

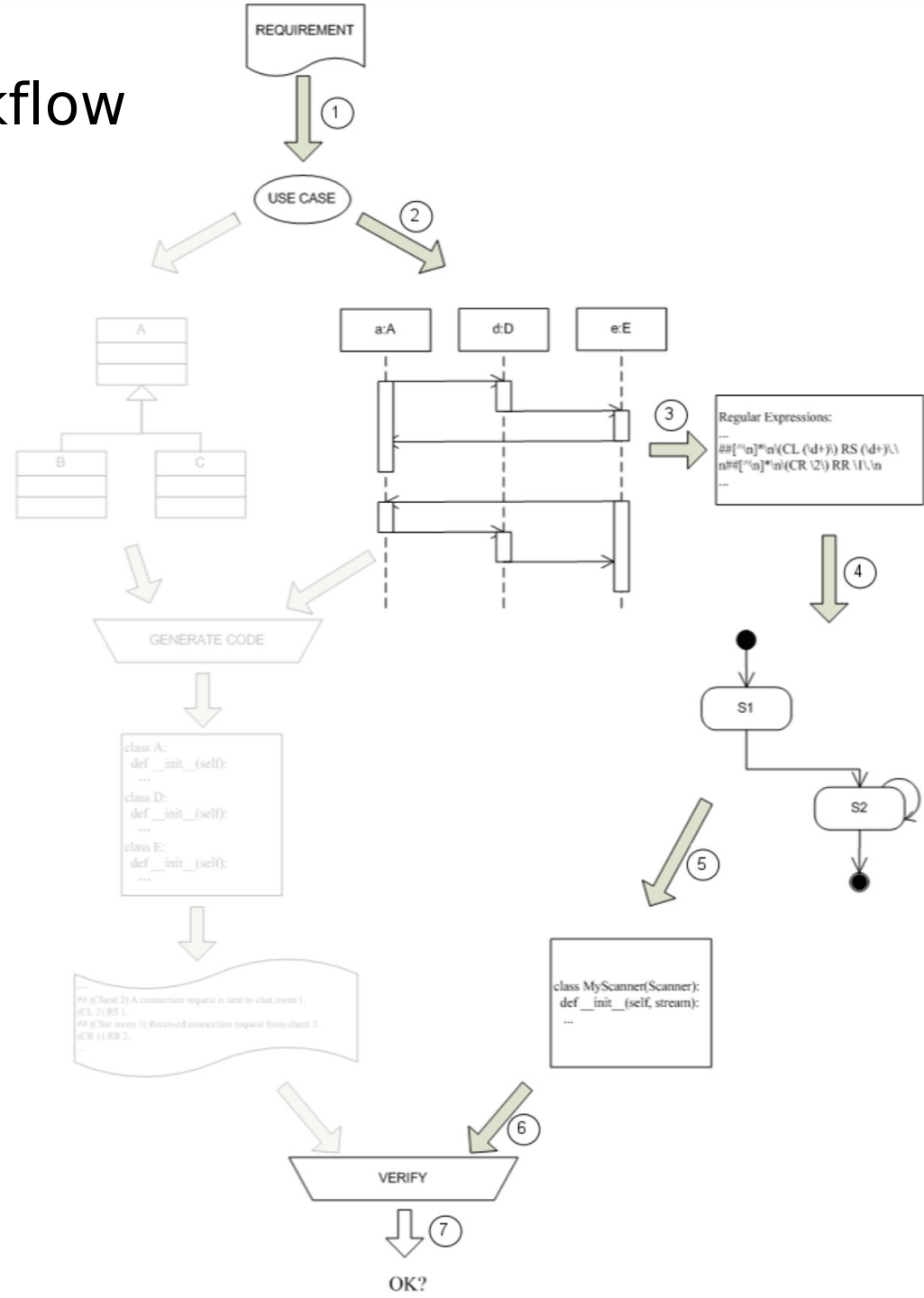


<https://www.bipdigitaltwins.org/>

Temporal Logics (LTL and STL) to express requirements used for testing/monitoring

Joachim Denil
(and Hans Vangheluwe)

Workflow



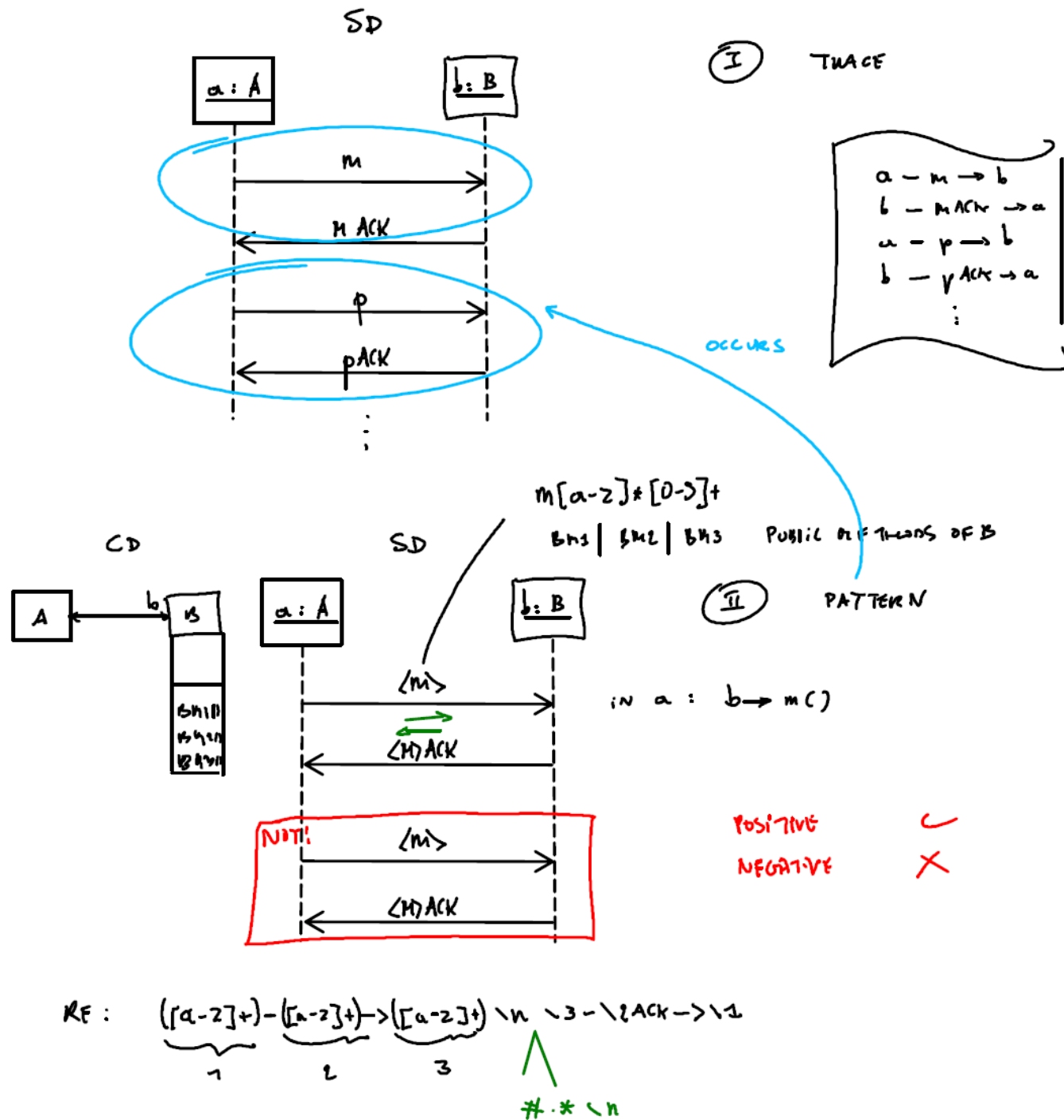
9 Formalisms with (manual|automated) transformations between them:

Formalism	Used for
Natural Language	User requirements
Structured Language	Use Cases containing Scenarios
Class Diagrams	Concepts of the problem domain and their relationships
Sequence Diagrams	The ways stakeholders interact with the system
Regular Expressions (with memory)	Encoding the interaction patterns/sequences from the Sequence Diagrams
State Automata (with memory)	Automata to recognize the patterns described in Regular Expressions
Code	An encoding of the state automaton checking the regular expression
Behaviour Traces	Obtained from the execution of an implementation of the requirements
Boolean (pass fail)	Test Result (comparing behaviour traces to a pattern encoding a requirement)

