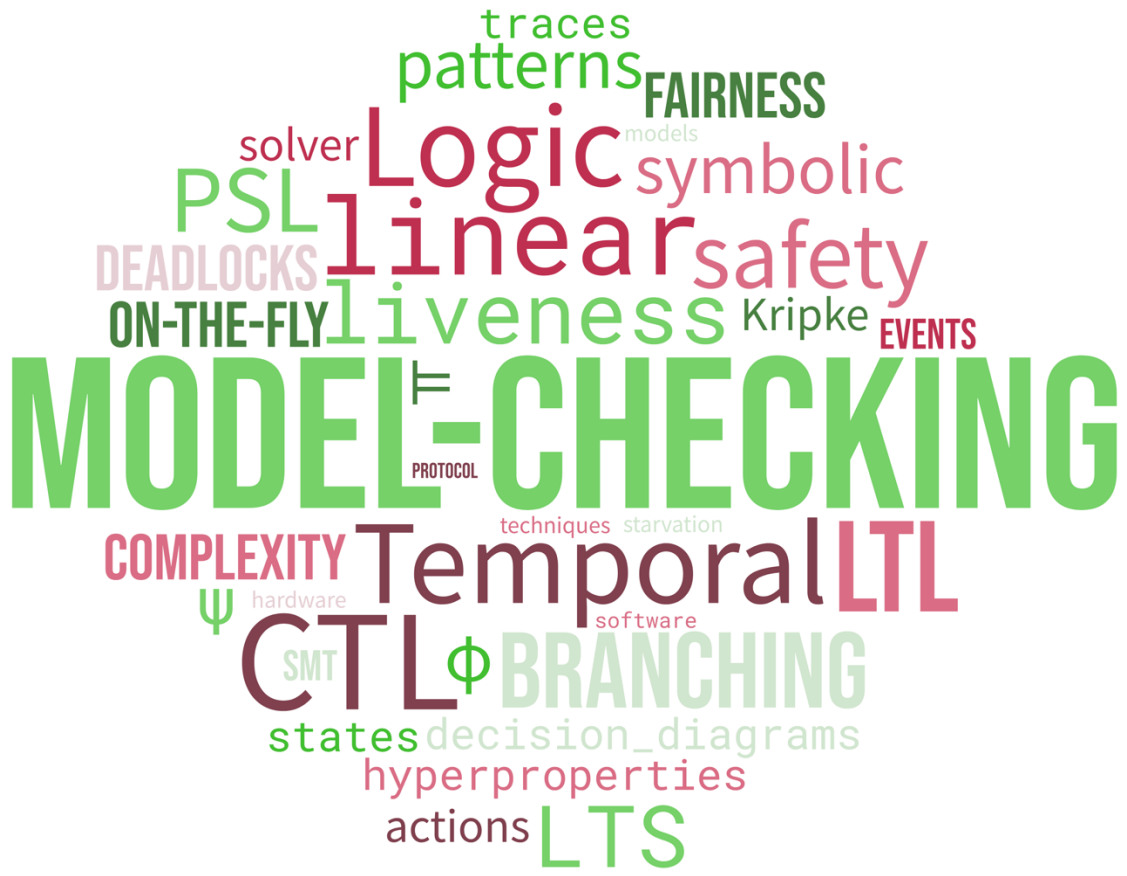
Model-Checking *et le temps*

Formation Systèmes à Evénements Discrets

2ème édition
Mars 2025

Model-Checking basics



''Model checking is the method by which a desired **behavioral property** of a reactive system is verified over a given system (the model) through exhaustive enumeration (explicit or implicit) of **all the states reachable by the system** and the behaviors that traverse through them.''

Amir Pnueli

$$M \models \phi$$

👉 *Old adage*: testing (and simulation) can find bugs, but cannot prove their absence. *Today*: (software M.-C.) results from Test-Comp show that the best test generators include a combination of dynamic tech. and formal methods.

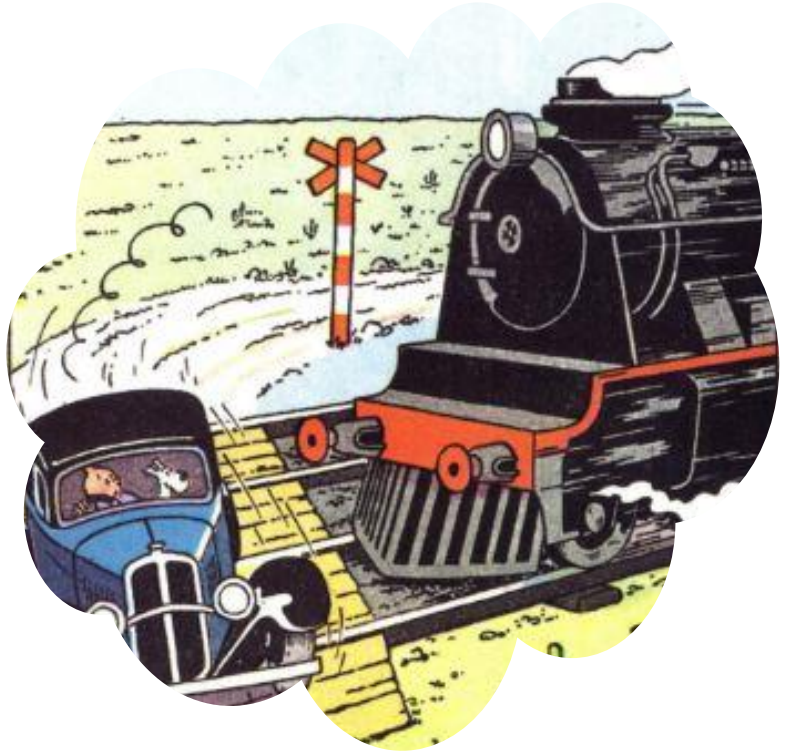
D. Beyer, T. Lemberger (2025), *Six years later: testing vs. model checking*. STTT 26.

1) Simple introduction: models, specifications, verification … and the state explosion problem

2) LTL model-checking and their timed extensions

3) CTL, TCTL and hands-on with Romeo and Uppaal

4) Patterns, observers and other interesting formulas



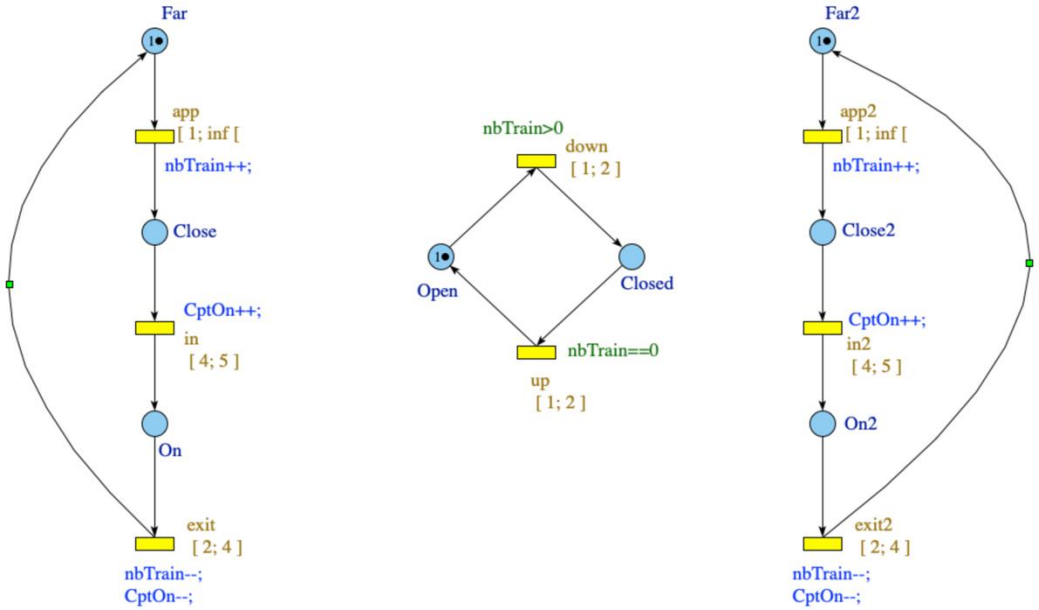
© Hergé-Moulinsart 2018

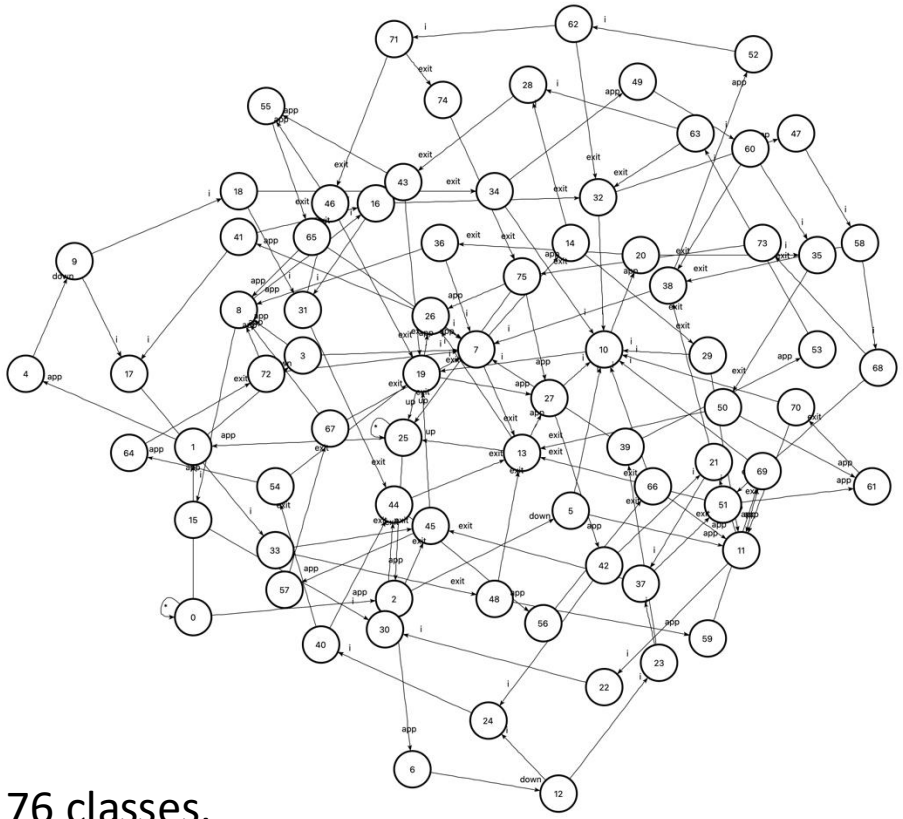
F. Sarikoc(2025), *Model Checking and Verification of a Rail-Side Protection System*. SCP (in press, Feb 28)
(uses a very similar model, exp. with Tina and Tappaal)

Model-Checking

an automata-based approach

2 trains + 1 barrier





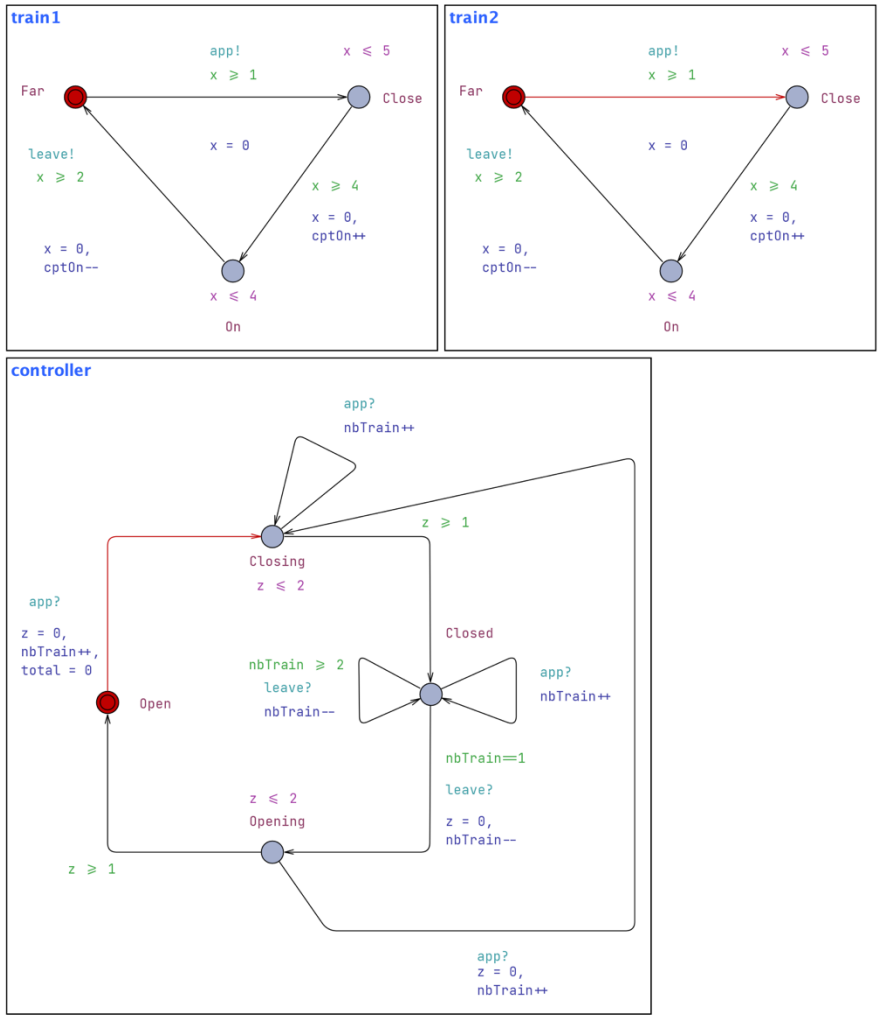
76 classes,

135 transitions

only 13 different markings (but 46 domains)

- *3 trains:* 2 812 classes, 35 markings, 0.020s
- *4 trains*: 145 410 classes, 97 markings, 0.805s
- *5 trains*: 10 342 579 classes, 275 markings, 104.918s
- *6 trains*: *killed after 6h of computation (28Gb of RAM used)*

