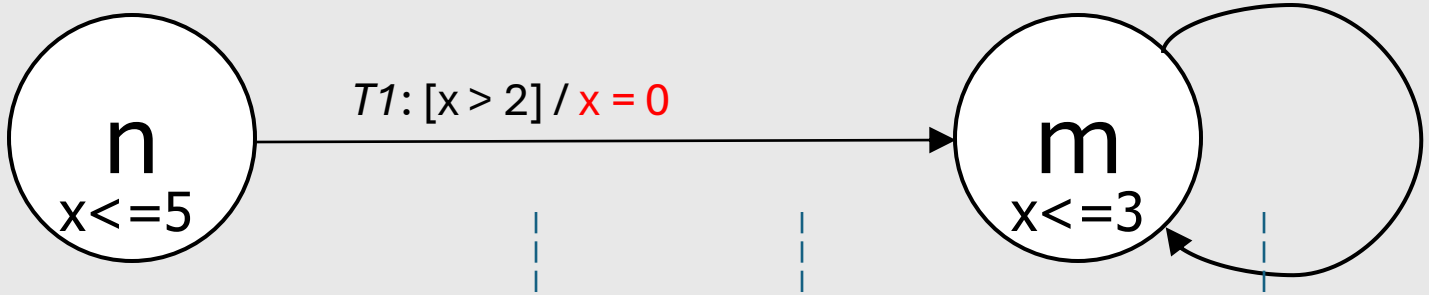


Effectivelly models that to get from n to m we take between 2 and 5 time units.

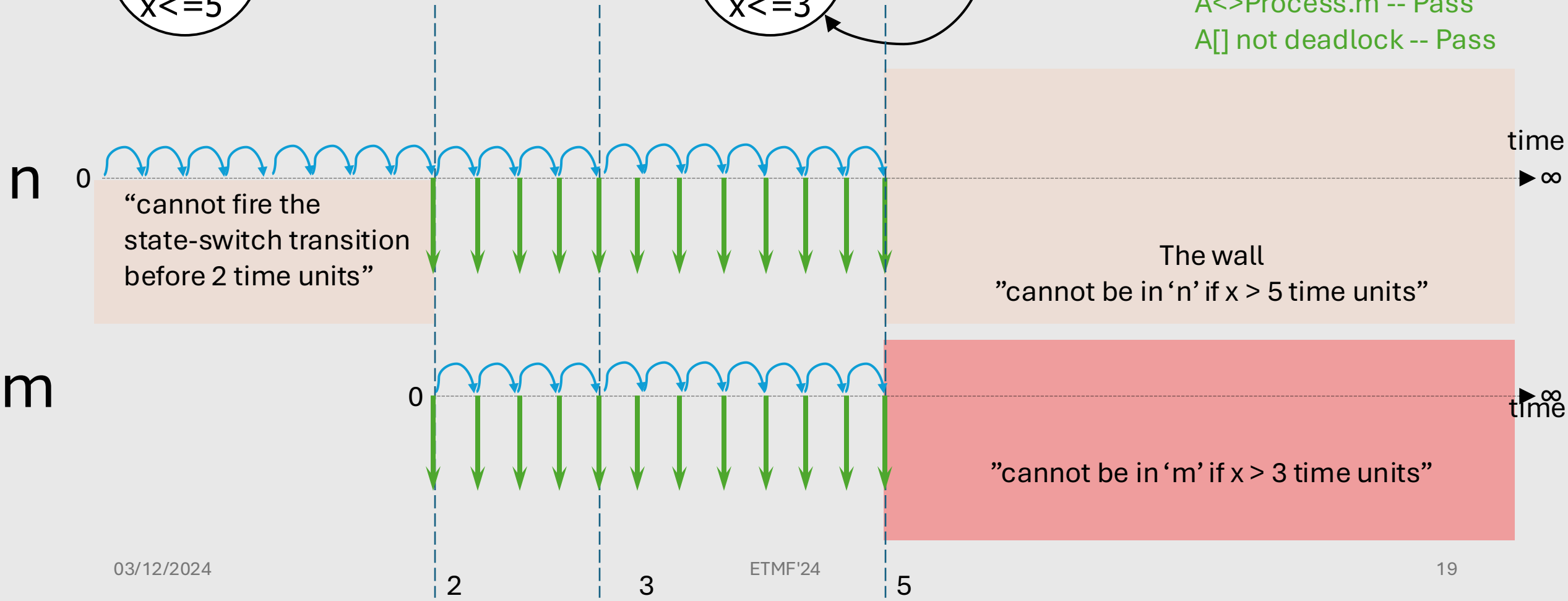
Deriving The Most Useful Pattern in Timed Automata