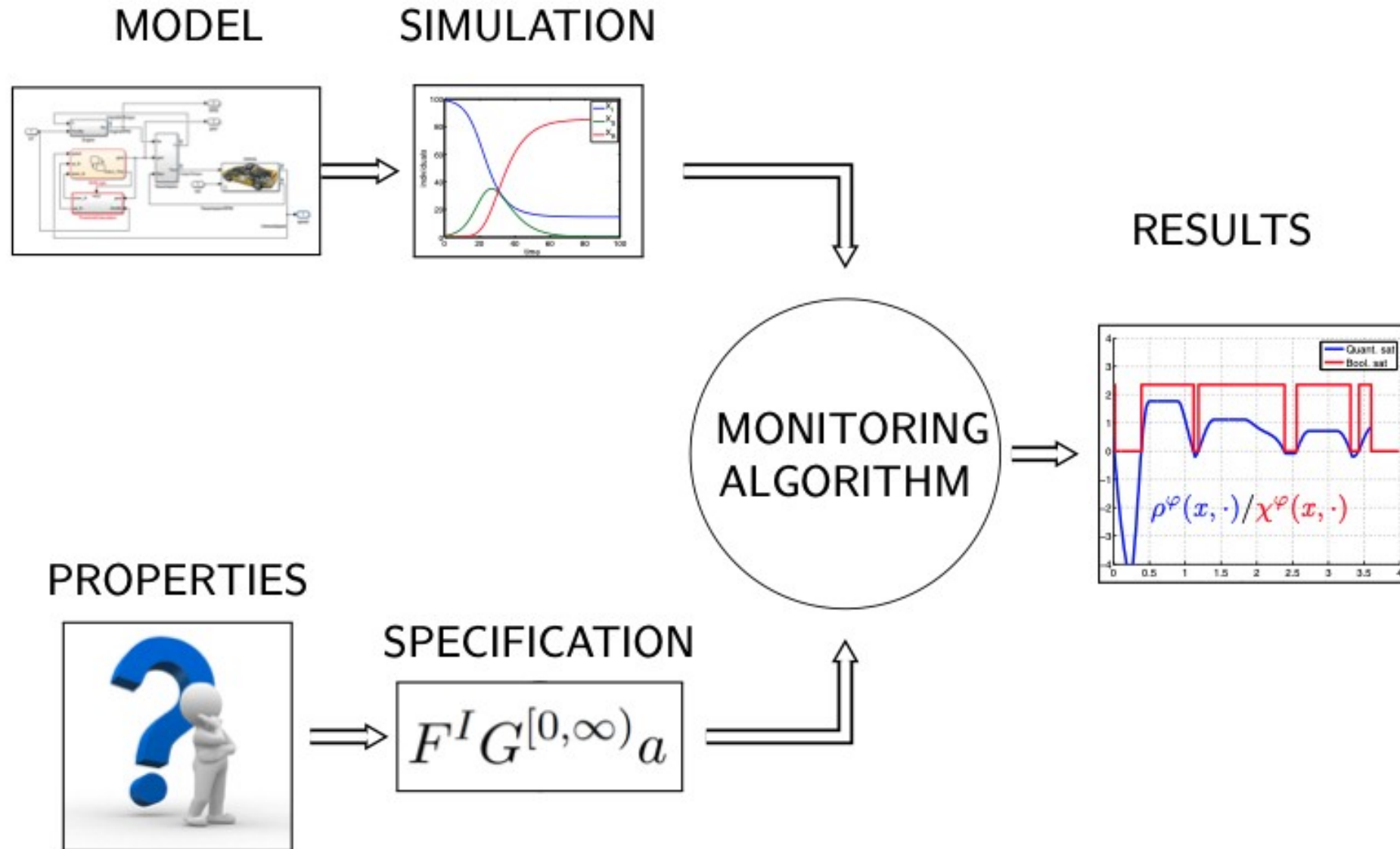
regexp

may not be the most appropriate formalism from a

- cognitive
- performance point of view



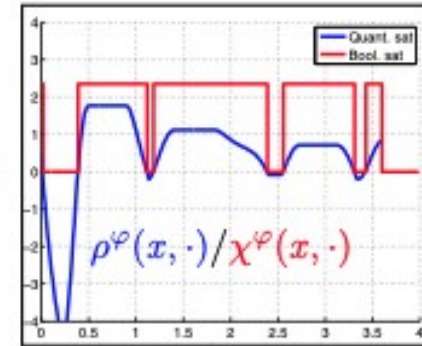
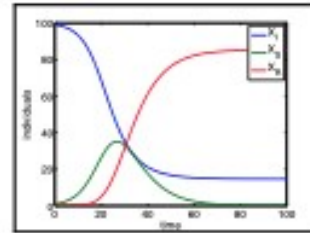
Model-Based Systems Engineering (MBSE) workflow: model and simulate



Model-Based Systems Engineering (MBSE) workflow: realize and operate

OPERATE REALIZED SYSTEM

SENSE



RESULTS

PROPERTIES

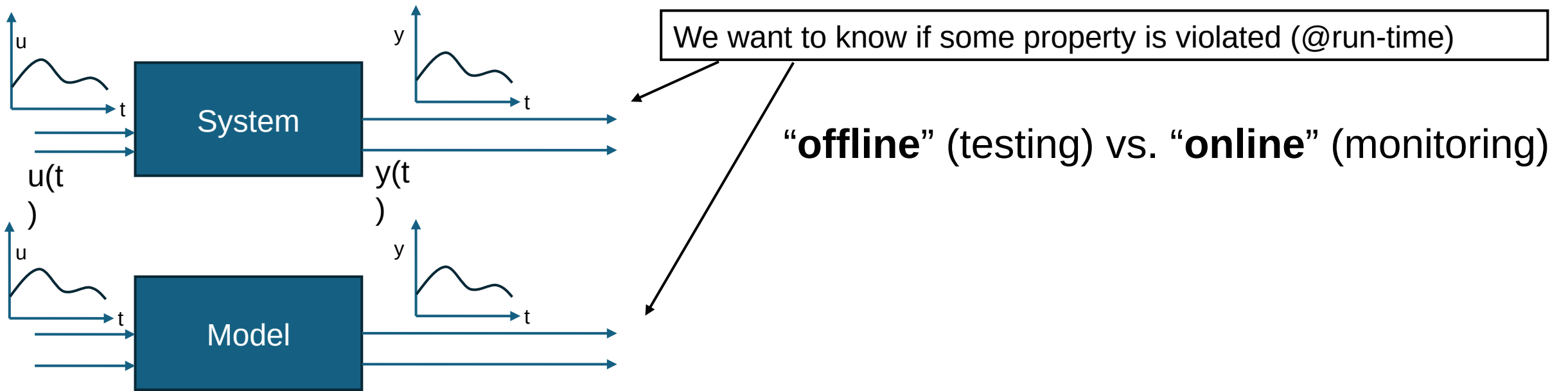


SPECIFICATION

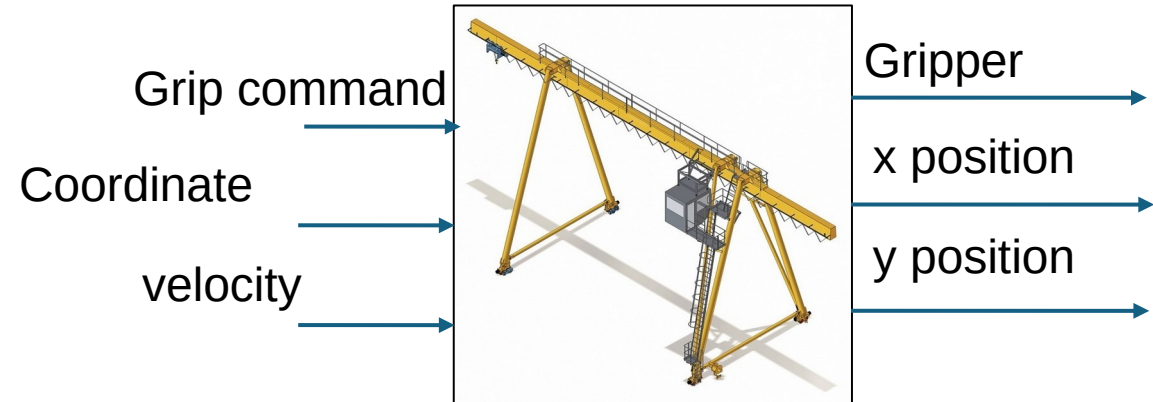
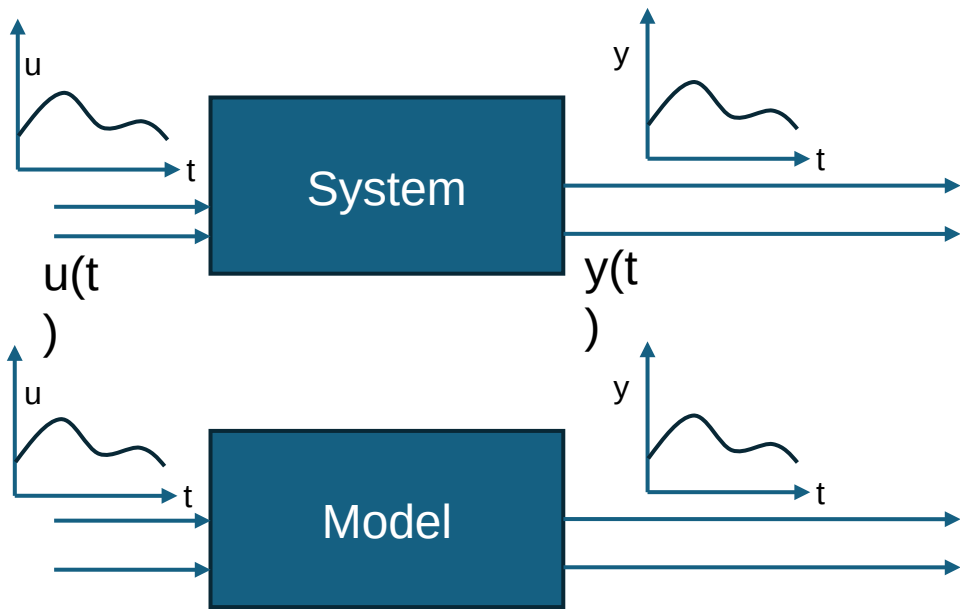
$$F^I G^{[0, \infty)}_a$$



Dynamic (i.e., evolving over time) Systems

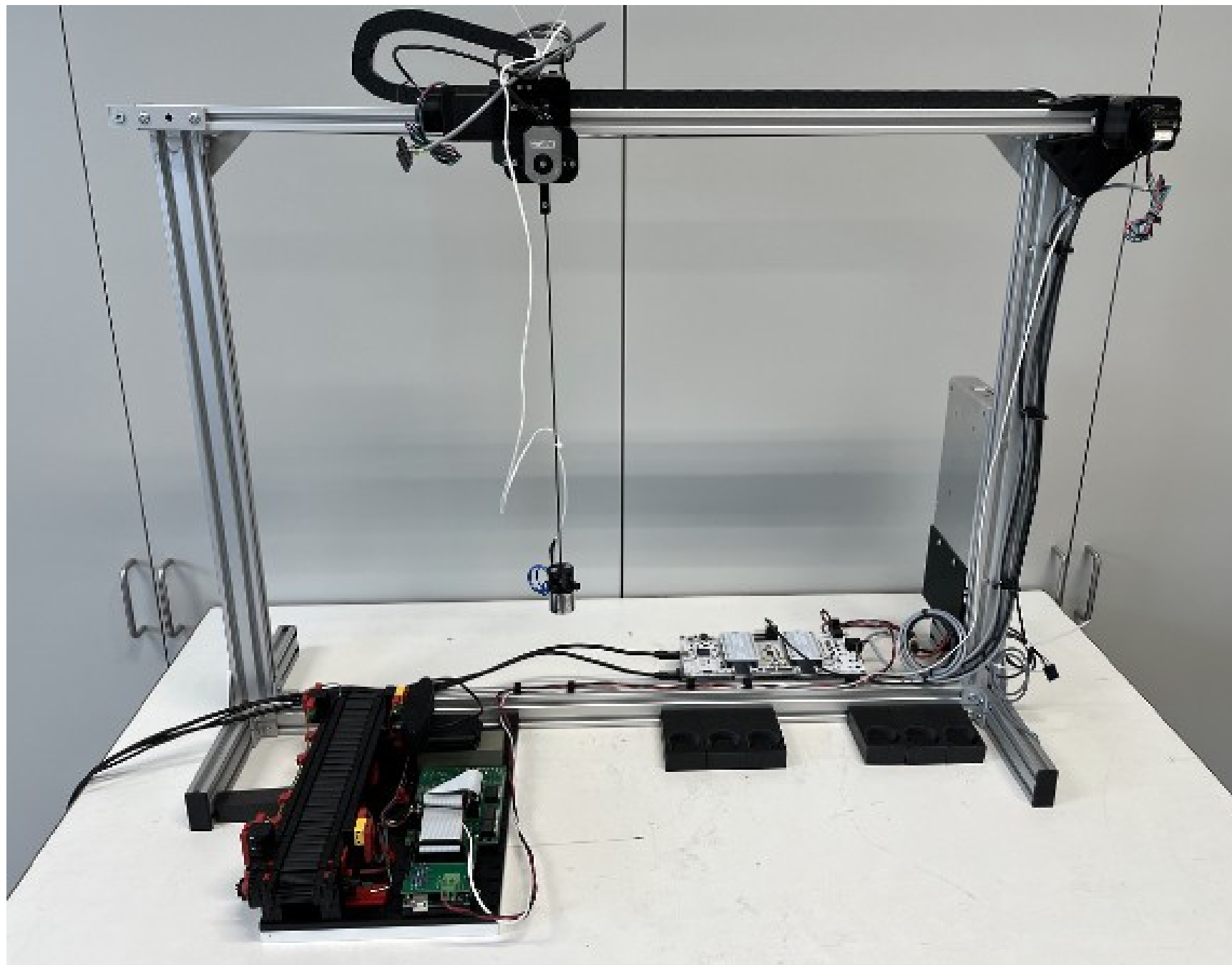


Dynamic (i.e., evolving over time) Systems



Requirements of the crane:

- Hoist height **never** not be higher than 20 meters
- **After** an emergency stop button is pressed, the power to the motors must be cut off **in the next state**, **and** the brakes must be engaged.
- ...