2 trains + 1 barrier



No possibility to “draw” the state space

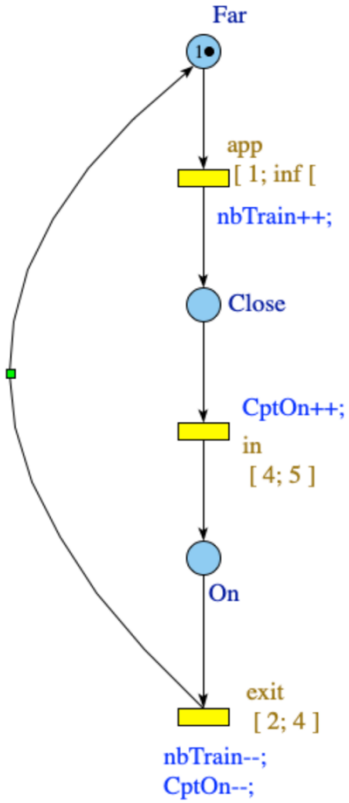
Use `verifyta -u` to get an idea of the state space size

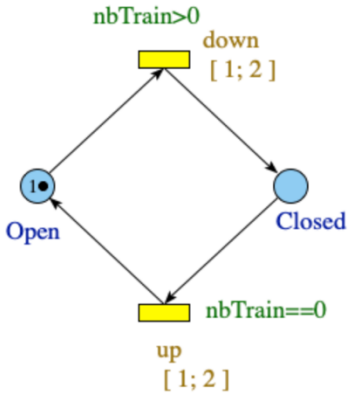
# trains	states (explored)	states (stored)	CPU time
2	43	39	0,001s
3	472	367	0,004s
4	6 385	4 457	0,11s
5	103 776	70 901	9,8s
6	2 073 097	1 428 985	1 984s
7	*killed after 6h of computation (< 2Gb of RAM used)*		

3 trains: 2 812 classes, 35 markings, 0.020s
4 trains: 145 410 classes, 97 markings, 0.805s
5 trains: 10 342 579 classes, 275 markings, 104.918s
6 trains: *killed after 6h of computation (28Gb of RAM used)*
5 trains: ..., 275 markings, 0.154s
6 trains: ..., 793 markings, 4.704s
7 trains: ..., 2315 markings, 314.424s

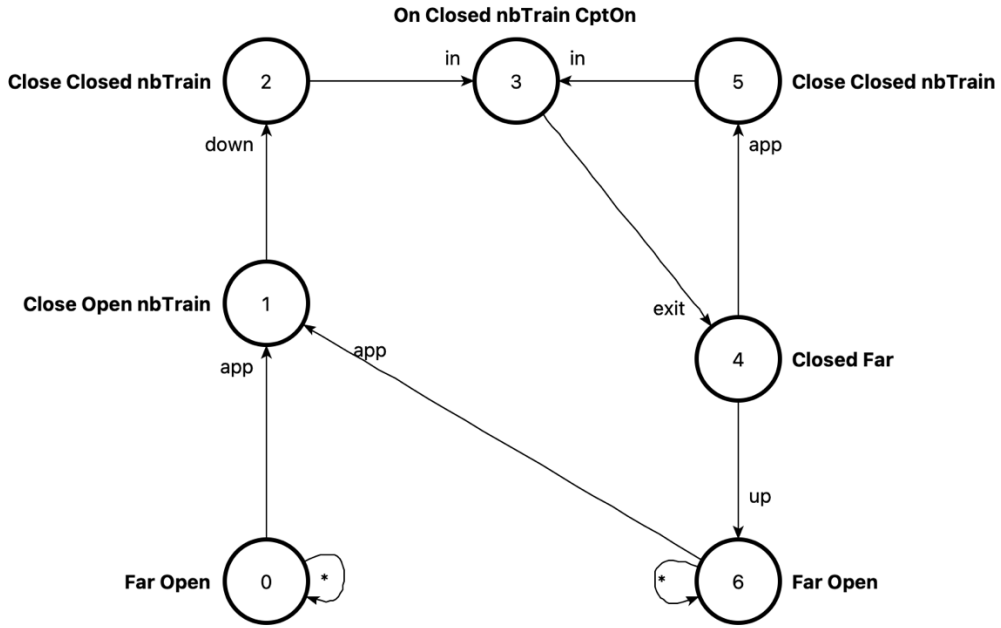
option -M

1 train + 1 barrier





State Class Graph (SCG) ≠ Marking Graph



👉 * ≡ time divergence

Kripke Structure

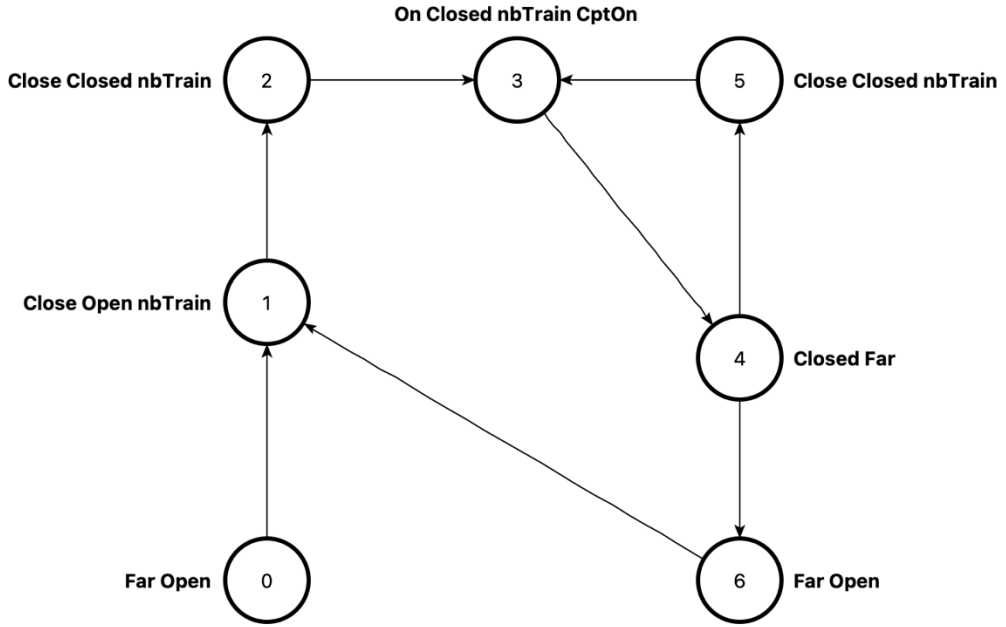
- A graph (initial state 0)
- States ↦ collection of *atomic propositions* (2^{AP})

  Close, Far, nbTrain, …
- Transition relation is *total*

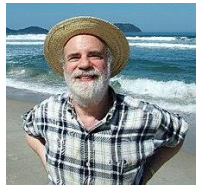
  deadlocks, time divergence, …
- Traces or runs

  sequence of states $s_1 \rightarrow s_2 \rightarrow \ldots$

  we consider maximal sequences Σ^{ω}



👉 in practice only a subset of AP and events are observable, controllable + some events are indistinguishable, e.g. On = (train1.On + train2.On) ; *use of labels*



Saul Kripke (1940—2022)

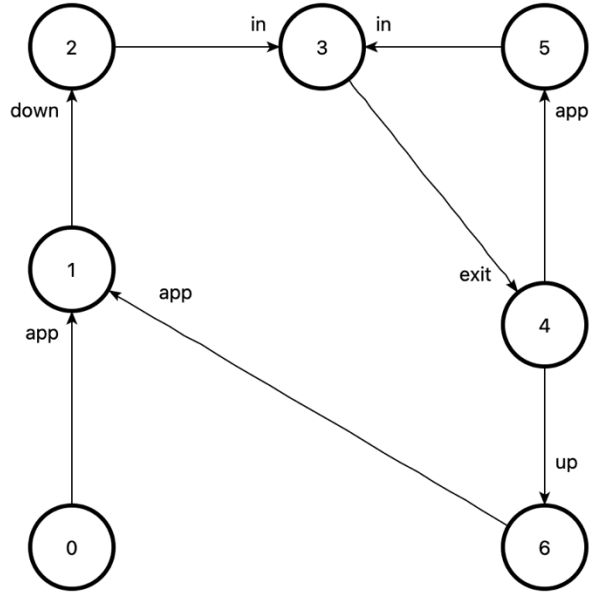
Labeled Transition System (LTS)

- A graph (initial state 0)
- actions ≡ events, transitions

  app, down, up, …
- Traces: $s_0 \xrightarrow{\alpha} s_1 \xrightarrow{\tau} s_2 \xrightarrow{\beta} \ldots$

  we consider maximal sequences Σ^ω
- ≈ DFA

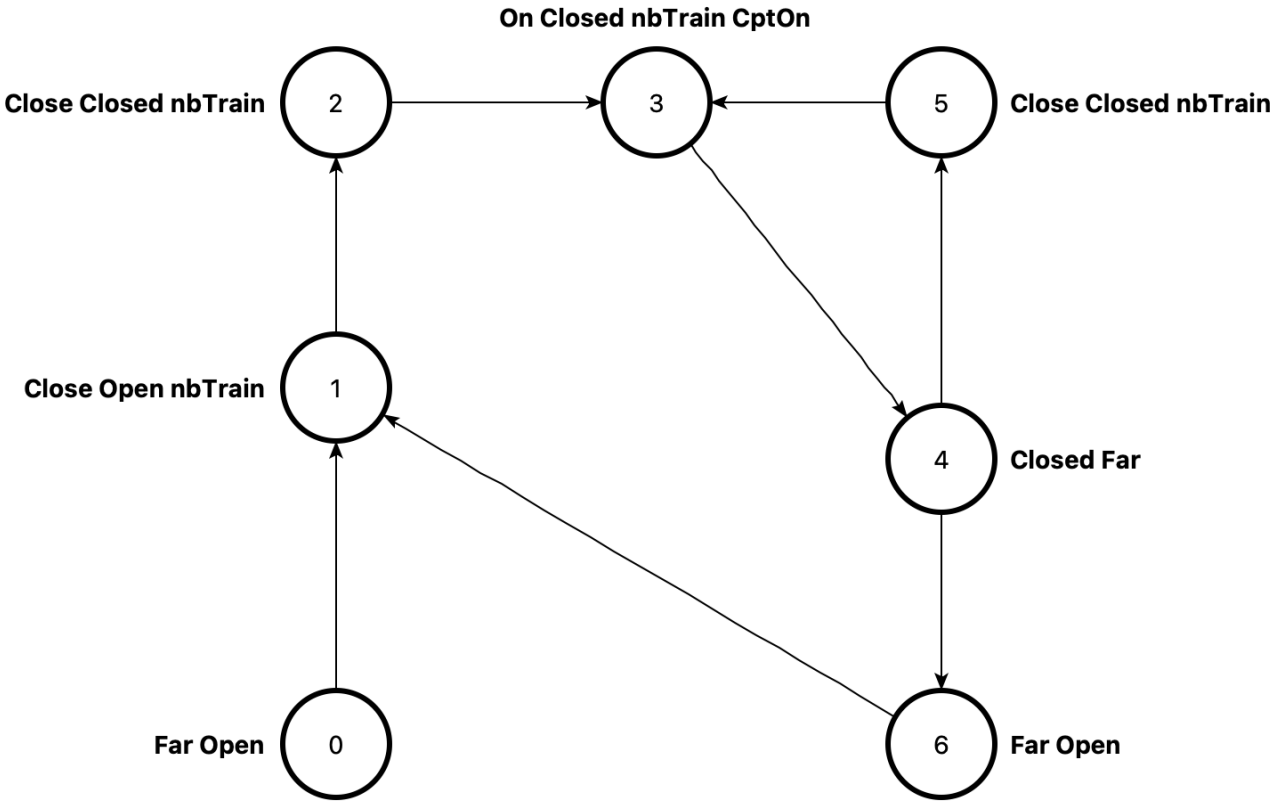
  except if we use labels or τ-actions

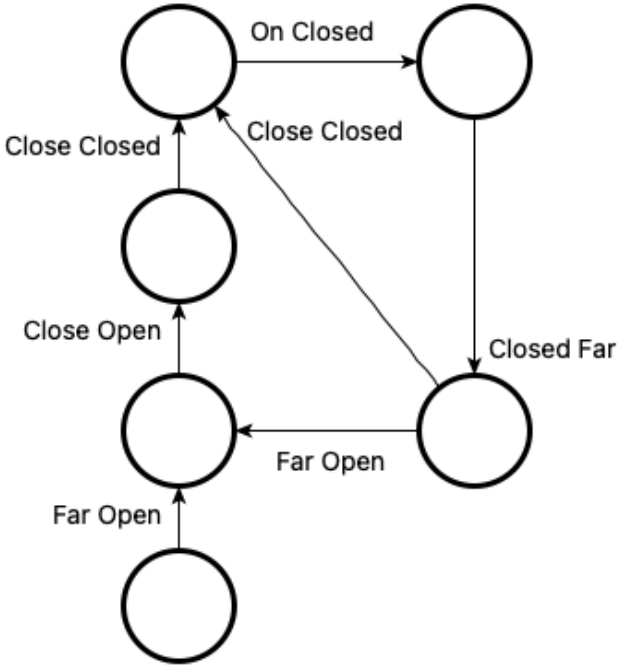


👉 used e.g. with Structural Operational Semantics (SOS); process algebra, …

Kripke ⇄ LTS

We can always encode a KS into a LTS





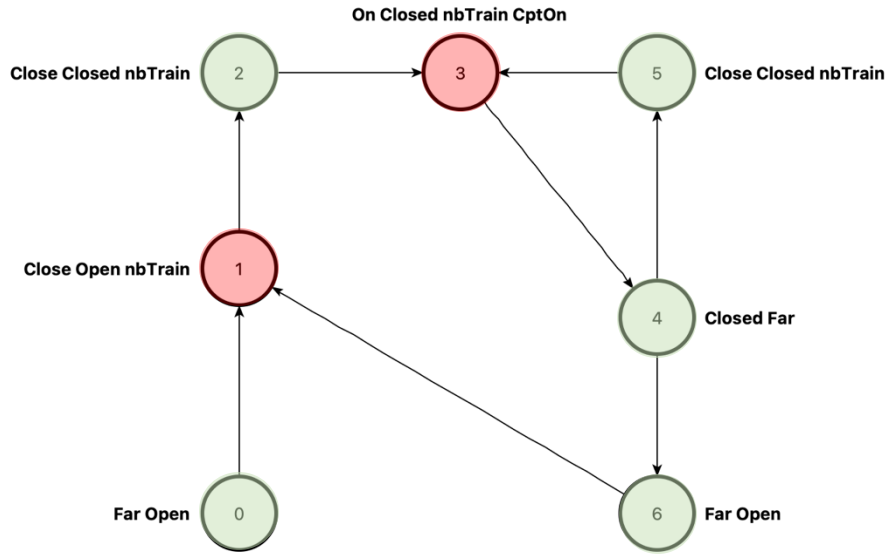
(atomic) state formulas: σ

Boolean combination of Atomic Properties, evaluated on each state.

$$\neg On \wedge (Closed \vee Far)$$

$$!On \ \& \ Closed$$

$$(On.1 + On.2 < 3)$$



👉 GMEC expressions ≡ linear constraints on marking

Safety / invariants

(Always, Globally) **AG** σ

Ex.: whenever the train is inside the gate then the gate should be closed.

$$\mathrm{AG}\left((On.1 + On.2 \geq 1) \implies Closed\right)$$

$$\square\ (\neg On \vee Closed)$$

$$\mathrm{AG}\ \neg deadlock$$

- 👉 can be easily checked *on-the-fly*
- 👉 denoted A[] in Uppaal

Dual: occurrence

(Exists, Finally) **EF** σ

Ex.: the barrier can be closed but with no train in sight !