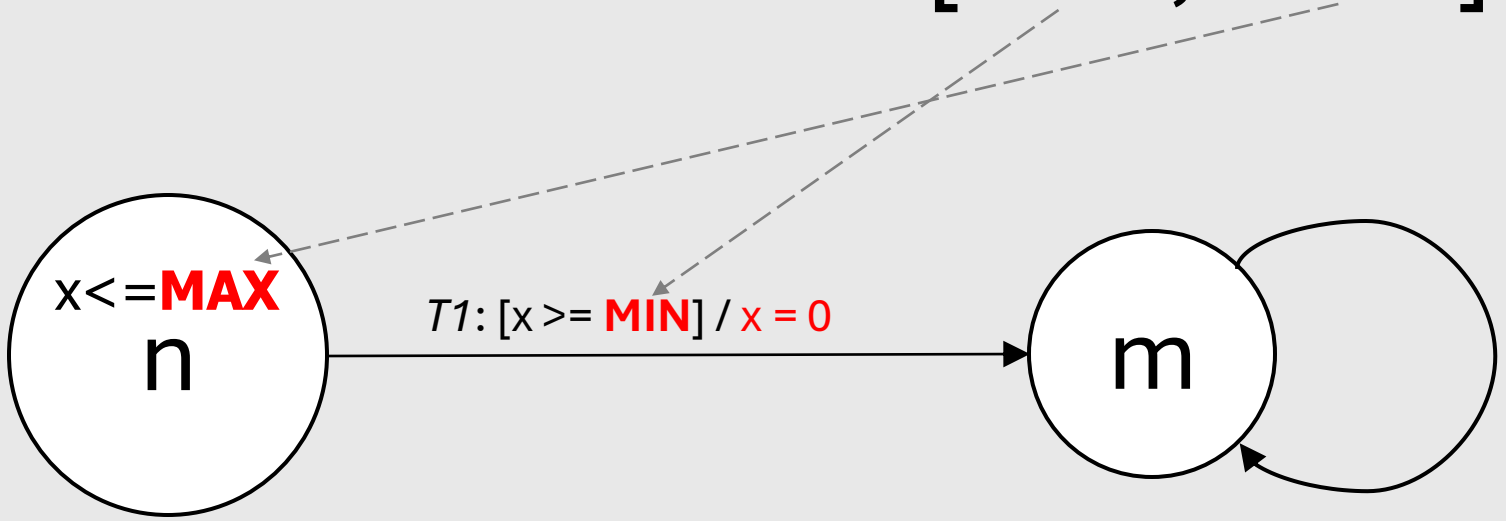
Deriving The Most Useful Pattern in Timed Automata



A<>Process.m -- Pass
A[] not deadlock -- Pass

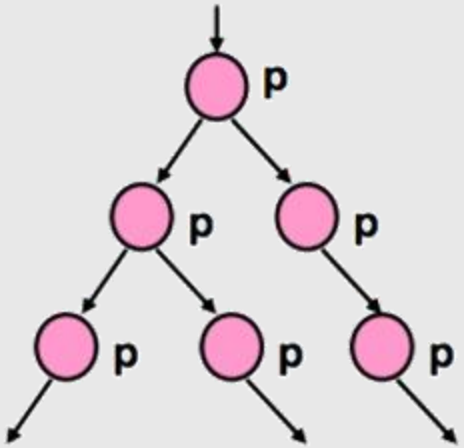


The transition T1 executes in [**MIN**, **MAX**]