Types of properties one might want to check

Safety

Something bad
should never
happen

E.g. Deadlock free, mutual
exclusion

Hoist height of the crane
must never exceed 20
meters.

Liveness

Something good
will happen

E.g. Each process will
eventually enter its critical
section

By the end of each
operational cycle, the crane
must eventually come to a
complete stop.

Fairness

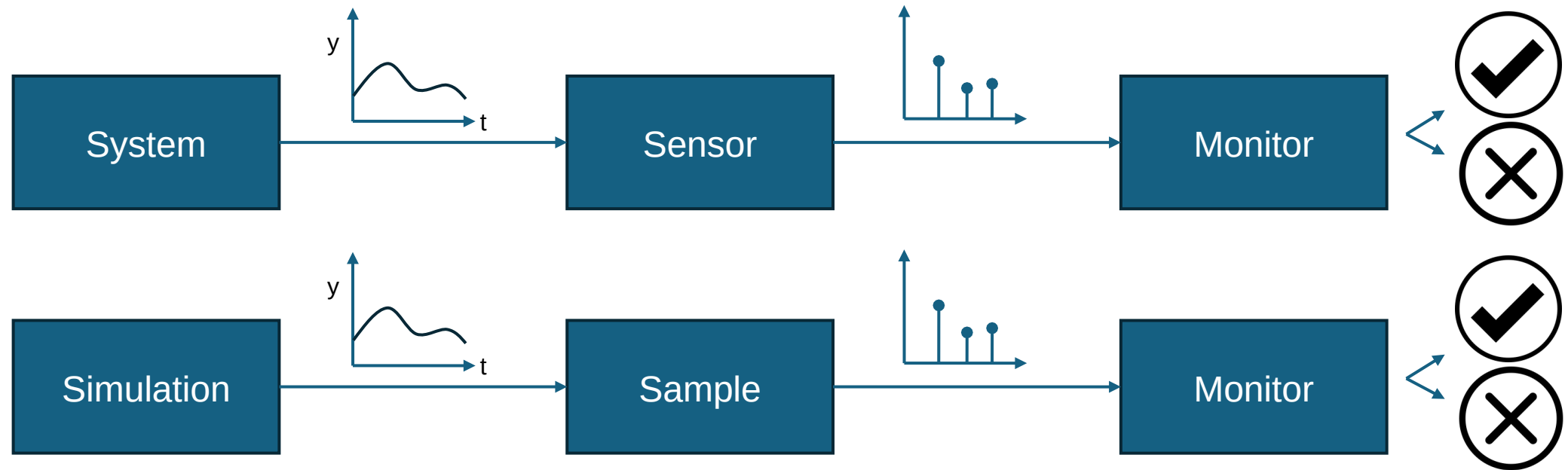
Guarantees that
there is progress

E.g. Starvation free

Each waiting container
shall eventually undergo
processing.

What is a monitor?

A **monitor** is a computational component that **observes** the **behaviour trace** of a system (real or simulated) to **determine** if it **satisfies** a specified **property**.



“offline” vs. **“online”** testing/monitoring

How can we monitor?

Some properties are just a check on every **value** that is sensed/sampled, **at each time**.

- Example: *The hoist height of the crane can never be higher than 20 meters*

→ Propositional Logic

Propositional variables: evaluate to either True or False

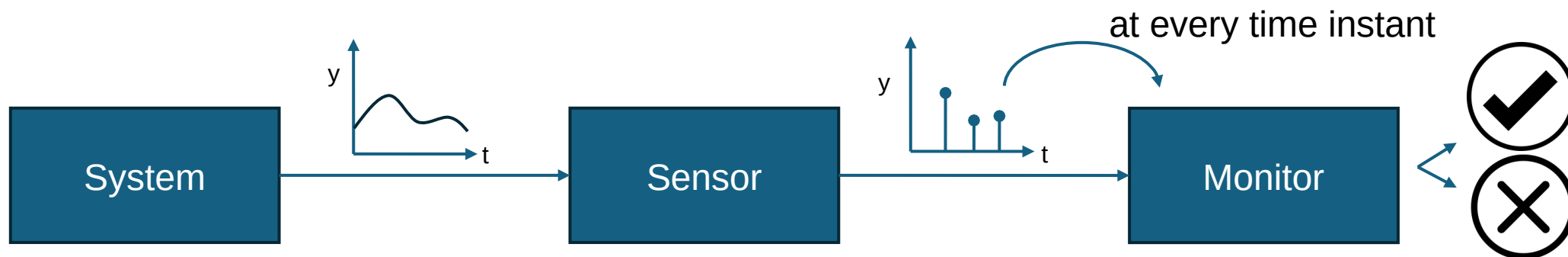
- $P_{\text{hoist_height_acceptable}}$: Hoist-height ≤ 20 [meter]
- monitor: check at every time that $P_{\text{hoist_height_acceptable}}$ holds

What's in a name (in logic)?

A **predicate** is an **expression or statement** containing **variables** that describe a **property** of an **object** or a **relationship between** objects, becoming either **true or false** when specific **values are assigned**.

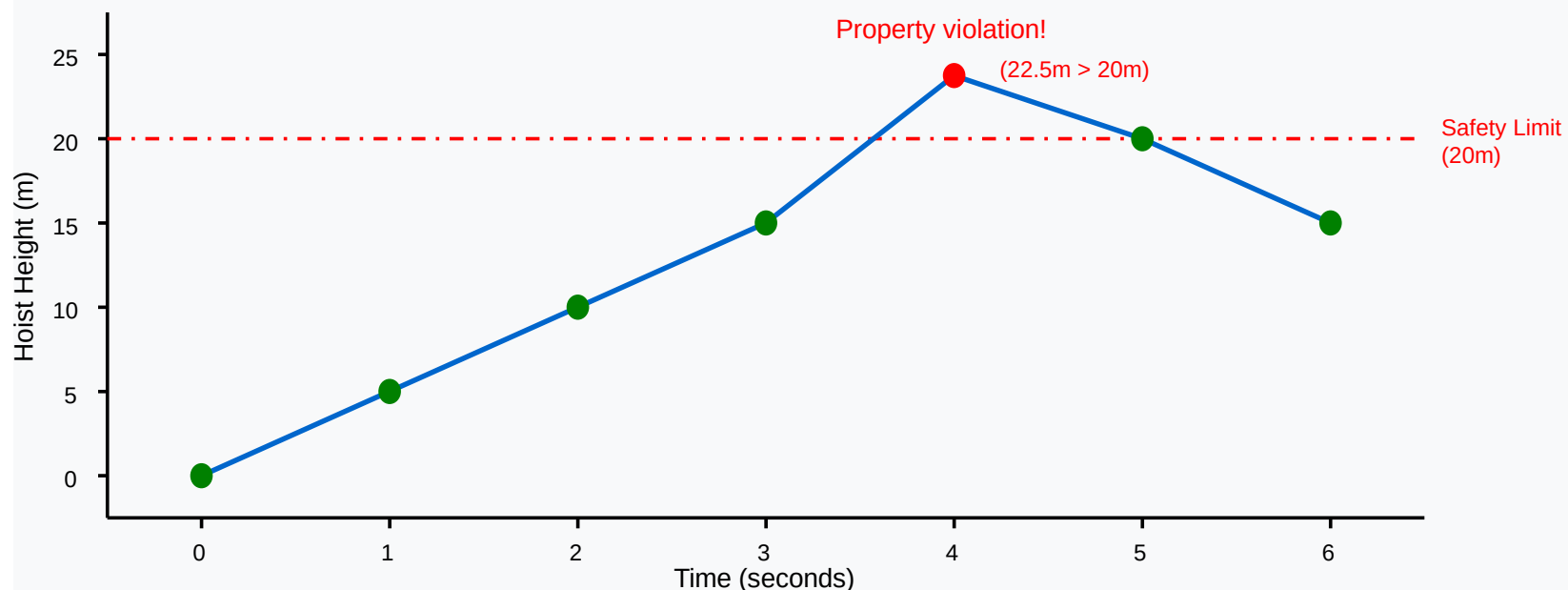
A **proposition** is a **declarative statement** that is definitely either true or false, but never both (the “excluded middle/third” principle in classical logics such as propositional and first-order logic).