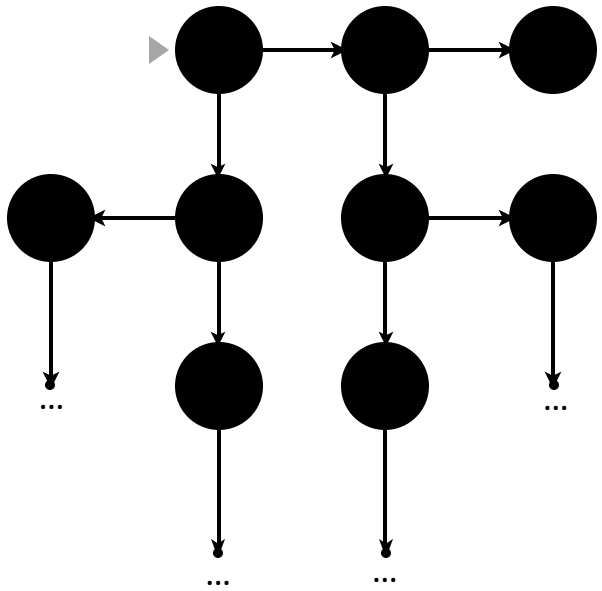
$$\mathrm{EF}\ \ (Closed \wedge Far)$$

$$\mathrm{EF}\ \sigma \ \equiv\ \neg(\mathrm{AG}\ \neg\sigma)$$

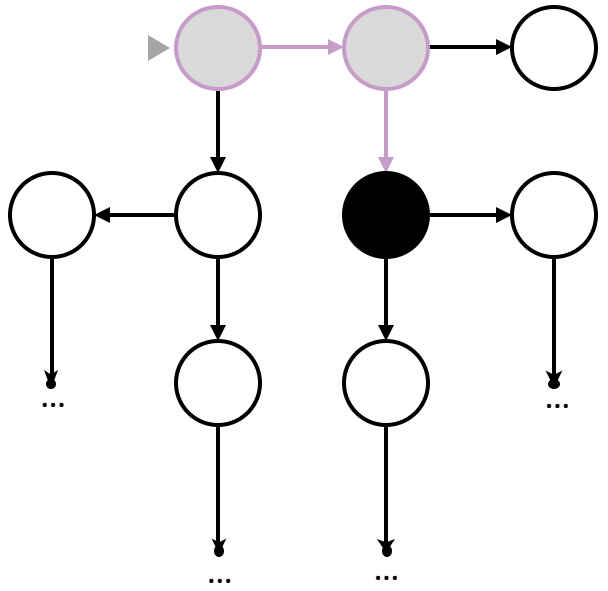
👉 De Morgan duality (like □ and ◇)

Basic operators

(Always, Globally) AG



(Exists, Finally) EF

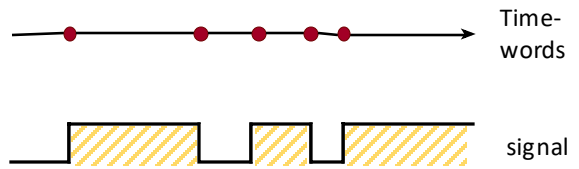


👉 when EF true then we can output a *scenario* (a witness). Dually, we have a *counter-example* if AG is false.

👉 if AG true the only possible witness is the whole state graph !

What about time

- In Uppaal, clocks can be used in atomic properties
  Example: train1.Far or train1.x <= 5
- Deadlocks, timelocks
- Time divergence: states where we can wait indefinitely
- Two different kinds of timed semantics: *timed-words* (pointwise) $\theta_0\ s_1\ \theta_1\ s_2\ \theta_2$ … (discrete sequence of states) versus *signal* $f\colon \mathbb{R}^+ \to S$ (states are continuous). Does it make a difference ?
- What about Zeno-traces (∞ run with finite duration) ?



Temporal Logic

A. N. Prior (1957). Time and Modality. Oxford Univ. Press.

A non-standard (modal) logic to reason and represent our knowledge about time: "*what is true at one time is in many cases false at another time, and vice versa*."

Amir Pnueli (1941—2009)

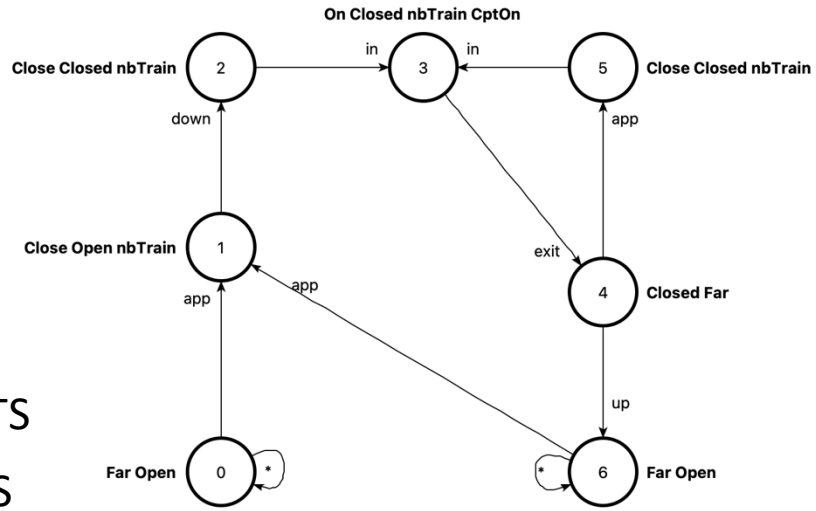
Turing award (1996) for seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification.

Temporal properties

When the verdict depends on the order of events in an execution

Examples:

- There is always a train *exit* before a barrier *up*
- Barrier is *Closed* until train is *Far*
- Barrier *Closed* is always (eventually) *Open*
- Train *On* is followed by *Far* (next)
- There is always barrier *up* between train *Close* and train *Far*
- System is *reinitializable*



(✅) LTS

(❌) KS

(❌)

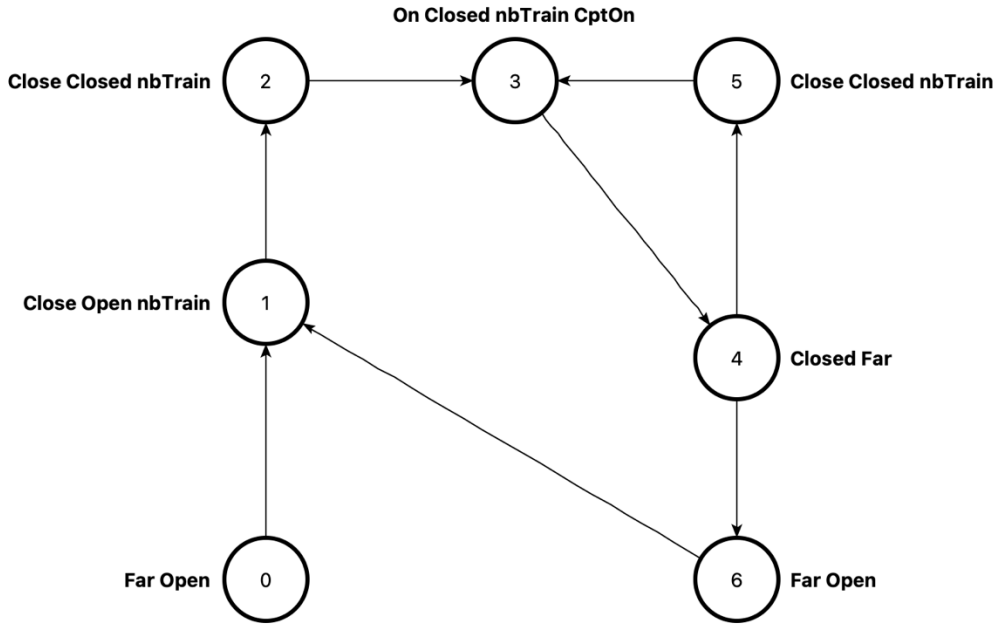
(✅)

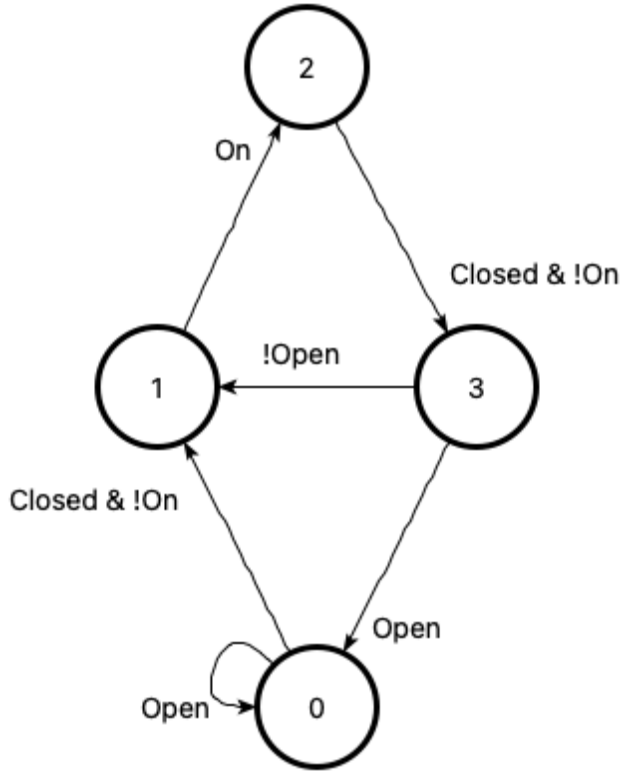
(❌) states + events !?

(✅) not a linear time property

👉 we will only consider properties about states from now on

Checking properties on 1 train + 1 barrier





- Always barrier.Closed before train.On

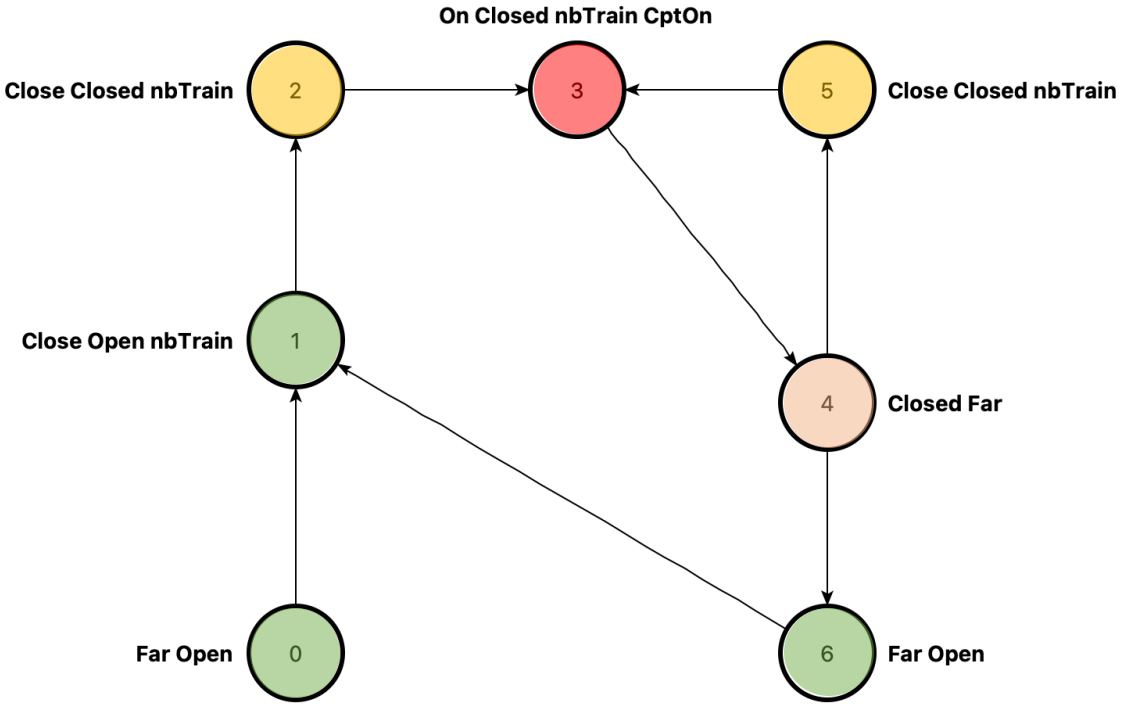
  Open . Closed . On . Closed . Open . … $\in \Sigma^{\omega}$
- We can infinitely avoid barrier.Open
- We can always eventually have barrier.Open