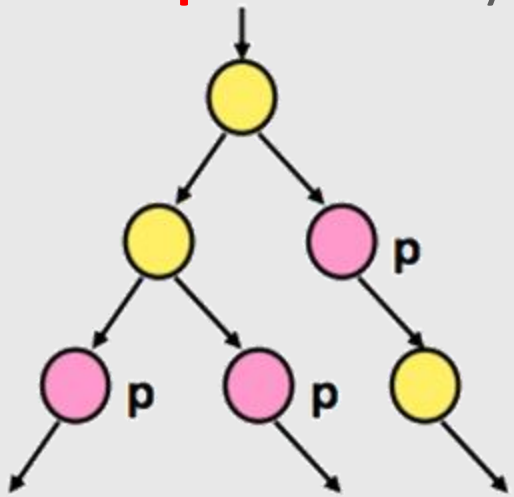


1. What is the earliest time I can leave?
   MIN goes to transition guard.
2. What's the maximum time I'm allowed to stay?
   MAX goes to state invariant
3. Always reset the clocks by default.
   Choosing not to reset a clock should be a deliberate and well-justified decision.

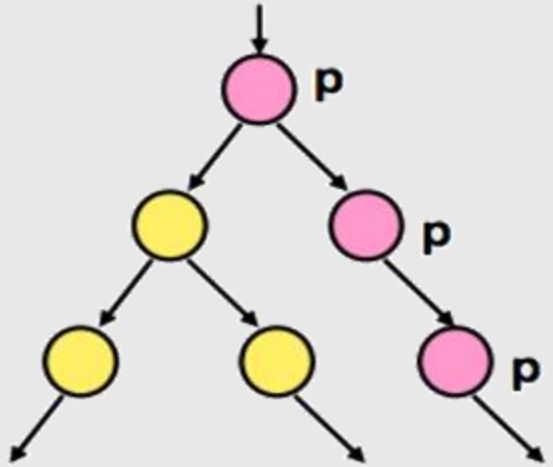
A[] p: Invariantly



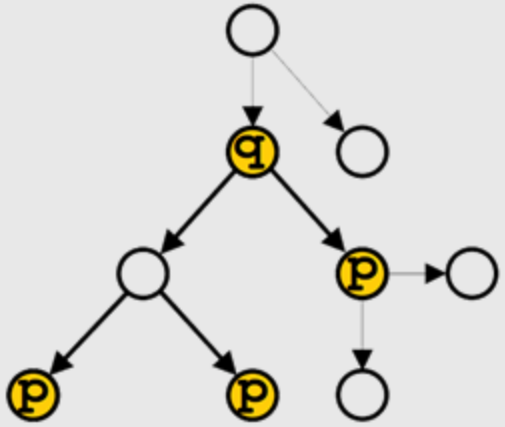
A<> p: Eventually



E[] p: Potentially Always



p --> q: p Leads To q



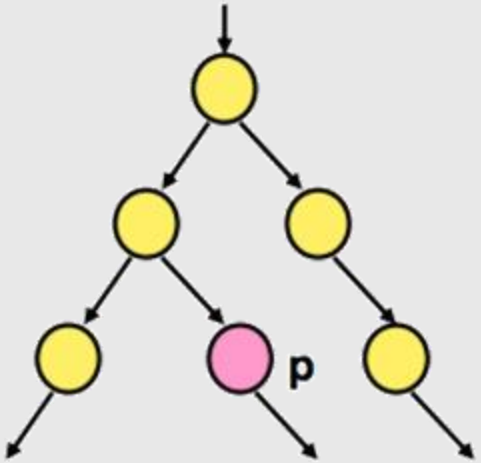
UPPAAL Specifications

subset of TCTL

Propositional Logic + **Temporal Layer**

- *Alice.W*
- *p1.v < 4*
- *p1.cs* **and** *p2.cs*
- **not** *deadlock*

E<> p: Possibly



Let’s do some UPPAAL now

clone: https://github.com/teodorov/ETMF24_LevelCrossing

start Uppaal

play

Example: Level Crossing

Goal: design a "Level Crossing Controller" that operates a barrier to stop road traffic when a train is passing.