Monitoring using Propositional Logic



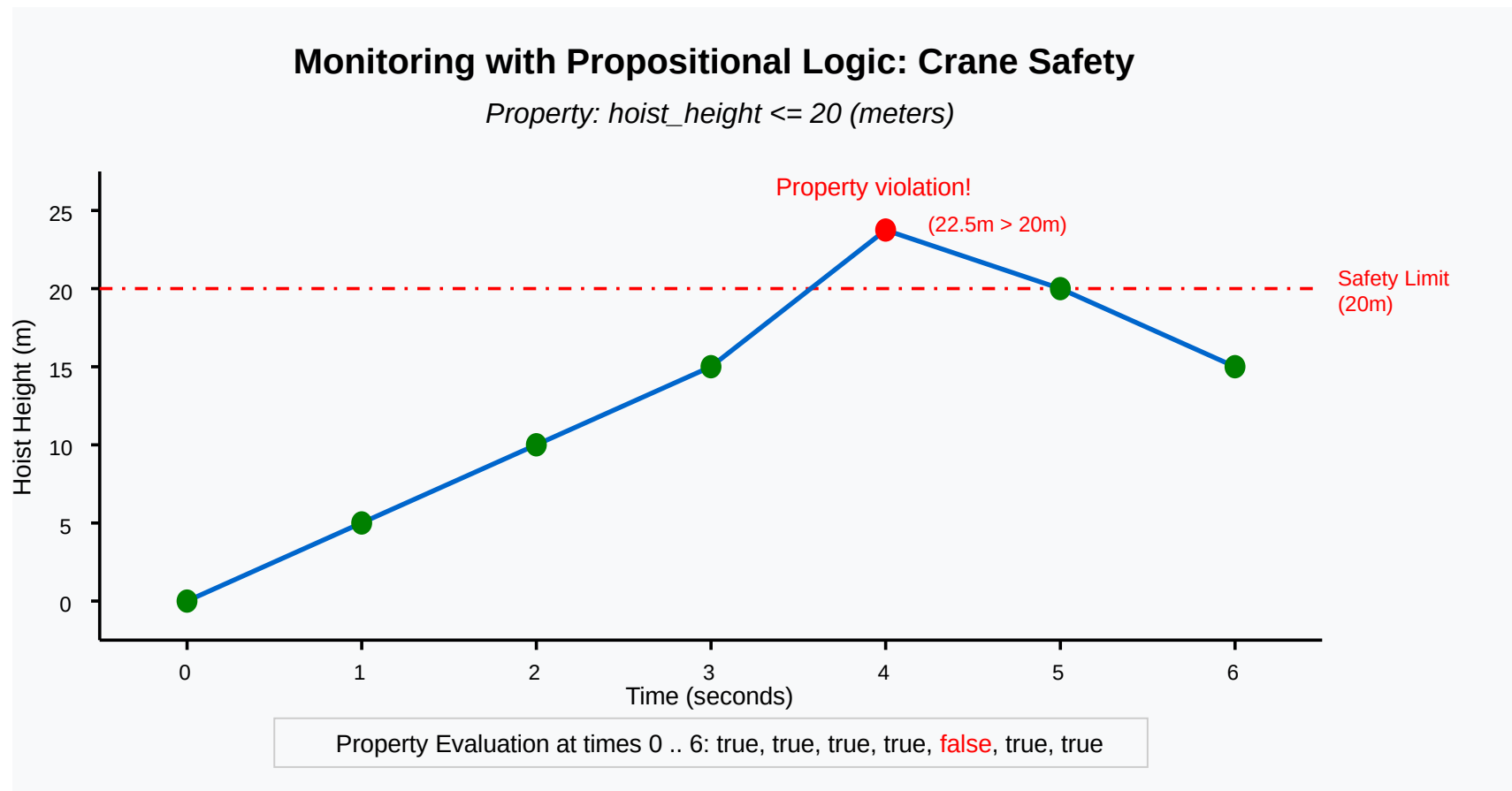
Monitoring with Propositional Logic: Crane Safety

Property: $\text{hoist_height} \leq 20$ (meters)



Property Evaluation at times 0 .. 6: true, true, true, true, false, true, true

“weak” notion of time, really only an ordered series of values



Propositional Logic

*Property: The hoist height of the crane can never be higher than 20 meters **and** has an absolute horizontal velocity less or equal than 8 m/s*

Propositional variables:

- P_hoist_height_acceptable: Hoist-height ≤ 20 [meter]
- P_horizontal_velocity_acceptable: $|\text{Horizontal_velocity}| \leq 8$ [m/s]

→ create complex properties with logical operators:

Property:

P_hoist_height_acceptable $\wedge$

P_horizontal_velocity_acceptable

Note: $p \rightarrow q$ is equivalent to $\text{not } p \text{ or } q$

<u>Formal Name</u>	<u>Nickname</u>	<u>Arity</u>	<u>Symbol</u>
Negation operator	NOT	Unary	$\neg$
Conjunction operator	AND	Binary	$\wedge$
Disjunction operator	OR	Binary	$\vee$
Exclusive-OR operator	XOR	Binary	$\oplus$
Implication operator	IMPLIES	Binary	$\rightarrow$
Biconditional operator	IFF	Binary	$\leftrightarrow$

How can I monitor?

- Some properties are just a check on every value that is sensed from the real system
 - Example: *The hoist height of the crane can never be higher than 20 meters*
- But what about **temporal orderings** of events?
 - *If the crane picks up a load, then, **within 5 seconds**, the crane must start moving to the designated drop zone and reach it within 1 minute.*
 - *After an emergency stop button is pressed, the power to the motors must be cut off in the **next** state, and the brakes must be engaged.*
- Much more difficult.
- Temporal logics are appropriate.

Decomposing a formula

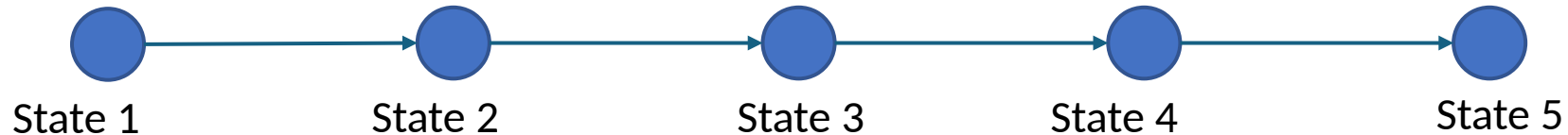
If the emergency button is pressed, then in the next state, motors must be off and brakes engaged.

Propositions in this formula:

- An emergency stop button is pressed: P_emergency_stop
- The power to the motors is cut off: P_power_off
- The brakes are engaged: P_brakes_engaged

Time abstraction...

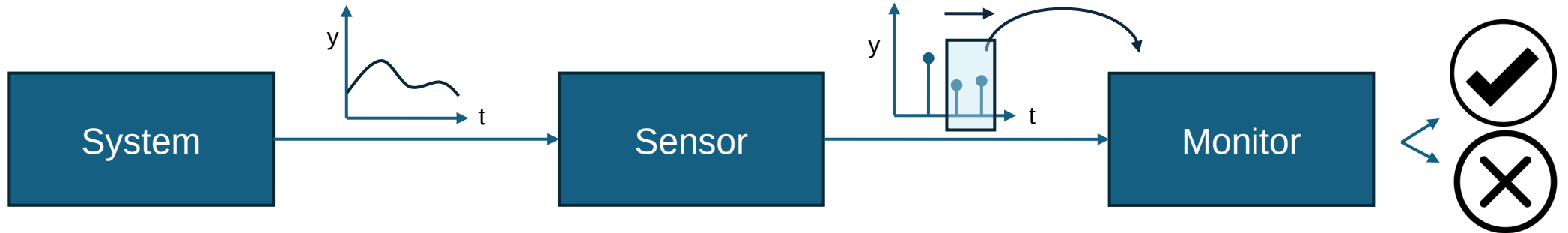
Look at time as an **ordering**
of **snapshots** of subsequent states a system is in



“weak” notion of time: *next state*

How could we program this?

2 samples are needed
→ need a **sliding window**
data structure



```
monitor (P_emergency_stop, P_power_off, P_brakes_engaged):  
    if (P_emergency_stop[0] == true):  
        if (P_power_off[1] == true and P_brakes_engaged[1] == true):  
            return true  
        else:  
            return false  
        endif  
    else:  
        return true  
    endif
```

HOW (to check)!

LTL (Linear Temporal Logic)

more appropriate (cognitive and performance): Temporal Logic

Linear Temporal Logic: along a single path of states (behaviour trace)

Each state is defined by a unique assignment of propositional variables

$$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi_1 \mathbf{U} \varphi_2$$

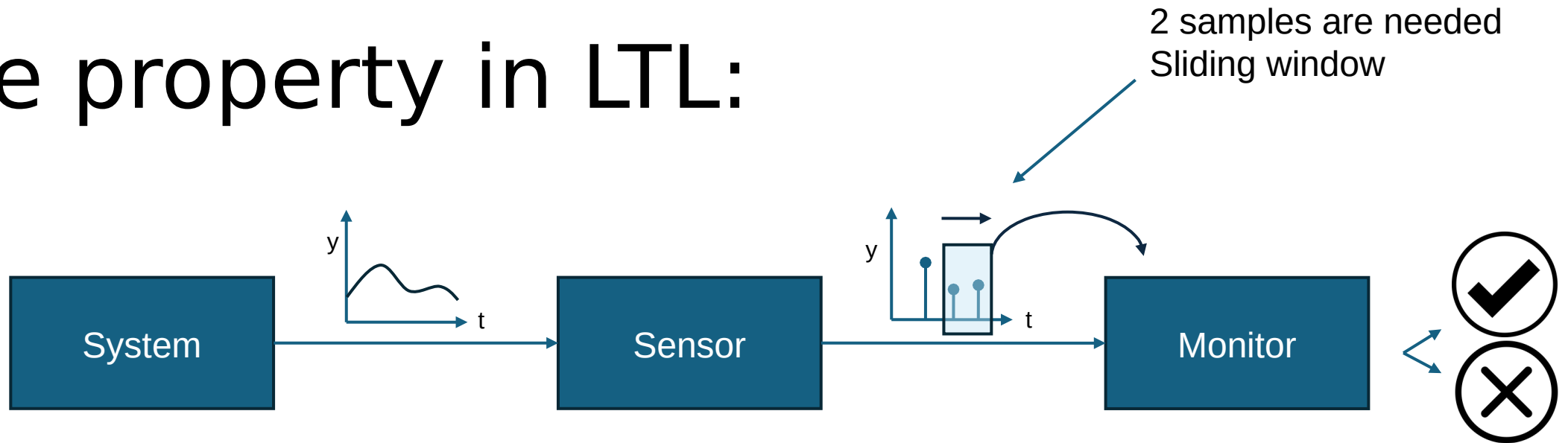
a = atomic proposition (e.g., $P_{\text{emergency_stop}}$, $P_{\text{motor_off}}$, etc.)