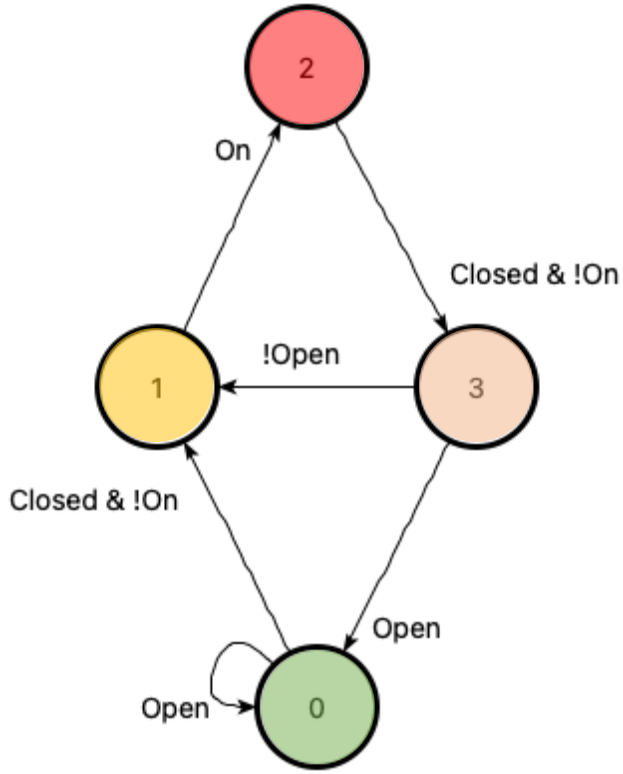
👉 checking property ≡ trace inclusion

👉 idea: use an automaton / language as a specification

Always barrier.Closed before train.On

// between checking the property and product between automata

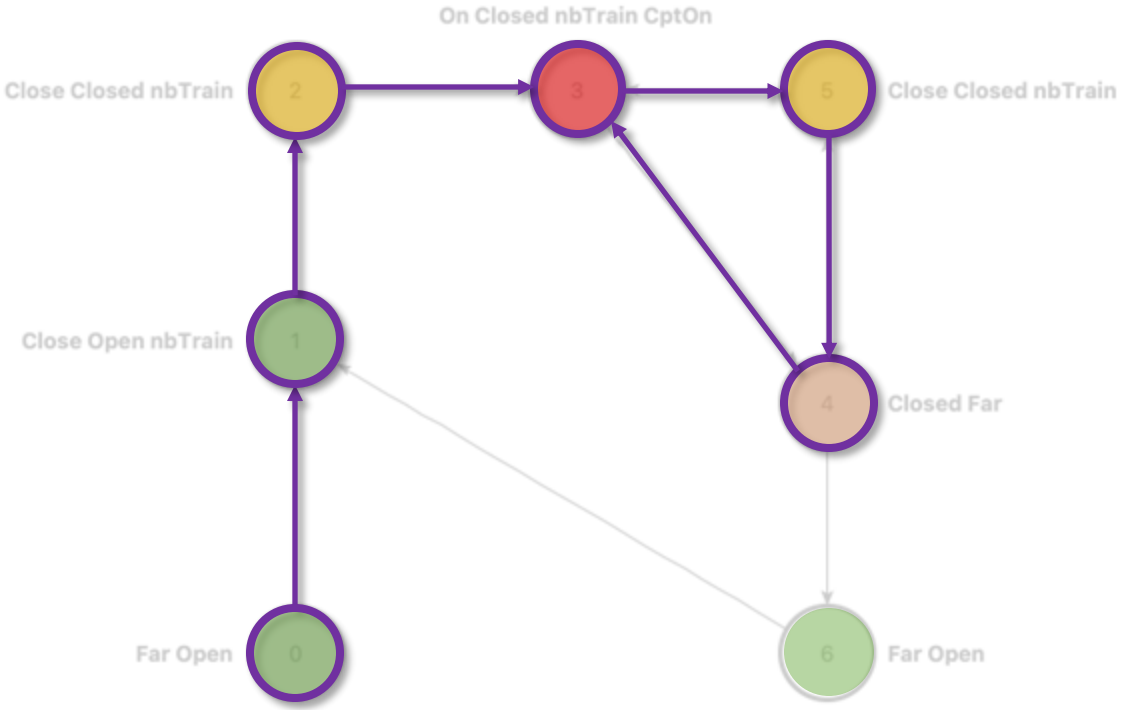


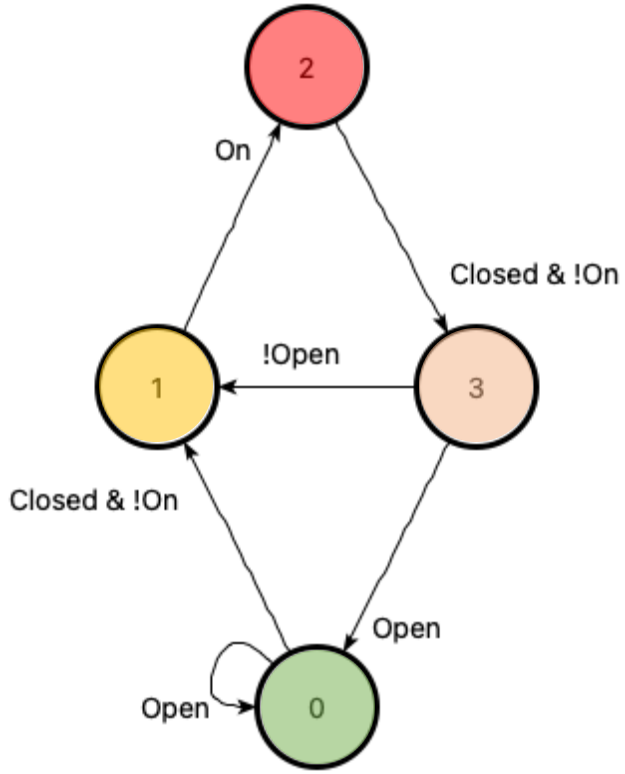


- imagine we exchange states / transitions
- minimizing the state space could be helpful

We can infinitely avoid barrier.Open (after some time)

≡ finding a cycle without 🟢 in the State Class Graph





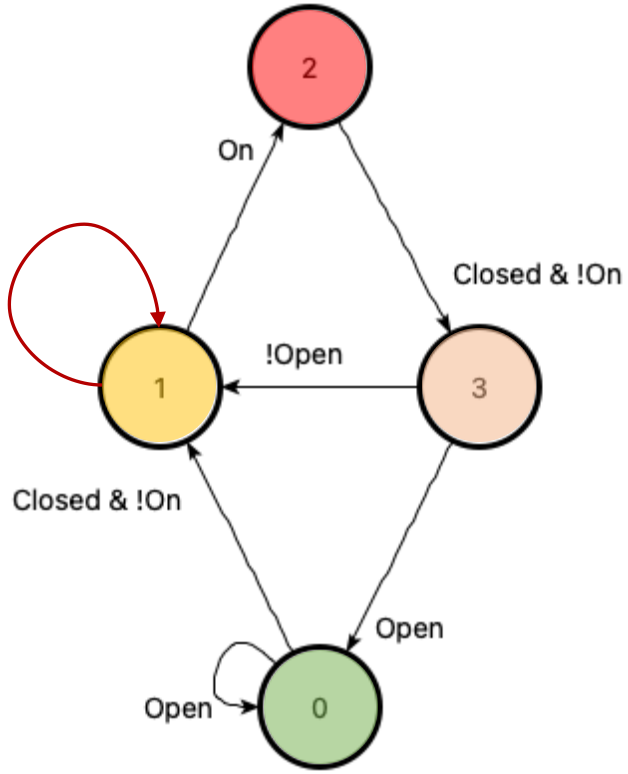
// between checking and finding cycles

🤔 we work with infinite words. What is our acceptance criterion. (generalized Büchi or not,

Model-Checking

- Generating the specification is not easy (is it ?)
- Prone to problems

  We missed some loops. A second train can approach (one train can hide another)
- Writing good properties may be just as hard as writing a good model



🤔 what is the equivalence induced by S1 ≈ S2 when S1 and S2 satisfies the same properties ?

Linear-time Temporal Logic (LTL)

👉 A stresses the fact that we express properties over ALL (∀) runs.
(not the traditional presentation)

LTL formulas ::= A φ , where φ is a path formula

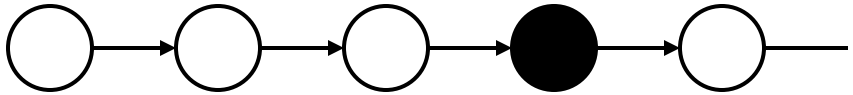
		other notation	Uppaal
φ ::= σ	state formula σ is true immediately		*n.a.*
X φ	φ is true in the next state	○ φ	*n.a.*
G φ	φ is always (globally) true	□ φ	A[] σ
F φ	φ is eventually (finally) true	◇ φ	A<> σ
ψ U φ	ψ should stay true until φ	ψ U φ	*n.a.*

👉 Hence LTL includes invariant formulas, AG, and more

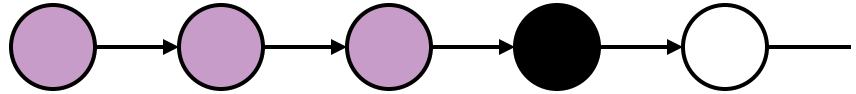
⚠ modalities/formulas can be nested ; hence F (G Open) is a valid LTL formula (this is prohibited in Uppaal)

Basic operators

(Finally) F ●

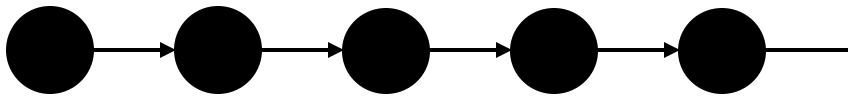


(Until) ● U ●

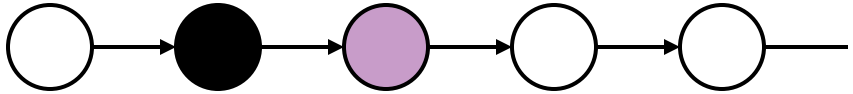


Liveness properties: something good eventually happens.

(Globally) G ●

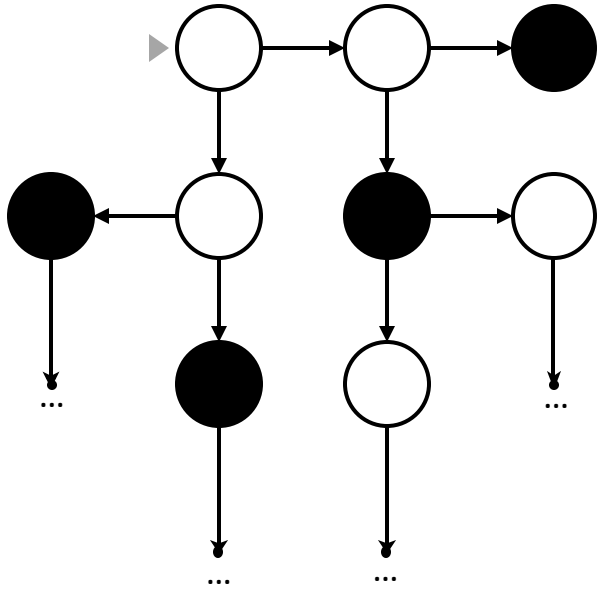


(Next) X(●) & X X(●)

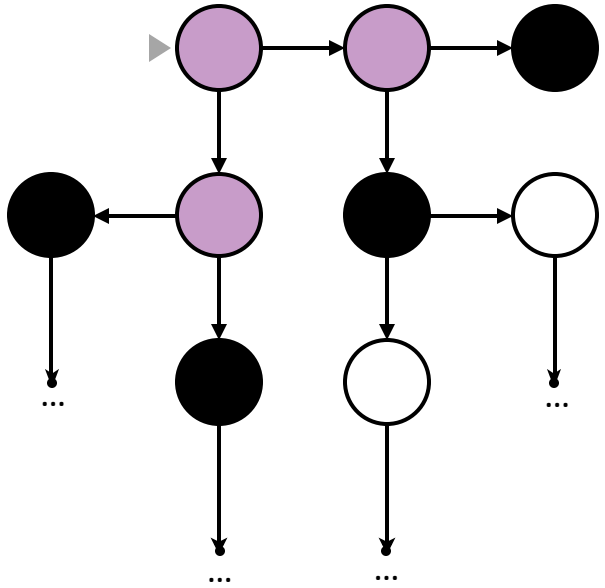


Basic operators

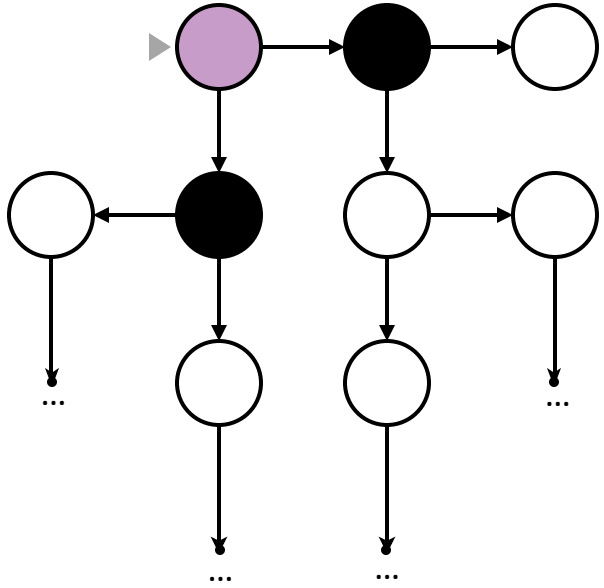
AF ●



A(● U ●)



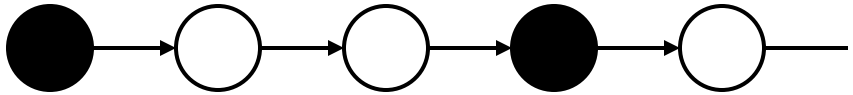
A(● & X(●))



👉 if AF false then there must be an infinite run without ● (a counterexample): “EG ! ●”.