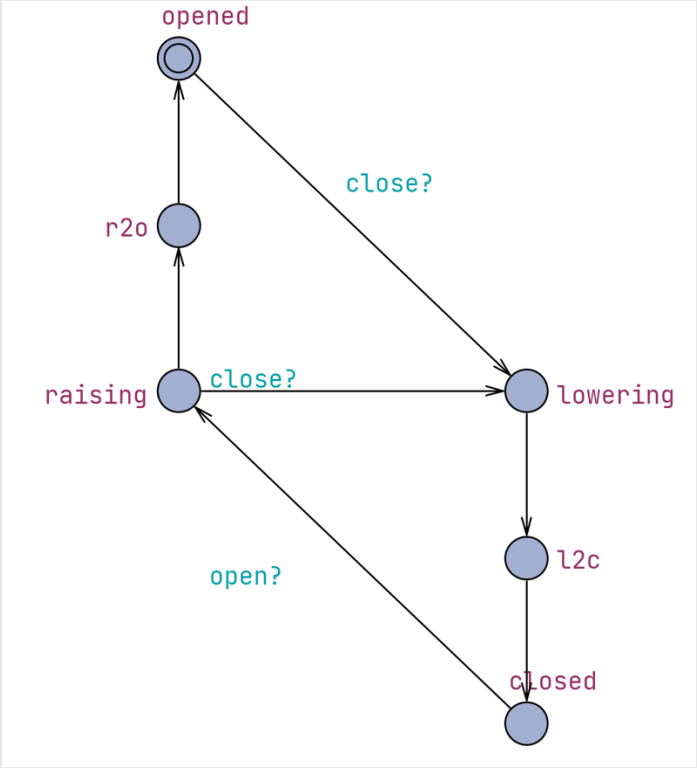
To detect an incoming train, the crossing is equipped with a "Detection Zone," which is implemented using a track circuit. This sensor not only detects when a train enters but also when the zone is clear.

For both safety and efficiency, once a train is detected in the zone, it must reach the crossing within 50 seconds at most. Once the train is on the crossing, it must clear it within a maximum of 40 seconds. However, the train's speed is limited, so it must remain on the crossing for at least 20 seconds.

The barrier, located at the intersection of the road and the railway, is operated by an electric motor, which is controlled by an electronic circuit with two inputs: "open" and "close."

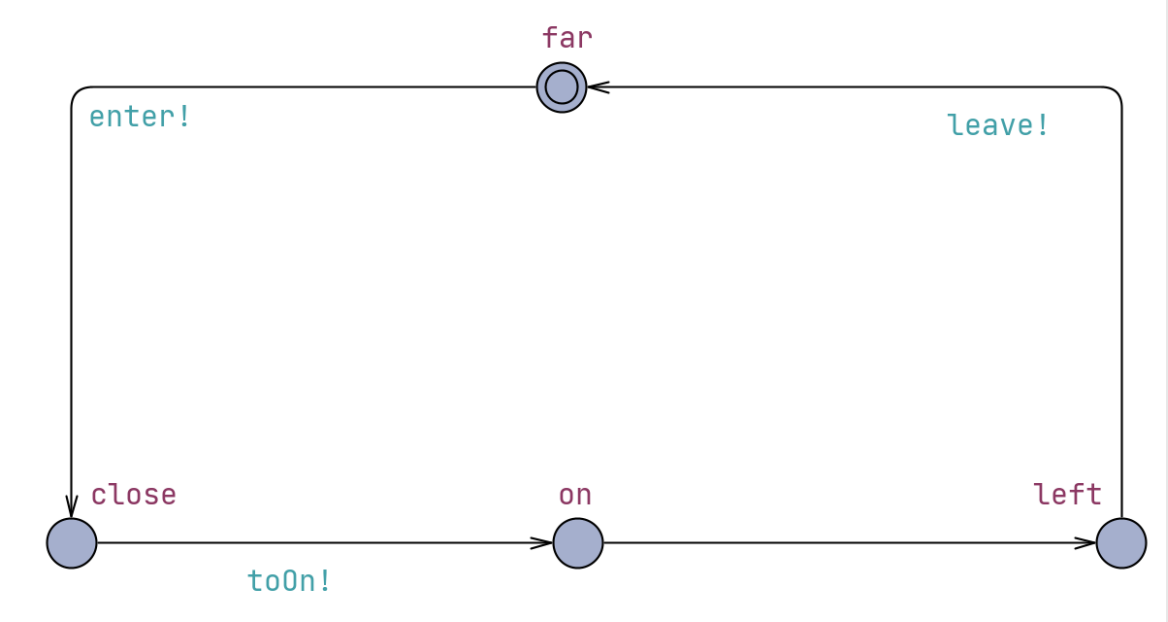
Upon receiving a command, the barrier takes between 10 and 20 seconds to either fully open or close. If a train reaches the crossing while the barrier is raising, the sequence can be interrupted at any moment.