With temporal operators, it assumes a sequence of states

$\bigcirc$ = next (sometimes $X \varphi$): property φ holds in the next state

$\mathbf{U}$ = Until ($\varphi_1 \mathbf{U} \varphi_2$): property φ_1 holds until φ_2 becomes true

The property in LTL:



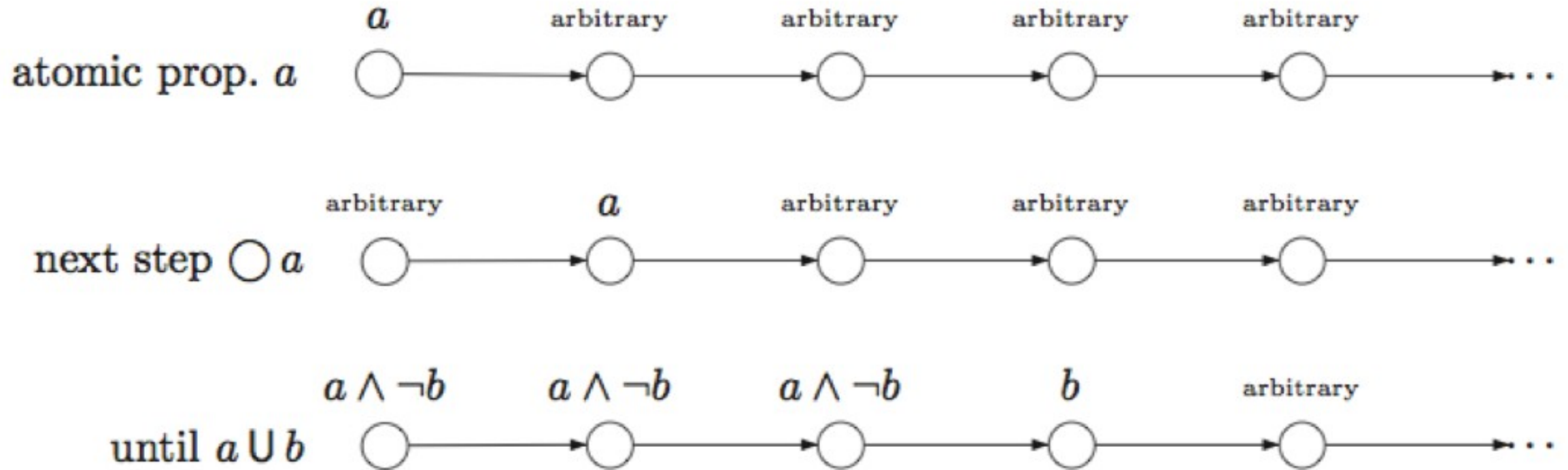
Specification: $P_emergency_stop \rightarrow X(P_power_off \wedge P_brake_engaged)$

More appropriate because:

- Declarative: Specifies **what** should happen, not **how** to check it
- Formal: Precise meaning
- Complex properties from simple ones

WHAT!

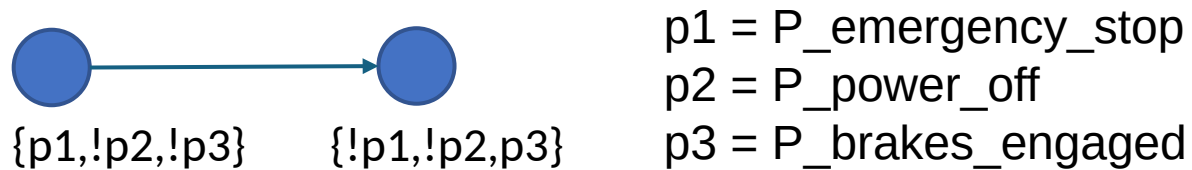
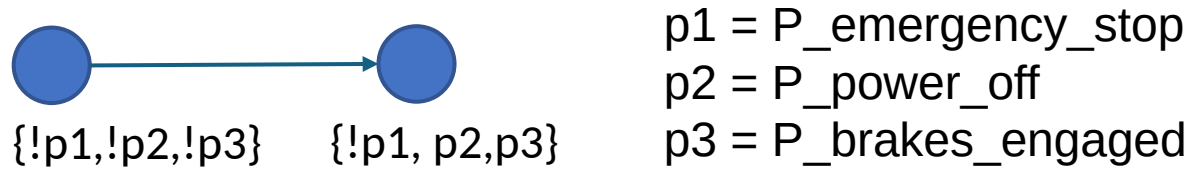
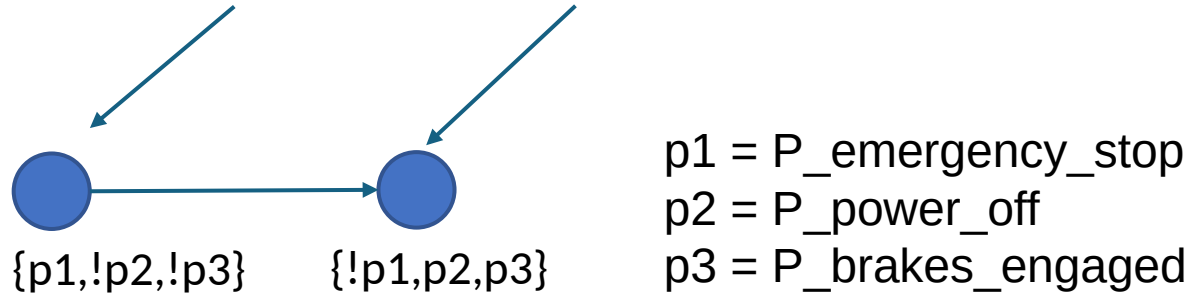
meaning of a formula (summary)



$\sigma \models \varphi$ (path “satisfies” specification)

Satisfaction of a formula

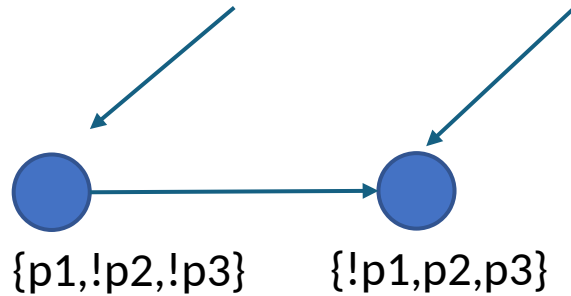
$P_emergency_stop \rightarrow X(P_power_off \wedge P_brake_engaged)$



$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi_1 \mathbf{U} \varphi_2$

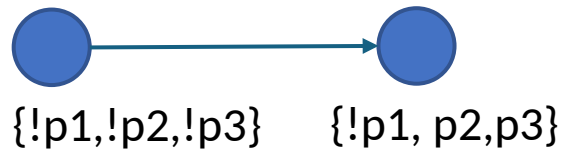
Satisfaction of a formula

$P_emergency_stop \rightarrow X(P_power_off \wedge P_brake_engaged)$



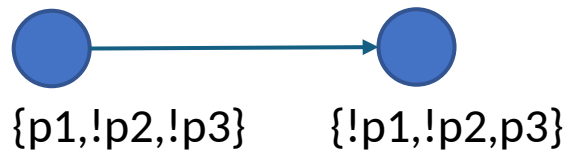
$p1 = P_emergency_stop$
 $p2 = P_power_off$
 $p3 = P_brakes_engaged$

True



$p1 = P_emergency_stop$
 $p2 = P_power_off$
 $p3 = P_brakes_engaged$

True



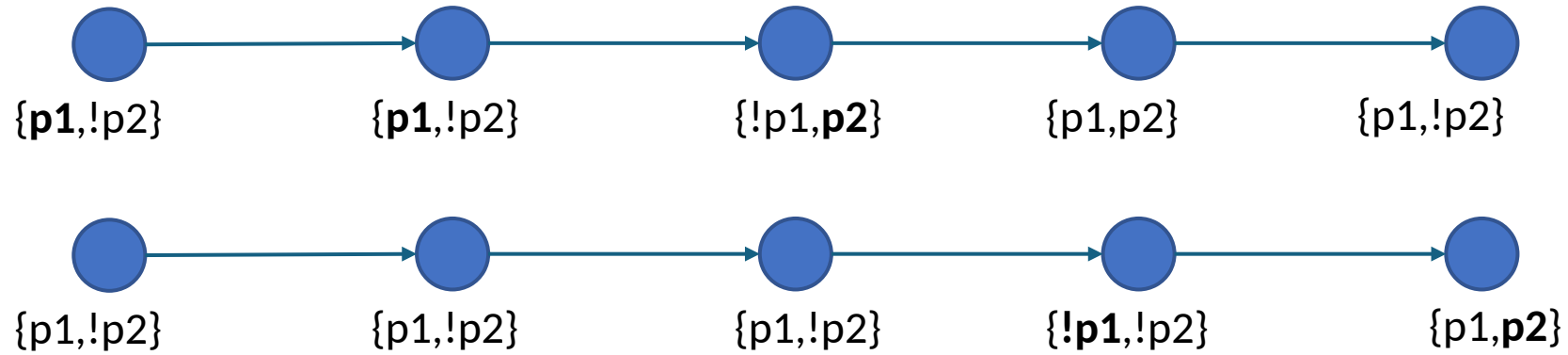
$p1 = P_emergency_stop$
 $p2 = P_power_off$
 $p3 = P_brakes_engaged$

False

$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi_1 \cup \varphi_2$

Satisfaction of a formula

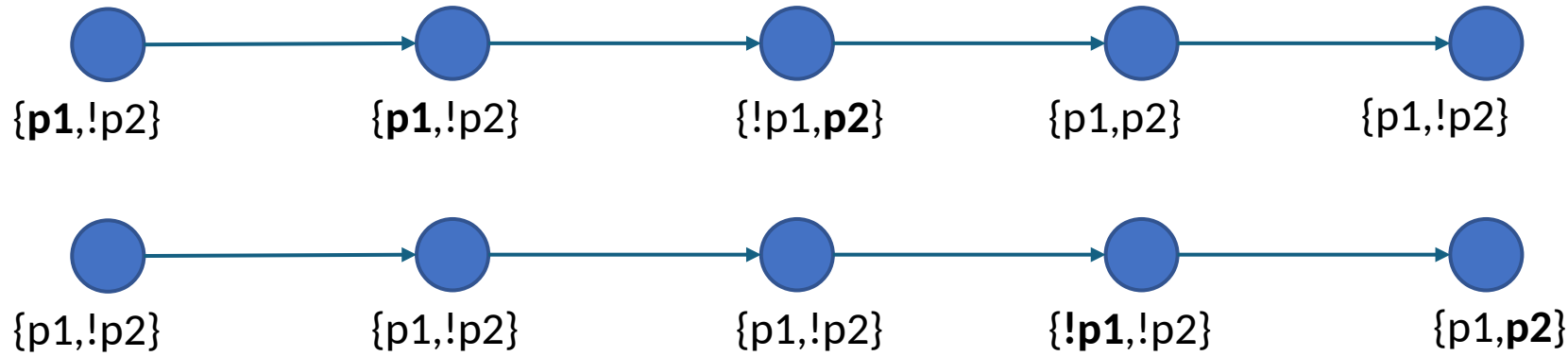
$p1 \text{ U } p2$



$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi_1 \text{ U } \varphi_2$

Satisfaction of a formula

$p1 \text{ U } p2$



True

False

$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi_1 \text{ U } \varphi_2$

True U p1

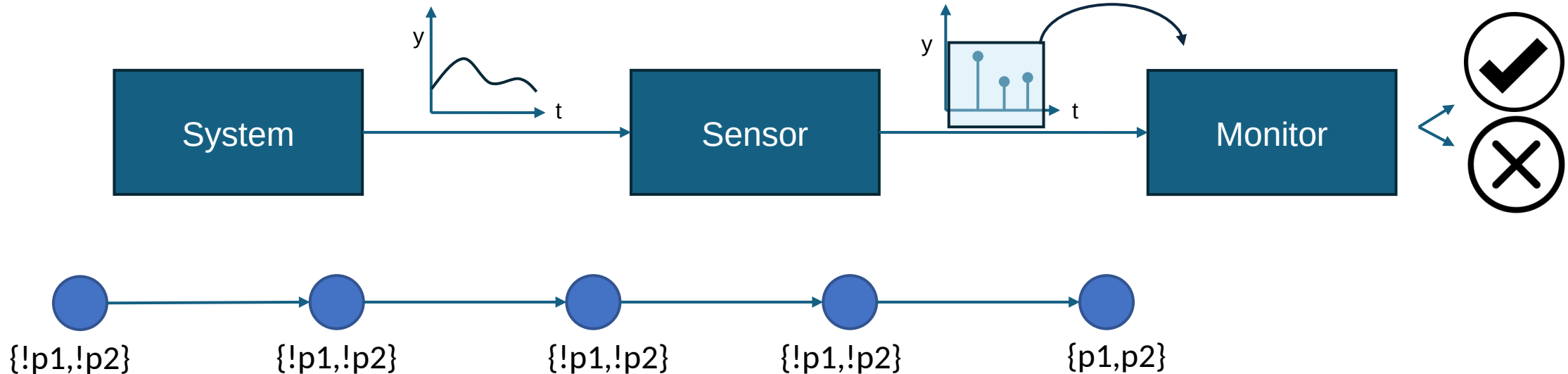
Property: by the end of each operational cycle, the crane must **eventually** come to a complete stop