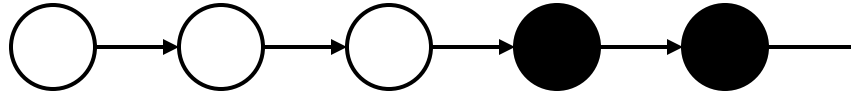
Interesting formulas

(Globally Finally) G F ●



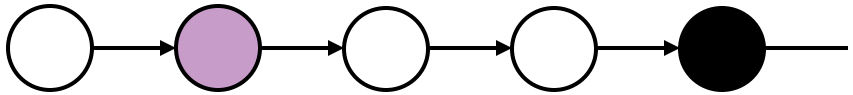
👉 infinitely often ●

(Finally Globally) F G ●



👉 reach a stable state

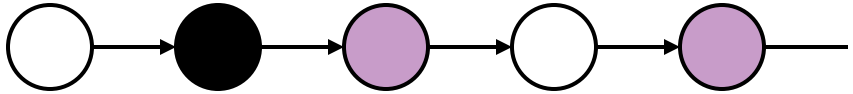
(F ●) ⇔ (F ●)



👉 not one without the other

👉 if order is important: [(F ●) & (!● U ●)] | […] | G(!● & !●)

(strong fairness) (G F ●) ⇔ (G F ●)



👉 but not necessarily as often

LTL – derived operators and other remarks

F φ ≡ True U φ

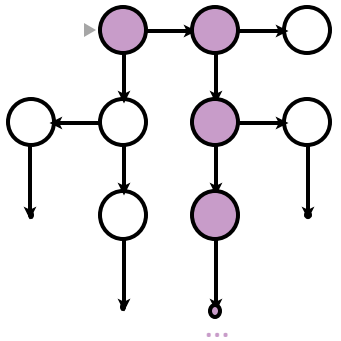
F φ ≡ φ ∨ X (F φ) it is a fixpoint: μz.(φ ∨ X z)

G φ ≡ ¬ F ¬ φ hence F and G are dual

(ψ W φ) weak-until (φ may never be true) ≡ (ψ U φ) ∨ G ψ

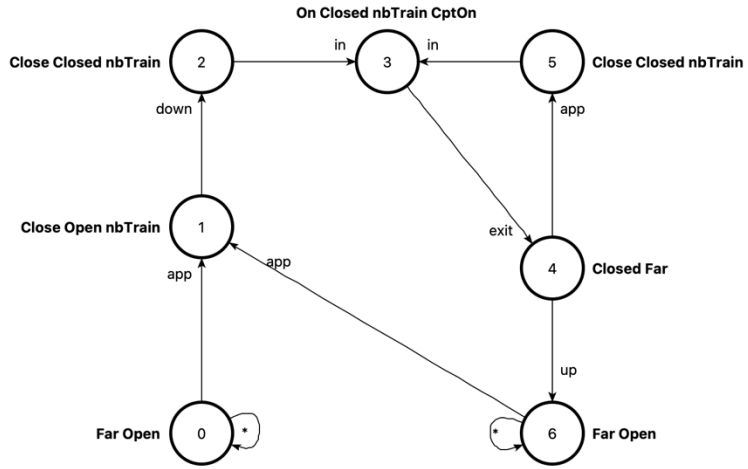
⚠ both AF (¬ φ) and AG φ can be false at the same time

(AF and AG are not dual)



Remember our running example

Verdicts depends on the order of events in an execution



Examples:

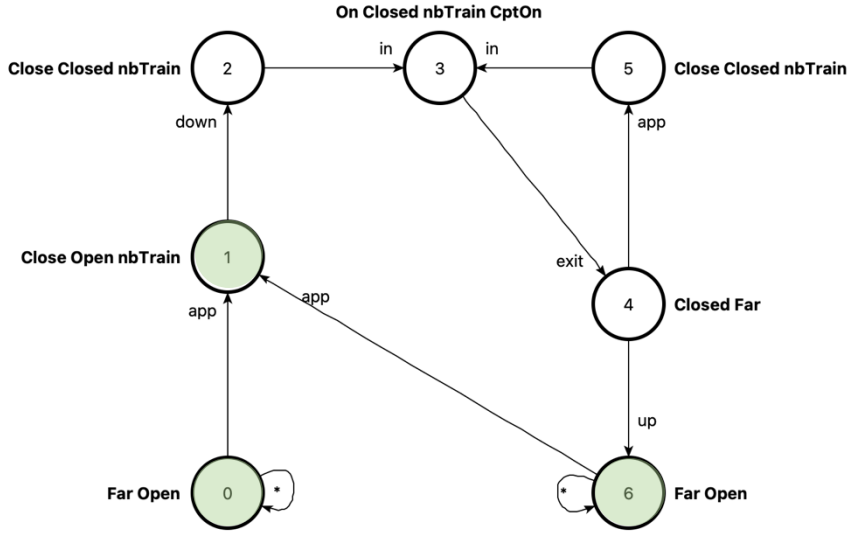
- There is always a train *exit* before a barrier *up*
- Barrier is *Closed* until train is *Far*
- Barrier *Closed* is always (eventually) *Open*
- Train *On* is followed by *Far* (next)
- There is always barrier *up* between train *Close* and train *Far*
- System is *reinitializable*

Temporal properties

- There is always a train ~~exit~~ On before a barrier ~~up~~ Open
- Barrier is *Closed* until train is *Far*
- Barrier Closed is always (eventually) Open
- Train *On* is followed by *Far* (next)
- There is always barrier *up* between train *Close* and train *Far*
- System is *reinitializable*

Idea: check where φ = (¬On U Open) is false

$$G\ (\neg Open \Rightarrow \neg\varphi)$$



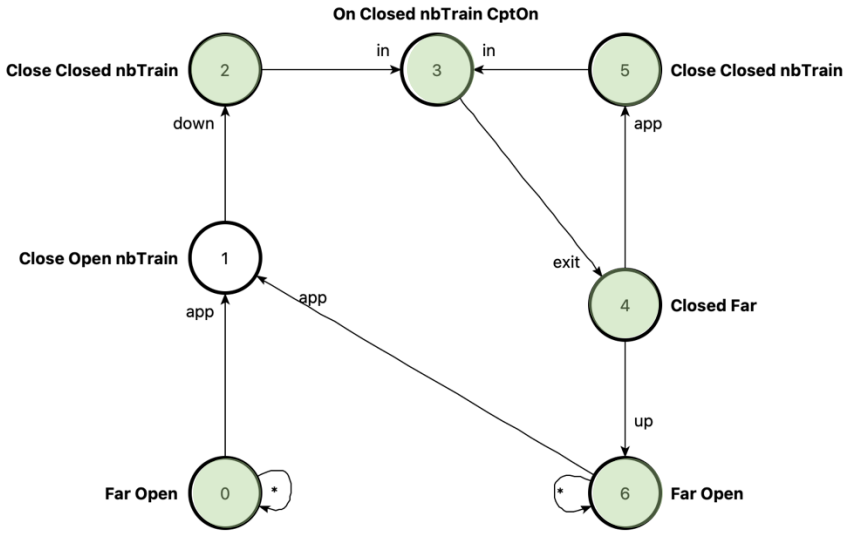
👉 we can “change” formulas about the past (Past-LTL) into formulas on the future

Temporal properties

- There is always a train exit On before a barrier up Open
- Barrier is Closed until train is Far
- Barrier Closed is always (eventually) Open
- Train *On* is followed by *Far* (next)
- There is always barrier *up* between train *Close* and train *Far*
- System is *reinitializable*

Check if (Closed U Far) is invariant

G (Closed U Far) is ❌



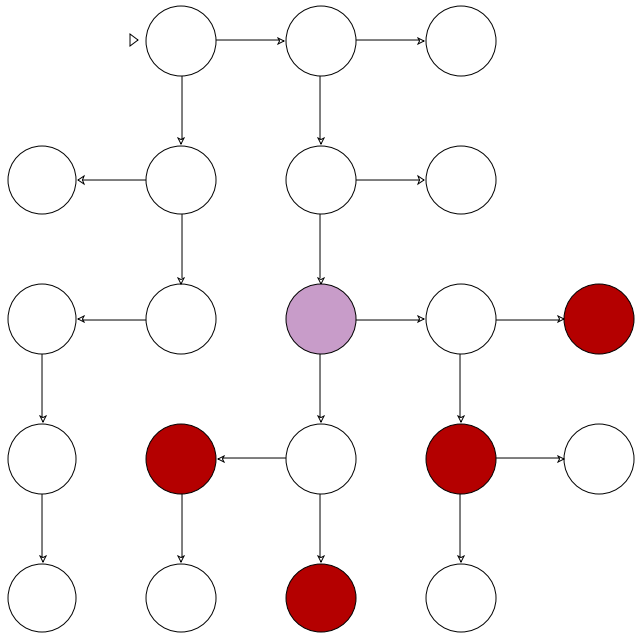
👉 corrected version: barrier stays Closed until train is Far

AG(Closed ⇒ A(Closed U Far)) is ✅

Leadsto: $\psi \longrightarrow \varphi$ or ψ -->φ

- There is always a train *exit* before a barrier *up*
- Barrier is *Closed* until train is *Far*
- Barrier *Closed* is always (eventually) *Open*
- Train *On* is followed by *Far* (next)
- There is always barrier *up* between train *Close* and train *Far*
- System is *reinitializable*

G (*Closed* ⇒ F *Open*)



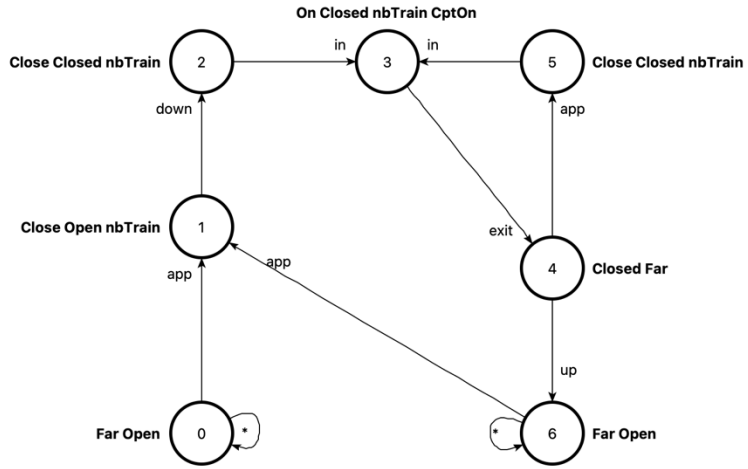
👉 useful to check “starvation”: a process that asks a resource will eventually get access to it.

👉 the “leadsto” formula, also written *Closed* ⟶ *Open*

Temporal properties

extra

- There is always a train ~~*exit*~~ *On* before a barrier ~~*up*~~ *Open*
- Barrier is *Closed* until train is *Far*
- Barrier *Closed* is always (eventually) *Open*
- Train *On* is followed by *Far* (next)
- There is always barrier *up* between train *Close* and train *Far*
- System is reinitializable



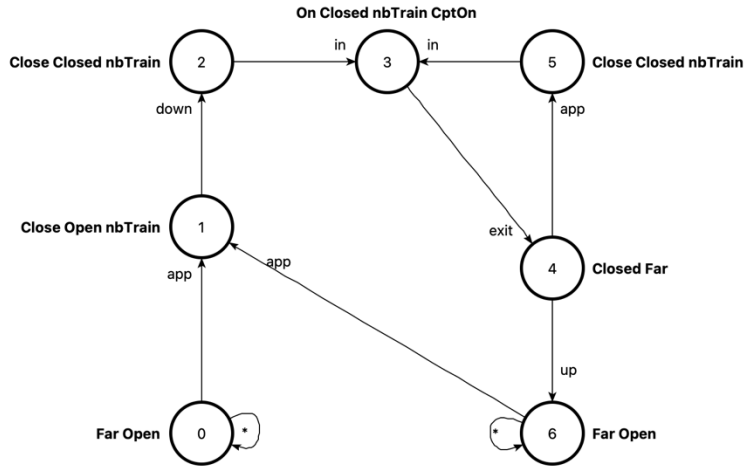
Possible by nesting Until:

$s1\ U\ (s2 \wedge (s2\ U\ (s3 \wedge (s3\ U\ \ldots))))$

👉 we cannot easily write regexp on paths

Temporal properties

- There is always a train ~~*exit*~~ *On* before a barrier ~~*up*~~ *Open*
- Barrier is *Closed* until train is *Far*
- Barrier *Closed* is always (eventually) *Open*
- Train *On* is followed by *Far* (next)
- There is always barrier up between train Close and train Far
- System is *reinitializable*



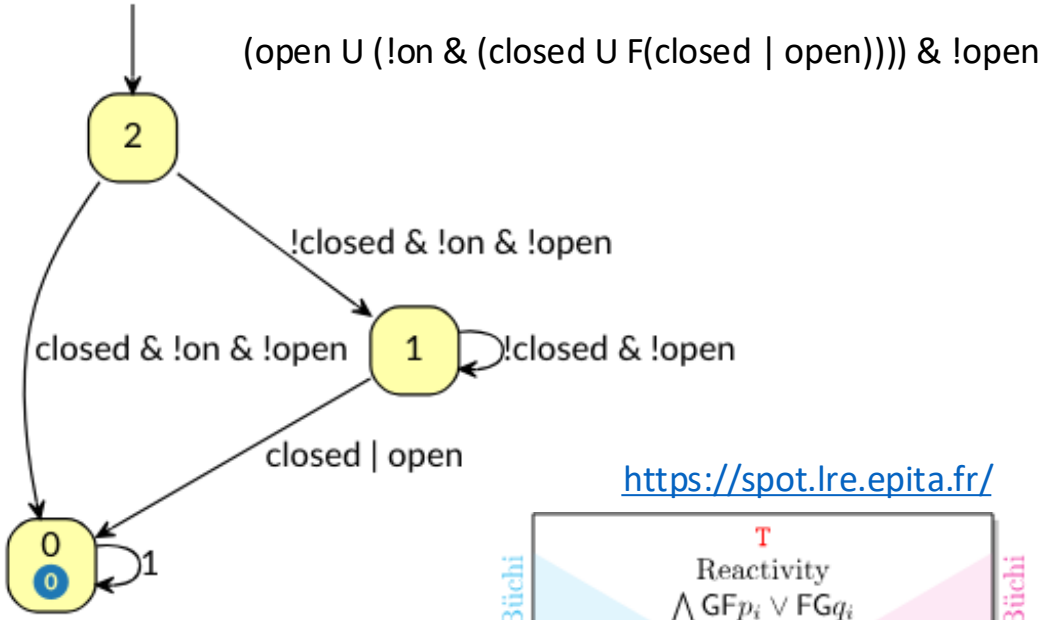
Equivalent to: always globally (AG) we can find (∃) a path going to ~~state 0~~.