Barrier: Properties



1. A[] not deadlock
2. E<> barrier.closed
3. E<> barrier.opened
4. barrier.lowering --> barrier.closed
5. A[] barrier.lowering imply barrier.time >= 0 && barrier.time <= 20
6. A[] barrier.l2c imply barrier.time >= 10 && barrier.time <= 20
7. barrier.raising --> barrier.opened || barrier.lowering
8. A[] barrier.r2o imply barrier.time >= 10 && barrier.time <= 20

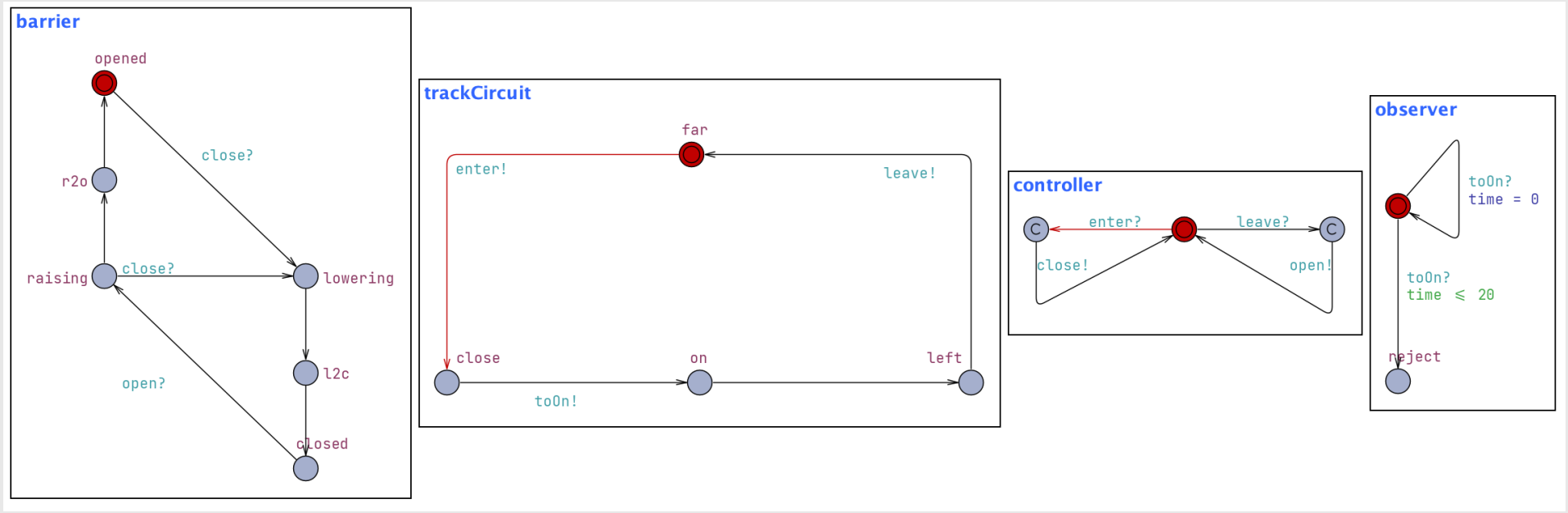
TrackCircuit: Properties



1. A[] not deadlock
2. E<> trackCircuit.far
3. E<> trackCircuit.close
4. E<> trackCircuit.on
5. E<> trackCircuit.left
6. trackCircuit.close --> trackCircuit.on
7. trackCircuit.on --> trackCircuit.left
8. trackCircuit.left --> trackCircuit.far
9. A[] trackCircuit.close imply trackCircuit.time >= 0 && trackCircuit.time <= 50
10. A[] trackCircuit.left imply trackCircuit.time == 0
11. A[] trackCircuit.on imply trackCircuit.time >= 0 && trackCircuit.time <= 40
12. A[] not observer.reject