$p1 = P_horizontal_no_velocity$

$p2 = P_vertical_no_velocity$

$true\ U\ (p1/\wedge p2)$

Trace of the operational cycle is needed



This is often needed $\rightarrow$ (syntactic sugar) operator: F or $<>$ “in the Fullness of time”, “eFentially”

True U p1

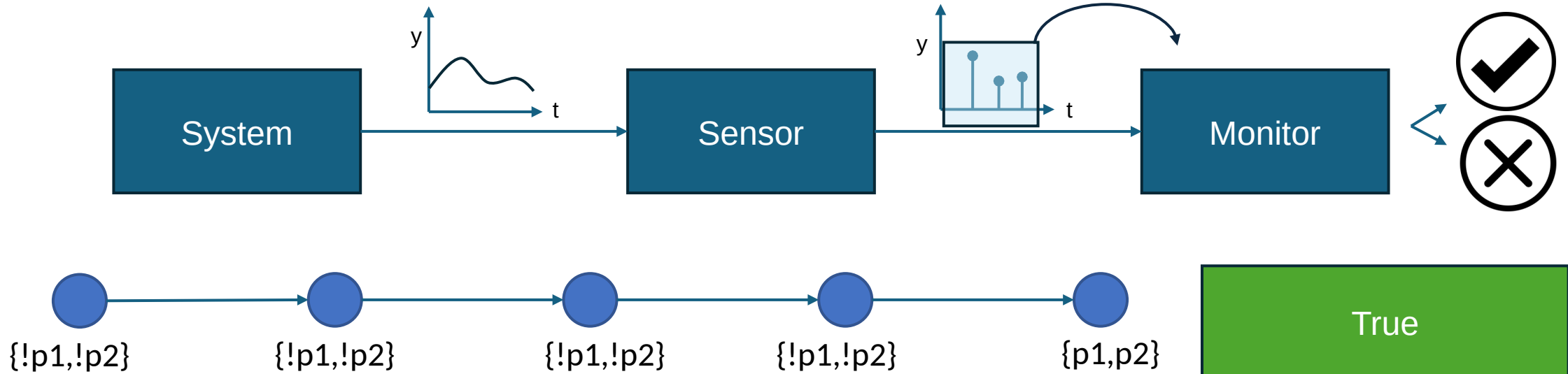
Property: by the end of each operational cycle, the crane must **eventually** come to a complete stop

$p1 = P_horizontal_no_velocity$

$p2 = P_vertical_no_velocity$

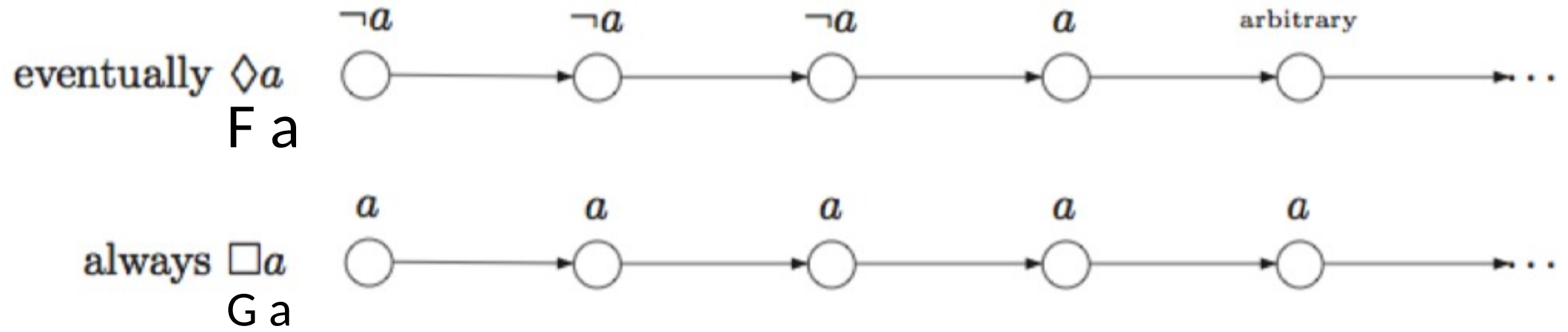
$true\ U\ (p1/\backslash p2)$

Trace of the operational cycle is needed



This is often needed $\rightarrow$ (syntactic sugar) operator: F or $\langle \rangle$ “in the Fullness of time”, “eventually”

Extra operators (summary)



$p \text{ W } q$

(syntactic sugar) operator: **weak until** W

$$\begin{aligned} p \text{ W } q &= (G \text{ } p) \mid (p \text{ U } q) \\ &= p \text{ U } (q \mid Gp) \end{aligned}$$

Binding

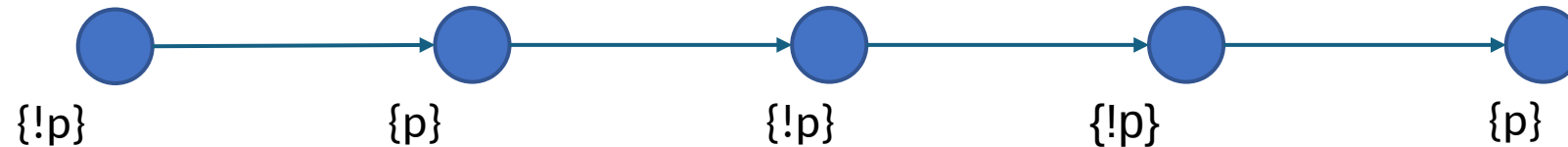
- **Operator precedence**

- Unary binds stronger than binary
- Bind from right to left:
- $\square \diamond p = (\square(\diamond p))$
- $p \cup q \cup r = p \cup (q \cup r)$
- $\cup$ takes precedence over $\wedge$, $\vee$ and $\rightarrow$

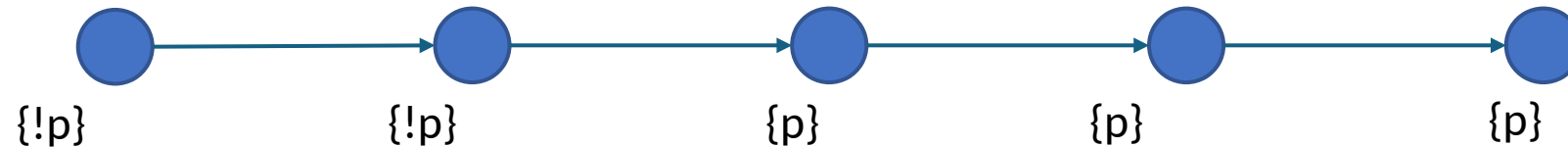
Dwyer Patterns

<https://matthewbdwyer.github.io/psp/patterns/ltl.html>

- What is the meaning of “G F p” ?



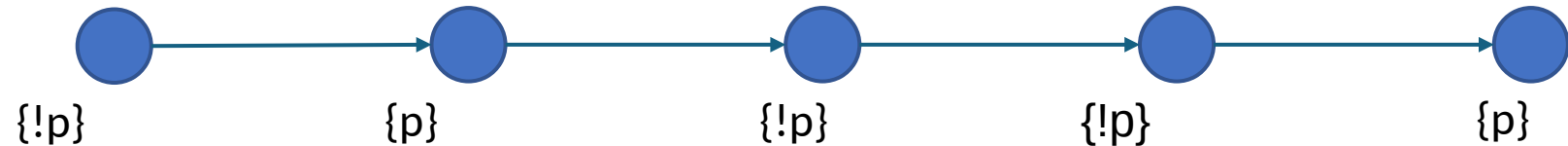
- What is the meaning of “F G p” ?



Dwyer Patterns

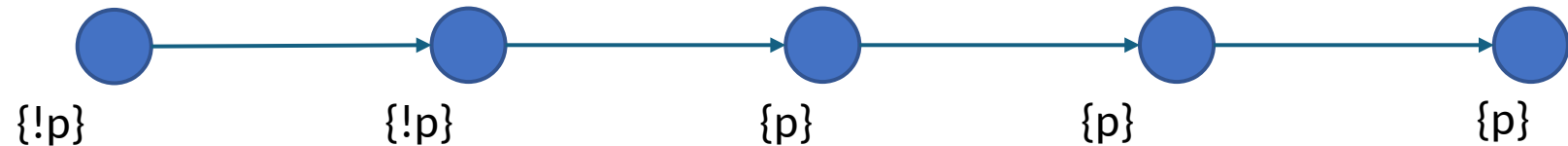
<https://matthewbdwyer.github.io/psp/patterns/ltl.html>

- What is the meaning of “G F p” ?



True

- What is the meaning of “F G p” ?