👉 a state matching Far & Open

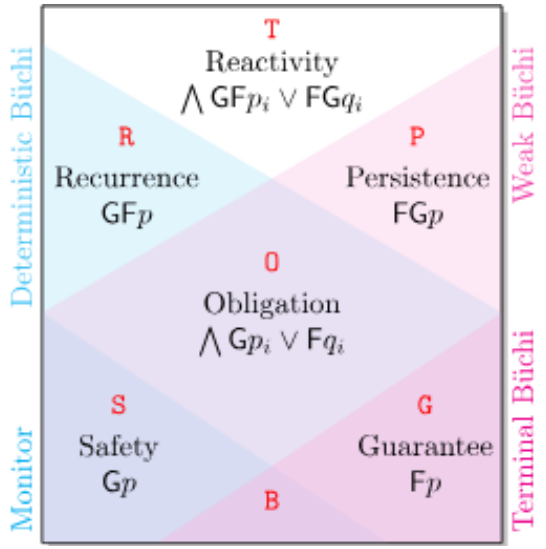
Not possible in LTL, since property must be true on all successor paths

LTL (quick) conclusion

- Model-checking LTL is decidable
  - We can build a Büchi automata $\mathcal{A}_\varphi$ "equivalent" to any LTL formula φ
  - Check emptiness of the $\otimes$ between KS and $\mathcal{A}_{\neg\varphi}$ (involves computing SCC)
  - $\mathcal{A}$ can be exponentially larger than φ
  - The problem is PSPACE-complete
- LTL is less expressive than Büchi.
- It is not even that expressive ≡ monadic FO[<], "star-free language", …



https://spot.lre.epita.fr/



👉 Pnueli (1977) was the first to use temporal logic for reasoning about concurrency.

👉 first version of SPIN was publicly released in January 1991 (G. Holzmann, Bell Labs)

LTL and time

Some undecidability results

Timed Temporal Logic

- **Idea**: use *explicit clocks* (TPTL: Timed-PTL) or add timing constraints to each modality (MTL: Metric-TL)
- MITL ≡ MTL without punctual intervals

👉 PTL (propositional temporal logic) is another name for LTL

👉 MTL is PTPL with local clocks

$$\mathrm{F}\,(a \wedge x.(\neg b\ \mathrm{U}\ (b \wedge x \leq 3)))$$

$$x.\mathrm{F}\,(a \wedge x \leq 1 \wedge \mathrm{G}(x \leq 1 \Rightarrow \neg b))$$

TPTL

$$\mathrm{G}_{[3,\infty[}\,(\phi_1\ \mathrm{U}_{[1,3]}\ \phi_2)$$

$$\mathrm{G}\,(\phi \Rightarrow \mathrm{F}_{[1,1]}\ \psi)$$

MTL

Complexity

Theorem: Model-checking TPTL and MTL is undecidable. For MITL, model-checking and satisfiability are both decidable but “expensive” (EXPSPACE-complete).

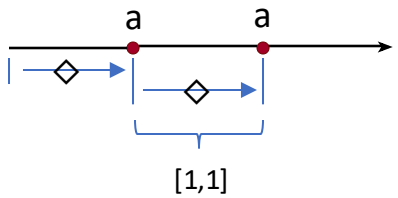
For every MITL formula we can build a T.A. with equal semantics. Then, it is not possible to define a T.A. whose language is: “*all runs except those where delay between (some) two events is 1*”.

$$\neg \mathrm{F}\left(a \wedge \mathrm{F}_{[1,1]}\ a\right)$$

J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. LICS’05. 2005

R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. Journal of the ACM, 43(1), 1996.

R. Alur and D. L. Dill. A theory of timed automata. Theoretical Computer Science, 126(2), 1994



What you should take away

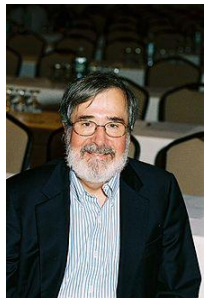
- LTL model-checking and automata (not only Büchi)
- Main ingredients are: product and finding cycles (computing SCC)
- Very efficient algorithms

  Cons: P-complete (not easily //); Pros: can be computed on-the-fly
- works well with State Class Graph (preserves runs)

  using temporal logic (untimed) on timed model can still be useful
- Tina includes a LTL model-checker: selt (for SE-LTL)

👉 Tina v1 (1983) ; Tina-selt (2006, v2.8)

Edmund Clarke, E. Allen Emerson, Joseph Sifakis. Turing award (2007) for their role in developing Model-Checking into a highly effective verification technology that is widely adopted in the hardware and software industries



Clarke 1945—2020

Computation Tree Logic (CTL)

Branching time logic

Computation Tree Logic (CTL)

👉 in CTL every modality is a pair of a path quantifier: A (∀) or E (∃),
and a temporal modality; X, F, G, U

φ ::= σ state formula σ is true immediately

$\psi \wedge \varphi$, $\neg \varphi$, …	Boolean comb. of formulas
EX φ	φ is true in (at least one) successor
EG φ	φ is always true, along one successor path
ψ EU φ	ψ true until φ, along one …
EF φ	φ is eventually true, along one …
AX φ	$\neg$ EX $\neg \varphi$
AG φ	$\neg$ EF $\neg \varphi$
ψ AU φ	$\neg ((\neg\varphi$ EU $\neg\psi \wedge \neg\varphi) \wedge \neg$ EG $\neg\varphi)$

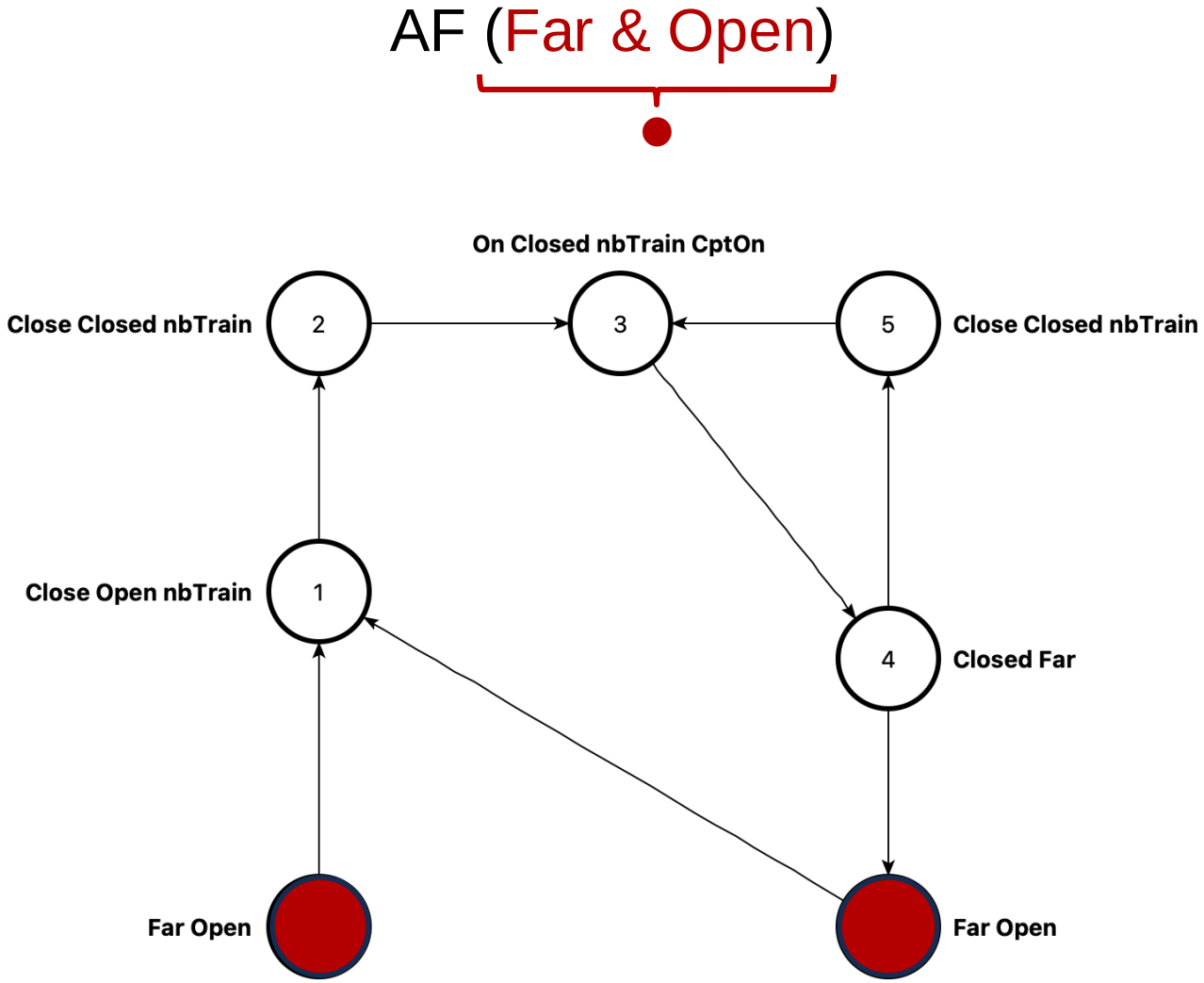
👉 EX, EG and EU are enough,
e.g. EF ≡ True EU φ

Model-Checking CTL

- Tag states with matching subformulas
- Direct for atomic state formulas
- Iterate, until fixpoint, using relations, e.g.:

  AX ● : all successors are ●

  AF ● ≡ ● or AX (AF ●)



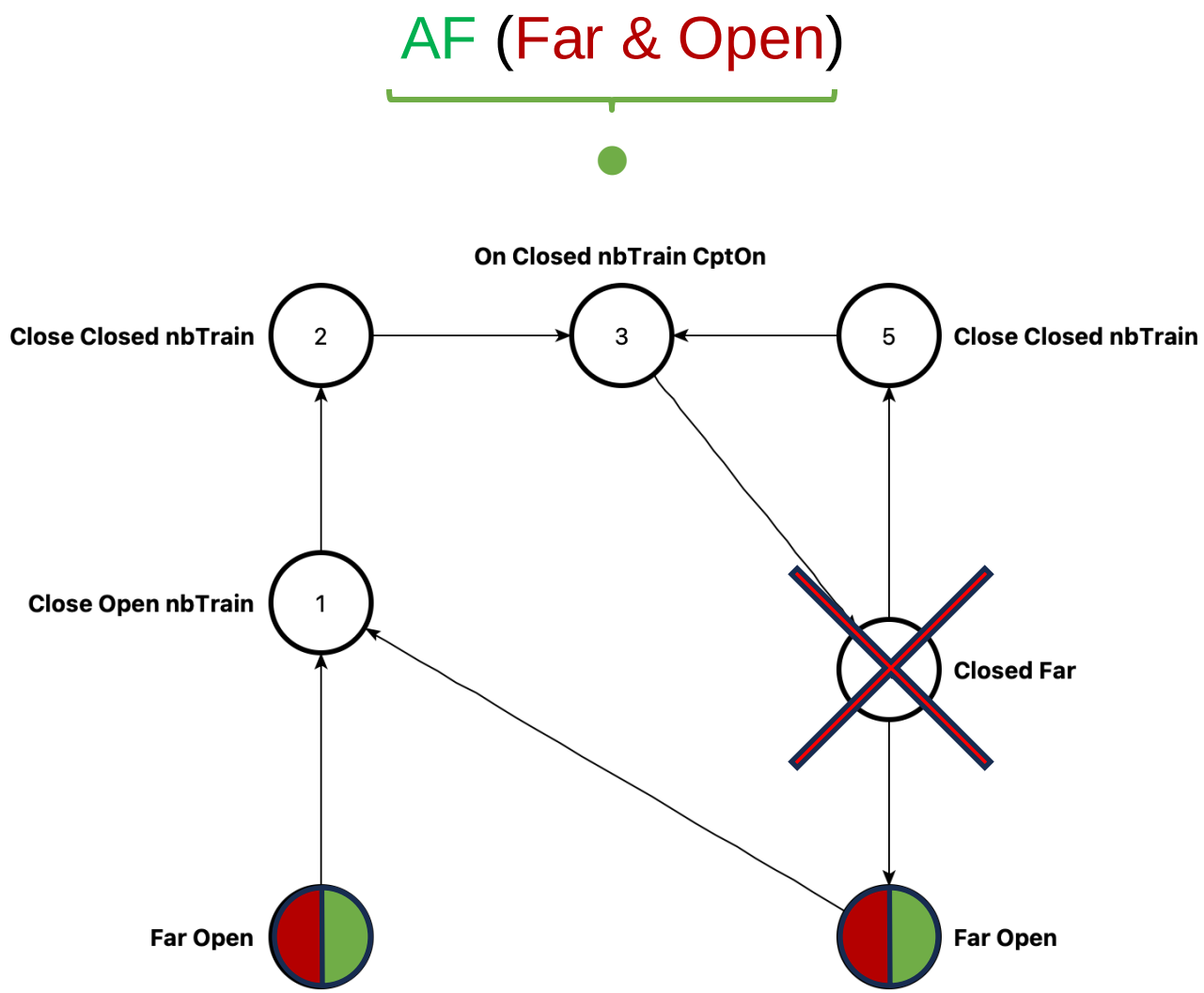
Model-Checking CTL

AX ● : all successors are ●

AF ● ≡ ● or AX (AF ●)

A CTL formula φ is true for some states (locally) : $\langle\!\langle\ \varphi\ \rangle\!\rangle = \{\ s \mid s \vDash \varphi\ \}$

Global semantics: when φ is true on the initial state



Model-Checking CTL

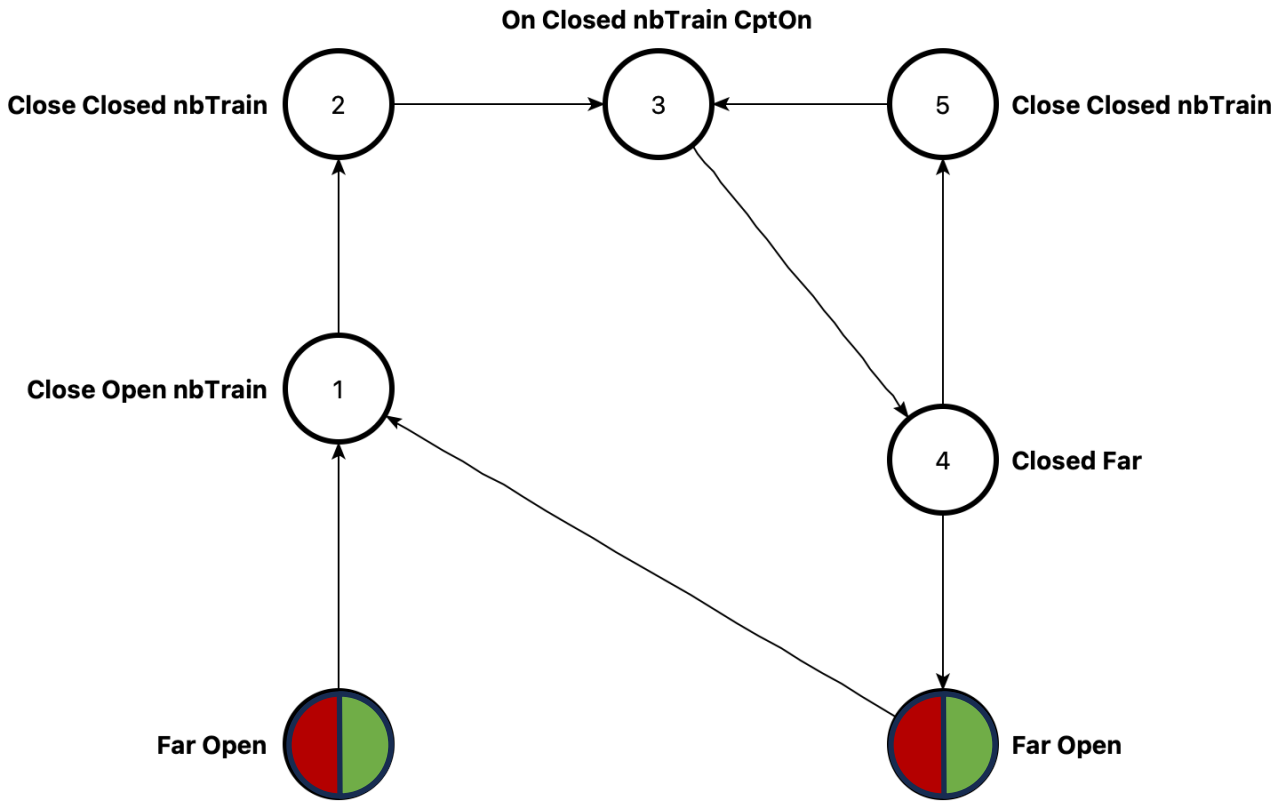
EX ● : one successor is ●

EF ● ≡ ● or EX (EF ●)