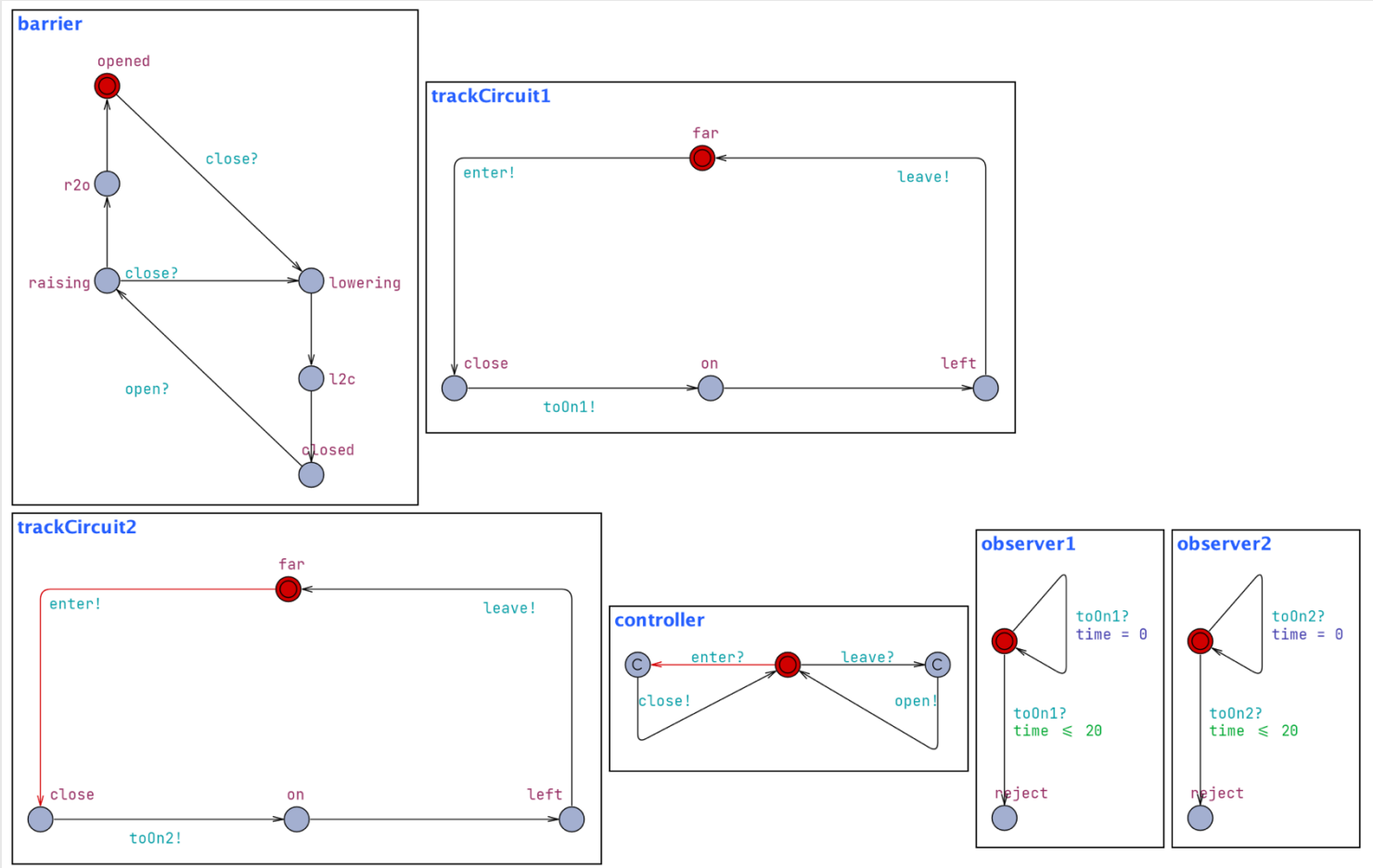
Full System: Properties

1. A[] not deadlock
2. A[] barrier.opened imply (not trackCircuit.on)
3. barrier.closed --> barrier.opened



To go further

1. Consider 2 track circuits
   - What needs to change
   - The 3 system properties have to pass
2. What if the barrier lowering and raising time are asymetric
3. How to generalize to n trains?



2. A[] barrier.opened imply ((not trackCircuit.on) && (not trackCircuit1.on))