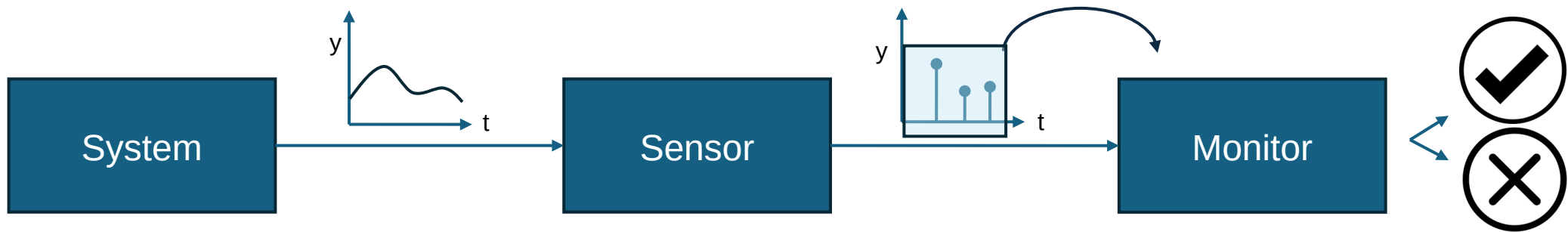
True

Examples

- *When carrying loads heavier than 5000kg, the crane's velocity must always be less than 2 m/s*
 - Propositions needed:
 - P_load_weight: load_weight > 5000
 - P_velocity: velocity < 2
 - $G((P_load_weight) \rightarrow (P_velocity))$
- *After initiating a grab, movement must not begin until the load is secured.*
 - Propositions needed:
 - P_grab_initiated: grab_command == 1
 - P_no_move: velocity < 0.001
 - P_secured: load_sensor > 100 && gripper_pressure > 50
 - $G(P_grab_initiated \rightarrow (P_no_move \cup P_secured))$

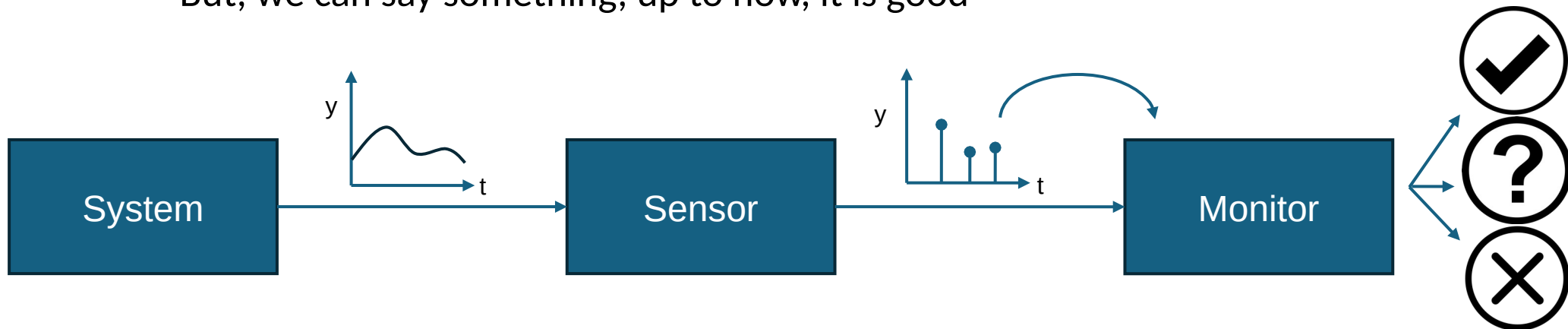
Monitoring Bounded Traces

- What we have shown up to now were properties over **bounded** traces
- Verification **window** corresponds to a natural operational cycle
 - For example, one pick-up/move/drop-off of the crane system
- Binary verdict (True or False) as the trace is **complete**



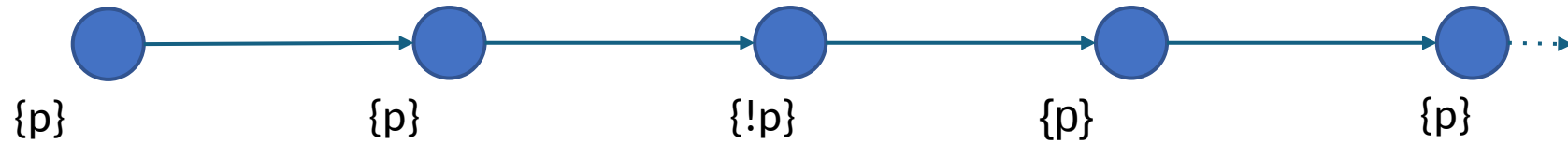
Monitoring Unbounded Traces

- We give the monitor a sample each timestep
- Requires reasoning about potentially **infinite future** behaviours
- Three possibilities:
 - Property satisfied over the finite trace, independent of future
 - Property falsified over the final trace, for every continuation
 - Finite trace still allows for falsification or satisfaction of the property, but **so far so good**
- Needs three-valued logic (True, False, **Inconclusive**)
- Watch out: some properties are non-monitorable
 - $G(r \rightarrow Fa)$: No continuation will ever lead to satisfaction or falsification.
 - But, we can say something, up to now, it is good

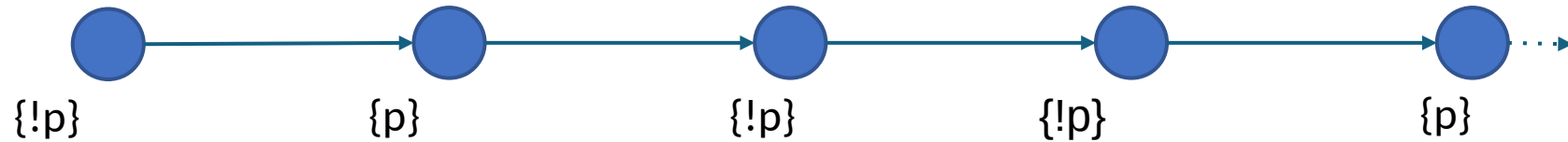


Monitoring Unbounded Traces

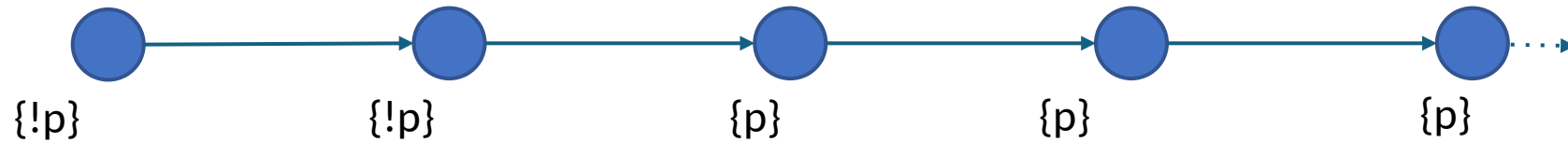
$G \ p$



$G \ F \ p$

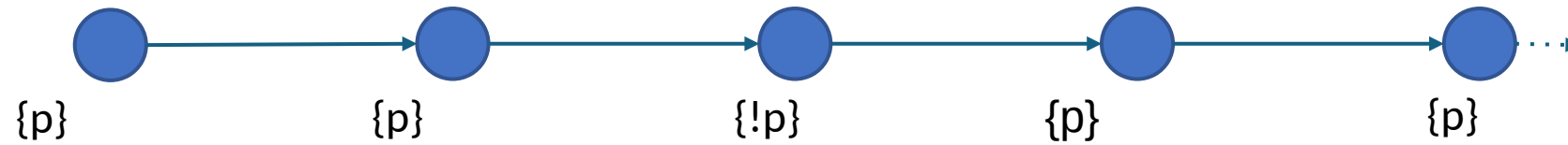


$F \ G \ p$



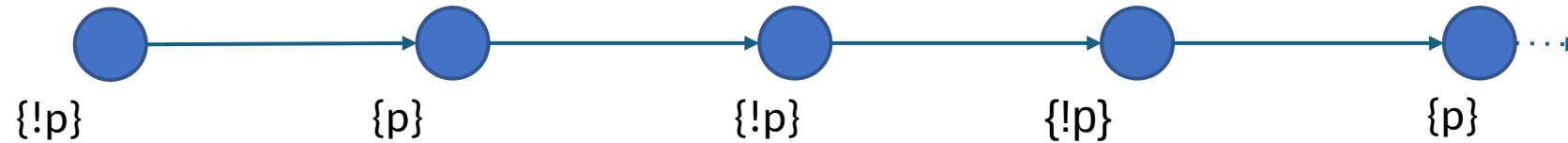
Monitoring Unbounded Traces

$G\ p$



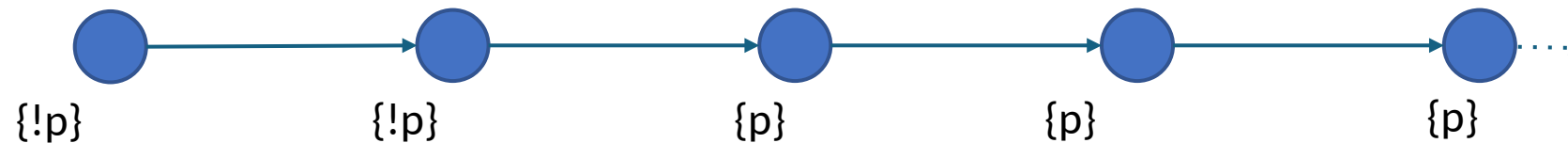
False

$G\ F\ p$



Inconclusive

$F\ G\ p$



Inconclusive

Towards STL (**S**ignal Temporal Logic)

*If the crane picks up a load, then **within 5 seconds**, the crane must start moving to the designated drop zone*

LTL $G(P_pick_up \rightarrow F P_velocity)$

Boolean predicates, weak notion of time: inadequate for our purpose

MTL (Metric (Interval) Temporal Logic) (metric = distance metric)

$G(P_pick_up \rightarrow F_{[0,5]s} P_velocity)$

Boolean predicates, real-valued timebase

STL $G((load_sensor > 100 \ \&\& \ gripper_pressure > 50) \rightarrow F_{[0,5]s} velocity > 0)$

Predicates over real values, real-valued time

++ **robustness**, how far am I **off from** violating the property?