AX ● : all successors are ●

AG ● ≡ ● and AX (AG ●)

AG (AF (Far & Open))

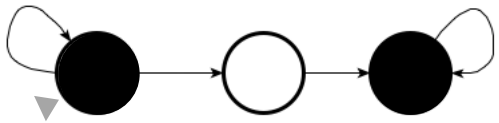


CTL*

- CTL* allows free use of temporal operators: (A G F ●) ∧ (E G ●)
- Superset of CTL and LTL

- LTL formulas A(F G ●) [stability] and A(G F ●) [infinitely often] cannot be expressed in CTL
- CTL formula AG EF ● ≡ it is always possible (but not necessary) to reach ● [e.g. reinit] cannot be expressed in LTL

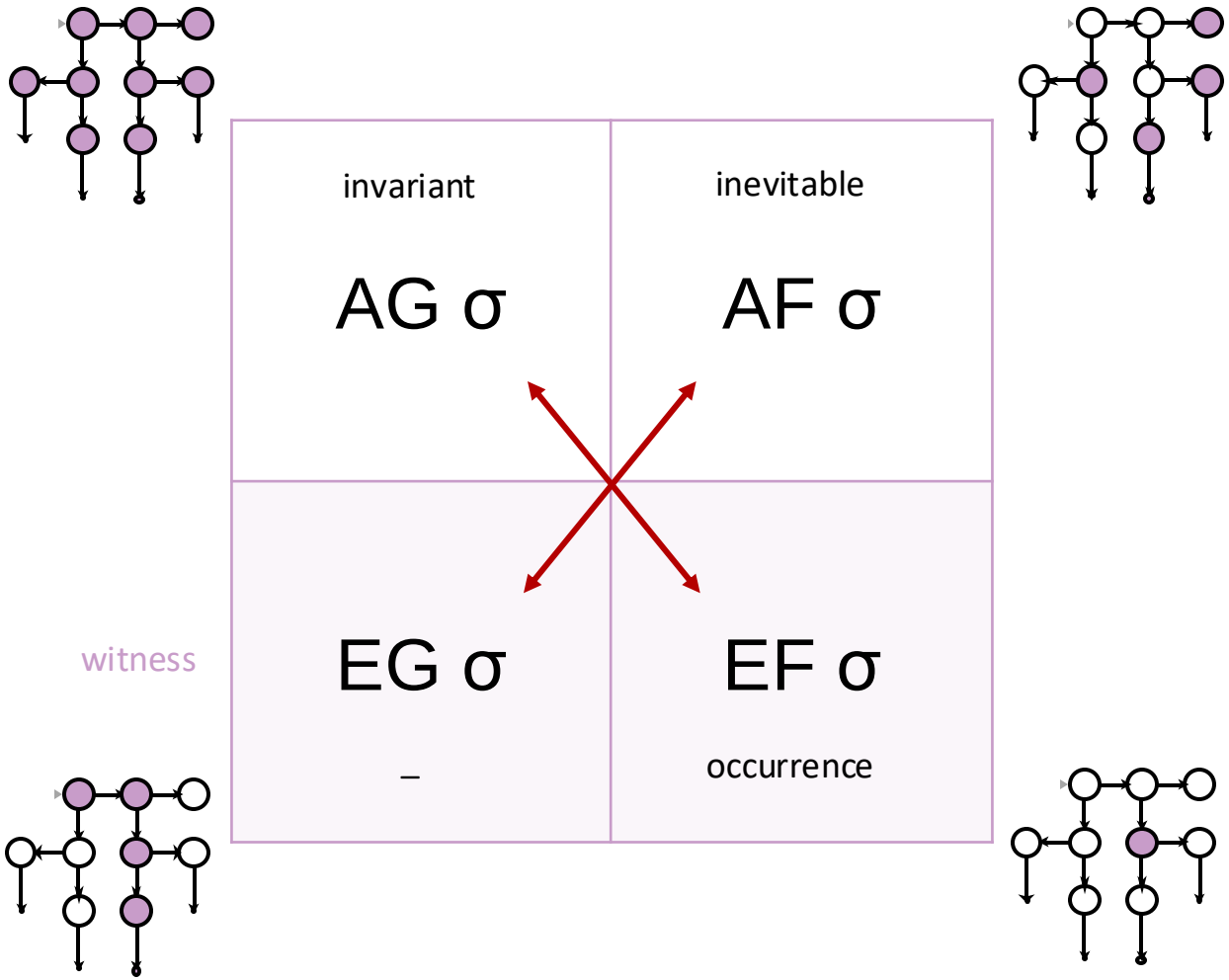
⚠ AF AG ● (in CTL) is not the same as A(F G ●) in LTL.
For the model (right) CTL formula is false, LTL is true.



Hands-On

Uppaal and Romeo

Uppaal queries ⊊ LTL ∩ CTL



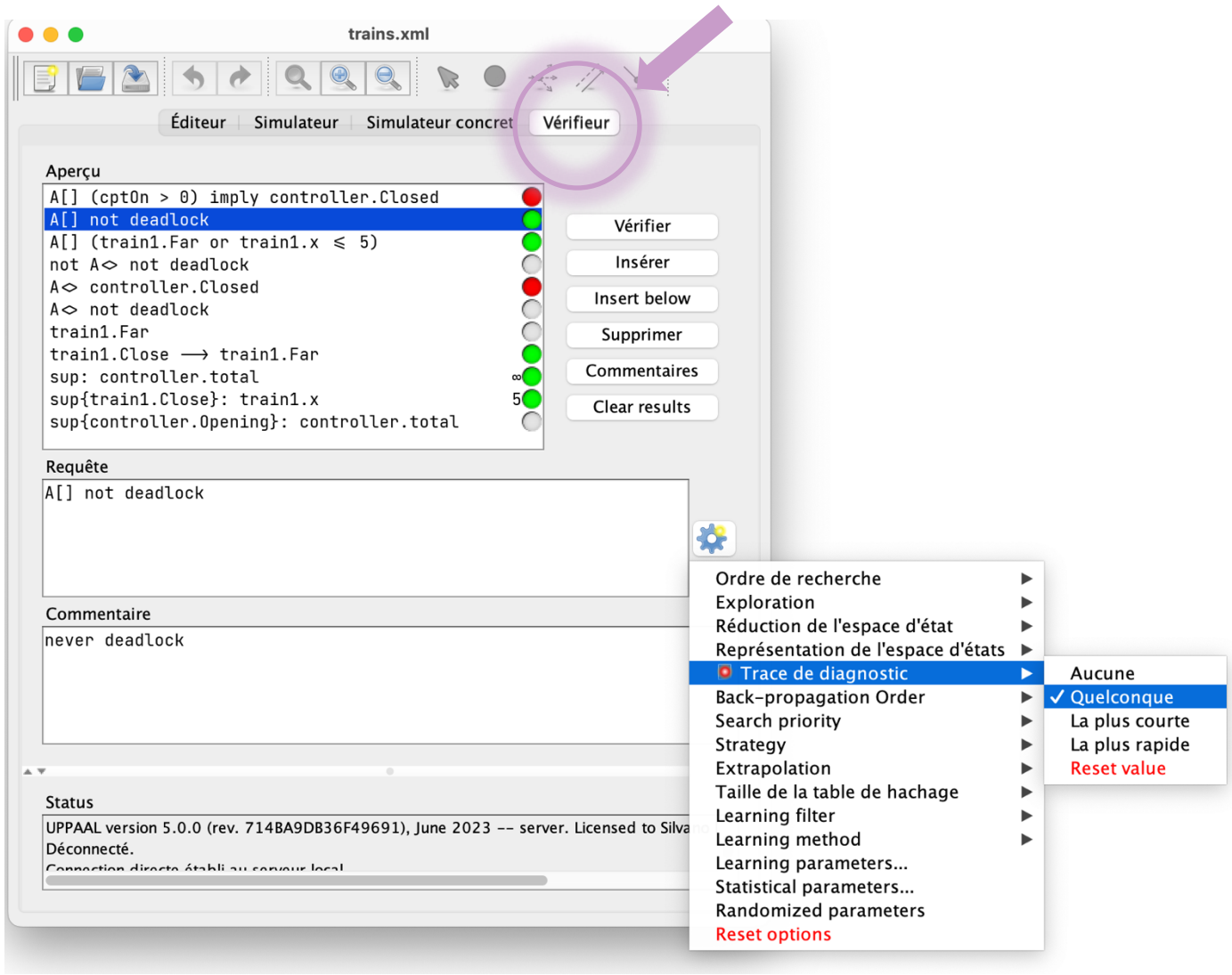
+

leadsto

$$\sigma_1 \longrightarrow \sigma_2$$

- $\sigma_1 \longrightarrow \sigma_2$ is AG($\sigma_1 \Rightarrow$ F σ_2) [LTL] but also AG($\sigma_1 \Rightarrow$ AF σ_2) [CTL]
- σ: only atomic state formula are allowed, but it possible to use clocks
- only 3 different cases (one is reachability), hence we can use specialized algorithms.
- **Uppaal** recently added queries for sup and inf of clocks, with predicates (and bounds in v5.1 beta)
- **Tappaal**: verification of Timed-Arcs Petri nets (does not include leadsto and also forbids nested modalities).

Uppaal hands-on exercices



1. try at least one formula

   `E<> controller.Closed`

2. use diagnostic trace to check a witness for this property

TCTL in Romeo

Romeo includes a TCTL model-checker

p, q ::= deadlock | a + b != 2 | …

φ ::= E (p)U[a,b](q)

| A (p)U[a,b](q)

| EF[a,b] (p)

| AF[a,b] (p)

| EG[a,b] (p)

| AG[a,b] (p)

| (p)-->[0,b] (q)

leadsto

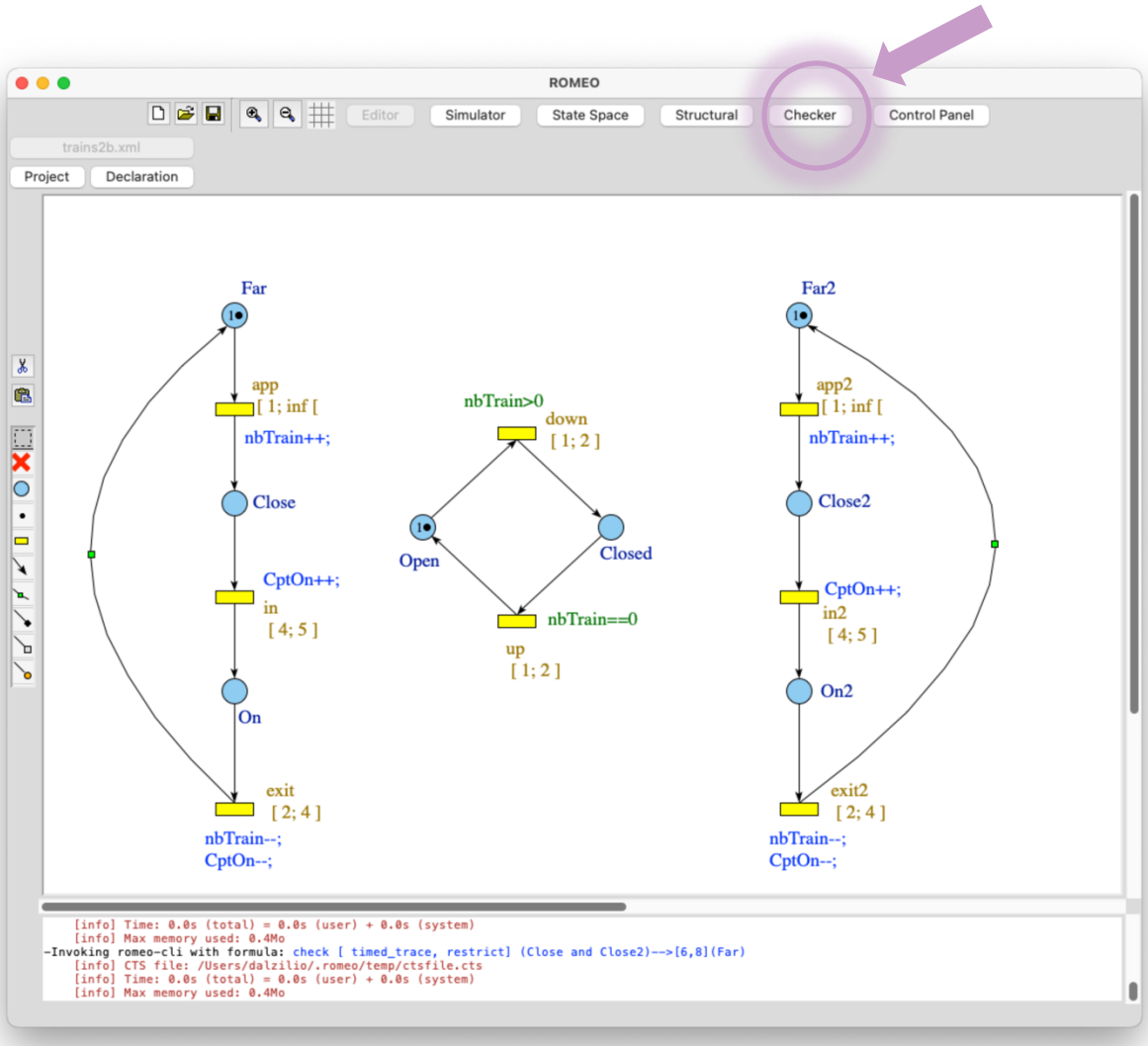
EF[0,10] (P2 - P3 >0) : reachability of a marking where M(P2) > M(P3) in less than 10 time units.

A[0,20] (P2==1) U (P3>2) : on all runs, before time 20, there is a token in P2 until there are more than 2 tokens in P3.

AG[0,inf]#<=3 (not deadlock) : there are no deadlocks in less than 3 “steps” from the initial state.

- you can declare “step bounds”: by adding #<=N or #<N after the time interval.
- A missing interval is equivalent to [0,inf]
- Kronos (1993--2002), TCTL for T.A.

Romeo hands-on exercices




```
AG ((On > 0) => (Closed > 0))

EG[3,inf] (Closed > 0)

(Close or Close2)-->[1,2](Closed)
(Close + Close2 > 0)
      -->[1,2] (Closed > 0)

~~AG (EG[5,inf](Open))~~
```

What you should take away

👉 but also automata-based, tableaux, …

- CTL model-checking and fixpoints
- Branching-time logic cannot express some fairness properties (G F φ); but LTL cannot express some liveness properties (AG EF φ).
- Very efficient algorithms based on decision diagrams BDD
- May need SCG preserving branching (expensive)
- CTL* model-checking is “mostly academic”
- Should you choose LTL or CTL … or TCTL
- Tina includes a CTL model-checker: muse (actually SE-Modal µ-Calc.)

Vardi (2001), *Branching vs. Linear Time: Final Showdown.* In proc. of TACAS

👉 CESAR (1981-82), Sifakis & Queille : model-checker for Interpreted Petri nets (IPN) and a branching-time logic

👉 EMC (1982), Clarke : model-checker for a subset of CSP with CTL (addition of fairness conditions in 86)

Useful formulas, Patterns and Observers

Property Specification Patterns

Not all formulas are equal

- often the same formulas: □ ¬ *deadlock*
- but we sometimes need complex formulas

```
AG(Q -> ![!R EU (P & !R & ([!S EU R]
                    | [!R EU (S & !R & EX([!T EU R]))]))])
```

There are some *requirement patterns*

- Dwyer's Property Specification Patterns
- Formula above reads (CTL version):
  "S,T responds to P between (Q and R)"
- Based on a survey
- Also with real-time properties

```
REQUIREMENT: Bad (initial) value never
             happens again once region
             is computed unless we
             restart the task
PATTERN: Absence
SCOPE:   After-Until
CTL: AG(!cs.r=reg-1 ->
  ![!cg.start_cont_3eo_mode_select EU …])
SOURCE: Bwolen Yang (…)
DOMAIN: requirements, avionics
```

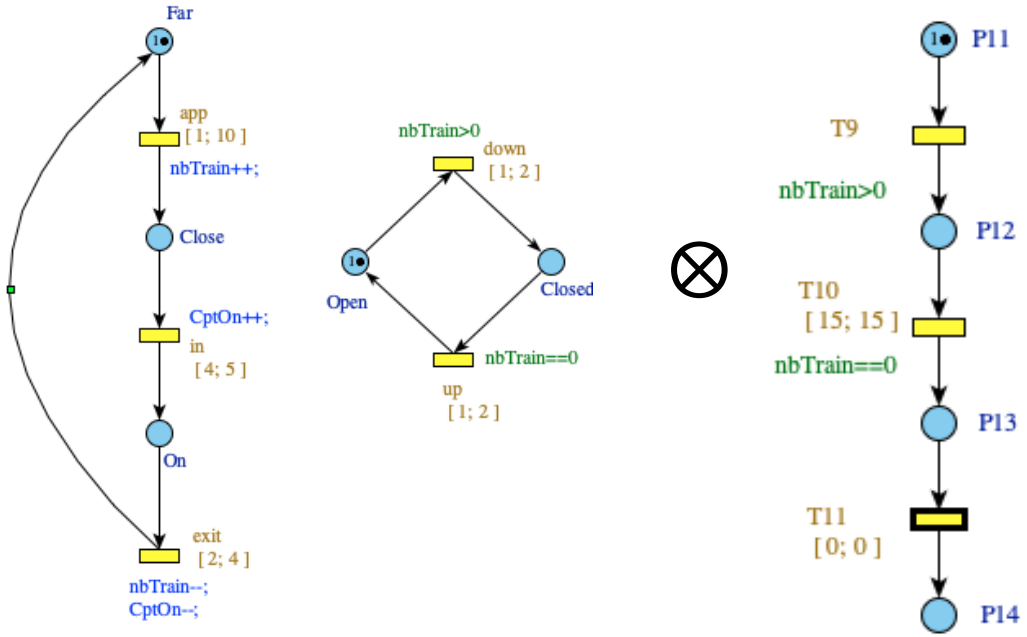
https://matthewbdwyer.github.io/psp/specifications/THEM.txt

N. Abid, S. Dal Zilio, D. Le Botlan. Real-Time Specification Patterns and Tools. FMICS 2012.

Observer-based verification

Idea: check a simple property (e.g. reachability) on the product of a system, $\mathcal{A}$, and an observer, $\mathcal{O}$

Example: barrier does eventually open when train are not too close



`(Closed)-->(Open)` ❌

vs

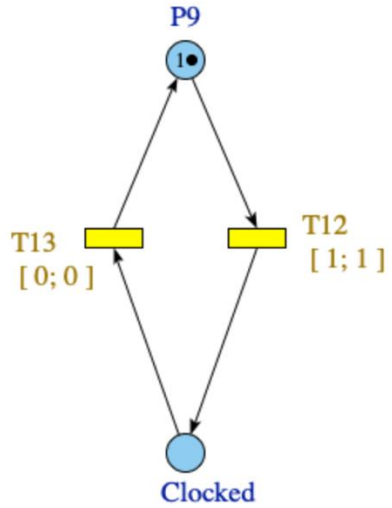
`EF(P13) and AG(P13 => Open)` ✅

Place P13 encodes: no approaching trains for 15s

Observer-based verification

Idea: check a simple property (e.g. reachability) on the product of a system, $\mathcal{A}$, and an observer, $\mathcal{O}$

Example: time progress

… ⊗ 

and check: `(Clocked)-->[1,1](Clocked)`

L. Aceto, P. Bouyer, A. Burgueno, KG Larsen . The power of reachability testing for timed automata. TCS, 2003.

checking inclusion btw the behaviour of two T.A., e.g. $\mathcal{A}$ (system) and $\mathcal{S}$ (property), is undecidable

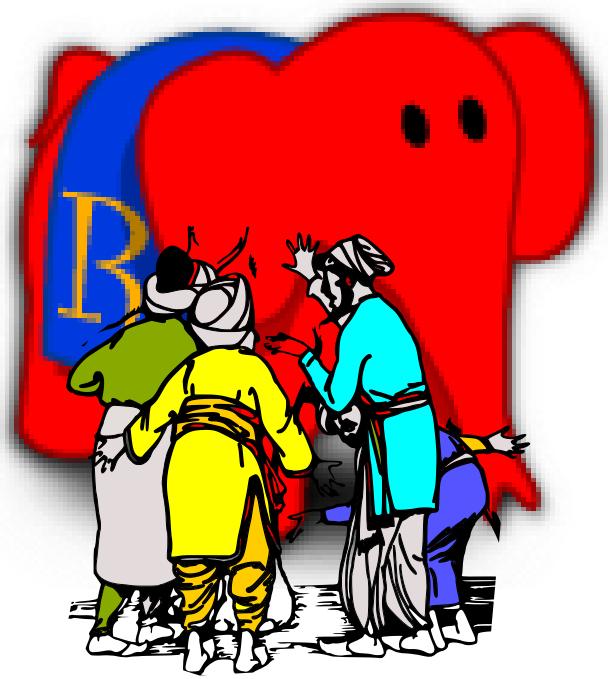
Conclusion

"Oh dear! Oh dear!
I shall be too late!"

Alice's Adventures in Wonderland (1865), CC0, British Library

Blind men and a model-checker

There are different kinds of (logical) model-checking.



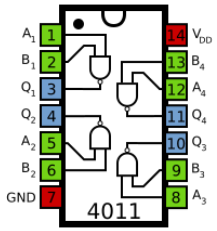
(CC0) Martha Adelaide Holton & Charles Madison Curry (1914) + the Romeo logo

hardware

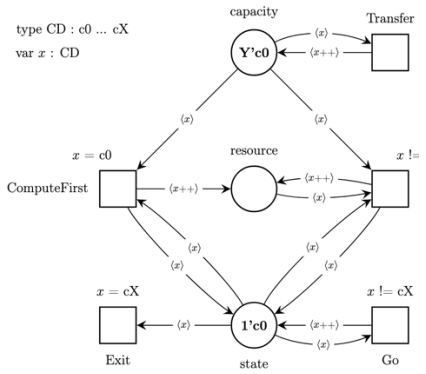
Explicit and symbolic M.-C., Bit and word-level abstractions, VHDL, …

Tools: ABC

Compet.: HWMCC, SATcomp



concurrency



Tools: Spin, Murϕ, SMV, …

Compet.: Model-Checking Contest (Petri nets)

software

C and Java programs, Windows drivers (SLAM project), …

Tools: CBMC, CPAchecker, Divine

Compet.: SVcomp

Model-Checking Pros & Cons

No proofs! Automatic

- Returns counterexamples and witnesses
- Can be used in the early design phases (with partial spec.)

Mature and fast tools, extension to the logics (PSL included in VHDL)

- Check only the model:

  la carte n'est pas le territoire
- Writing a good model is hard
- Writing temporal logic may be even harder

State explosion

- Parametric (∞) systems ; compositionnal verification

Some other related tools

Kronos (Verimag)
- last release 2002

TINA (LAAS-CNRS)

TChecker (LaBRI)
- T.A. model-checker

REDLIB (Taiwan Univ.),
- T.A. model-checker + TCTL, last release 2009

TAPAAL (Aalborg)
- Timed-Arc P.N., limited queries

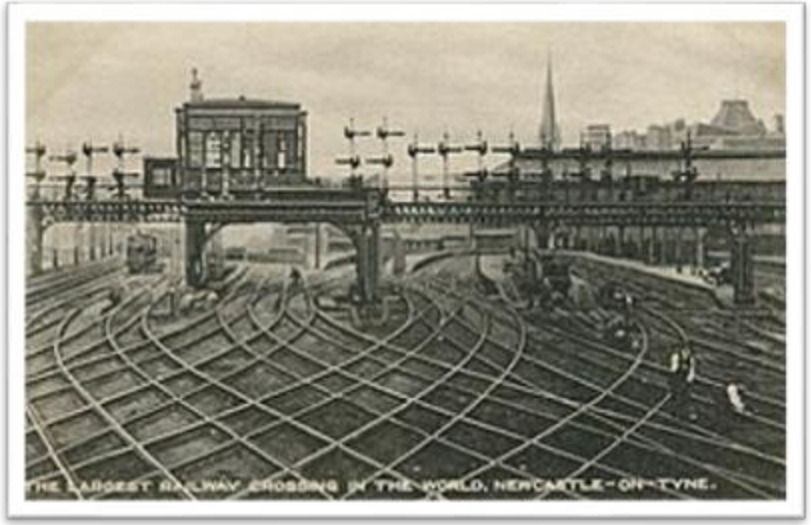
nuXmv (FBK)
- includes Time Transition Systems and specs. in MTL and realtime CTL

That is a lot of trains !

tracks		SCG	$SCG/\approx$	$SCG^{\leq}$	$SCG^{\leq}/\approx$
3		3101	578	172	41
		0.02	578	0.00	0.00
4	*size*	134501	6453	1175	76
	time	1.67	0.15	0.02	0.00
5		8557621	84510	10972	143
		179.33	2.61	0.38	0.01
6		697913229	1183782	128115	274
		24346.77	46.95	8.37	0.02
7		7.278×10^{10}	18143796	1772722	533
		–	1060.21	614.38	0.07
8		9.262×10^{12}	297205635	28208543	1048
		–	25105.87	177602.90	0.16
10		–	–	–	4126
		–	–	–	1.10
12		–	–	–	16420
		–	–	–	8.29
14		–	–	–	65578
		–	–	–	95.07
16		–	–	–	262192
		–	–	–	1418.38
18		–	–	–	1048630
		–	–	–	22407.25

State Explosion

E 29th st, Wichita, KS: 14 tracks. (presumed) largest public railroad crossing in the US in # of tracks



Newcastle-on-Tyne (XIX^{e}): largest Railway Crossing in World. [Wikipedia]

👉 obtained with Tina + symmetries on a slightly more complex version of the train model.

Some References

Books

- ★ E. M. Clarke, O. Grumberg, D. Peled. *Model Checking.* MIT Press.
- Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* MIT Press.

- L. Popova-Zeugmann (2013). *Time and Petri Nets.* Springer.
- Insup Lee et al. (2008). *Handbool of Real-Time and Embedded Systems.* Chapman & Hall.

Manuscripts

- Nicolas Markey (2011). *Verification of Embedded Systems – Algorithms and Complexity.* Mémoire HdR.

- Alexandre Duret-Lutz (2007). *Contributions à l'approche automate pour la vérification de propriétés de systèmes concurrents.* PhD Université Pierre et Marie Curie.

papers

- Vardi, M.Y. (2001). *Branching vs. Linear Time: Final Showdown*. In: Proc. of TACAS. LNCS vol 2031.
- Sascha Konrad, Betty H.C. Cheng (2005). *Real-time specification patterns.* In: Proc. of ICSE
- Edmund M. Clarke (2007). *The Birth of Model Checking.* Research Report CMU-CS-07-110
- P Bouyer (2009). *Model-checking timed temporal logics.* Electronic notes in theoretical computer science.

Equipe pédagogique

Equipe pédagogique

Auteur.rice.s : Silvano DAL ZILIO

https://dalzilio.github.io/

Crédits