Examples

The signal is never above 3.5

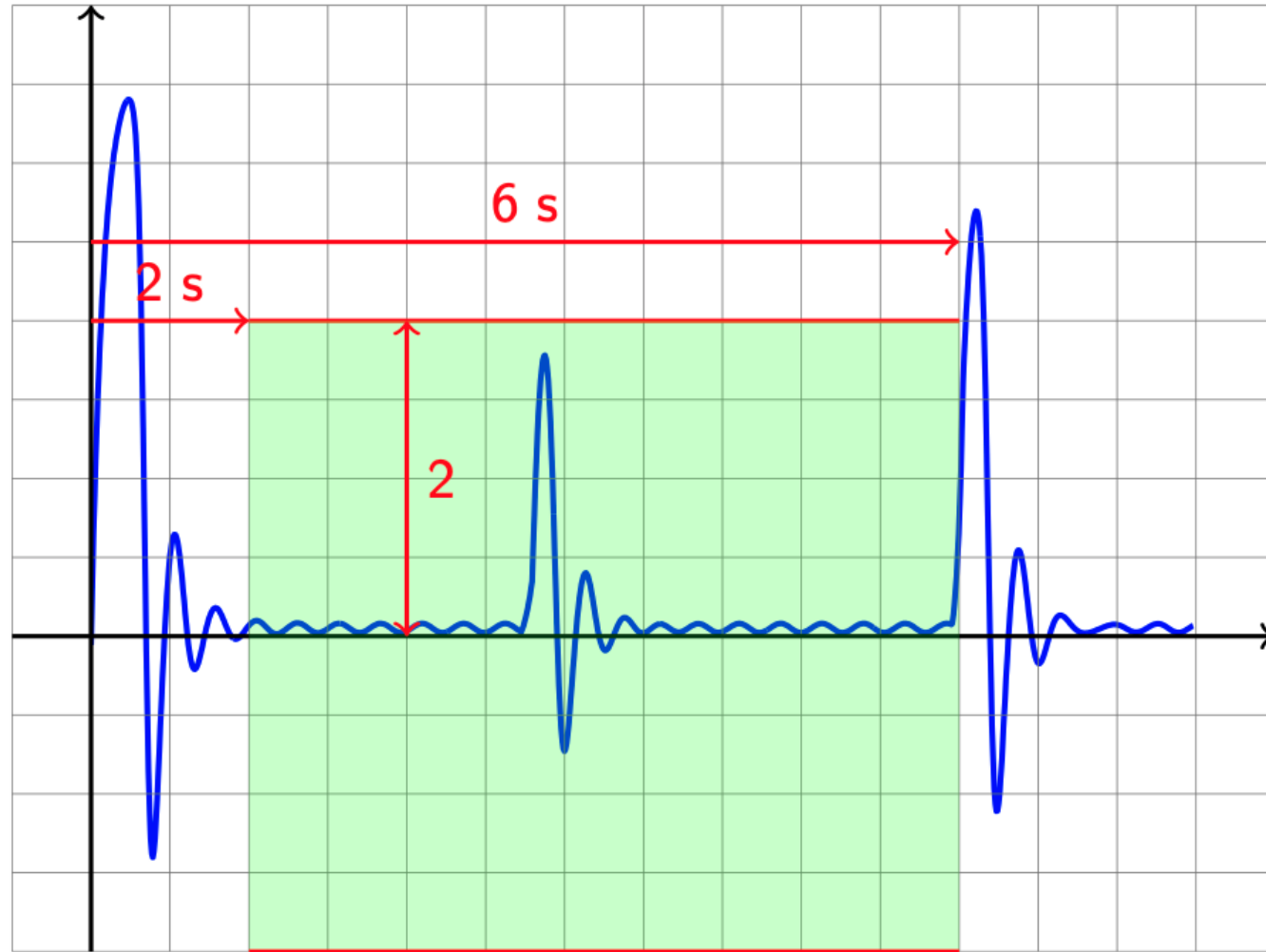
$$\varphi := G (x[t] < 3.5)$$



Examples

Between 2s and 6s the signal is between -2 and 2

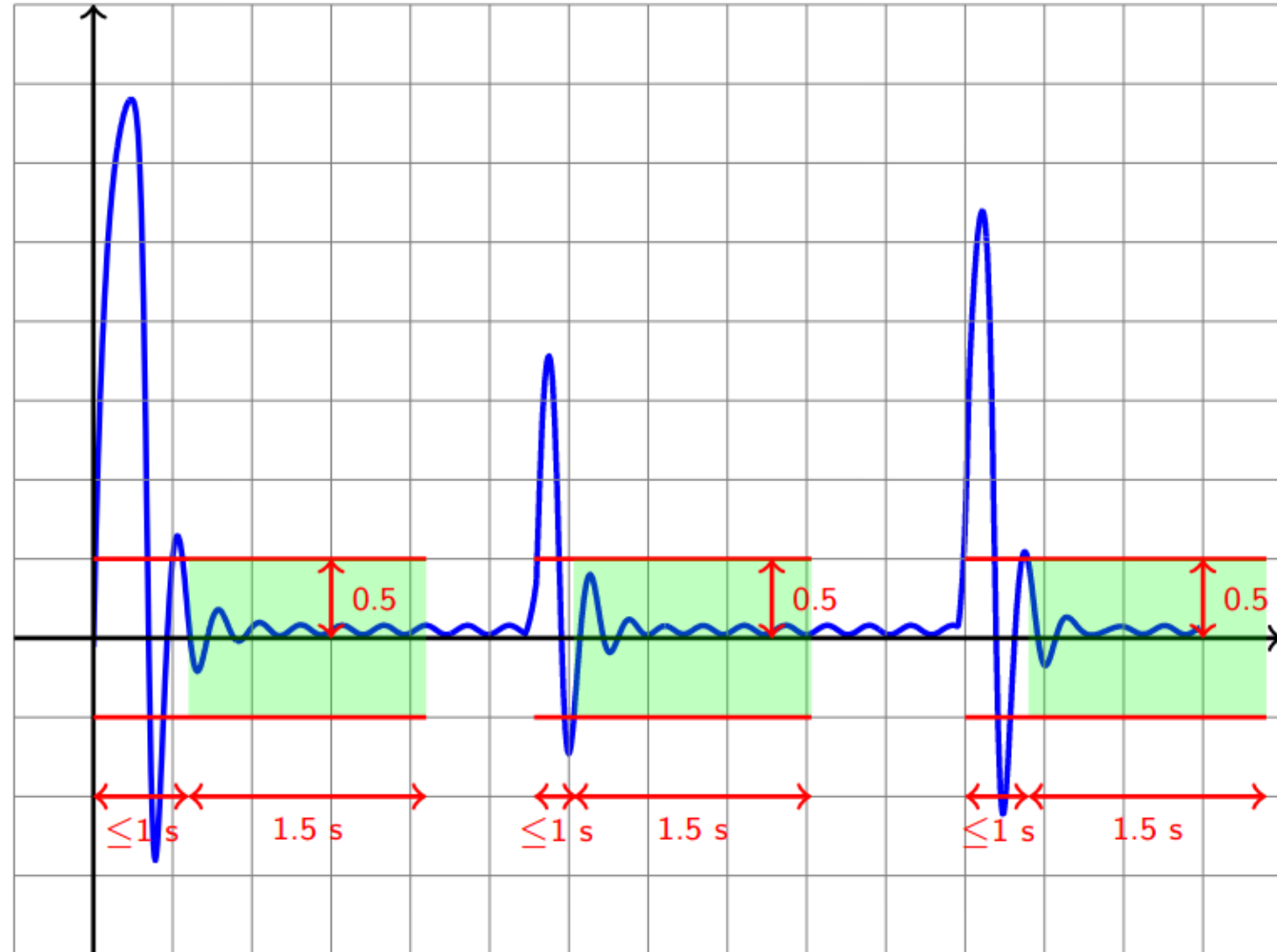
$$\varphi := G_{[2,6]} (|x[t]| < 2)$$



Examples

Always $|x| > 0.5 \Rightarrow$ after 1 s, $|x|$ settles under 0.5 for 1.5 s

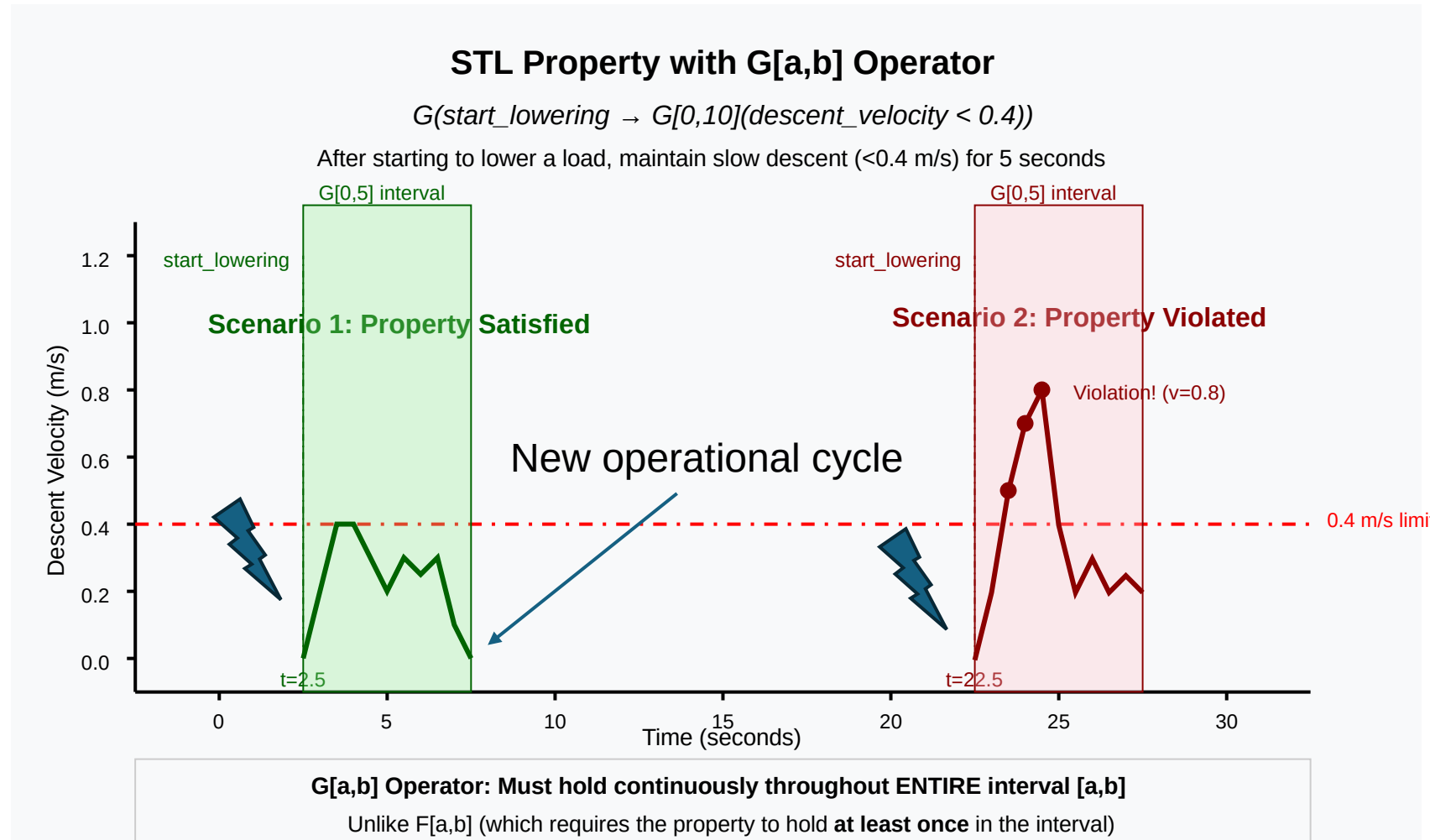
$$\varphi := G(x[t] > .5 \rightarrow F_{[0,1]} (G_{[0,1.5]} x[t] < 0.5))$$



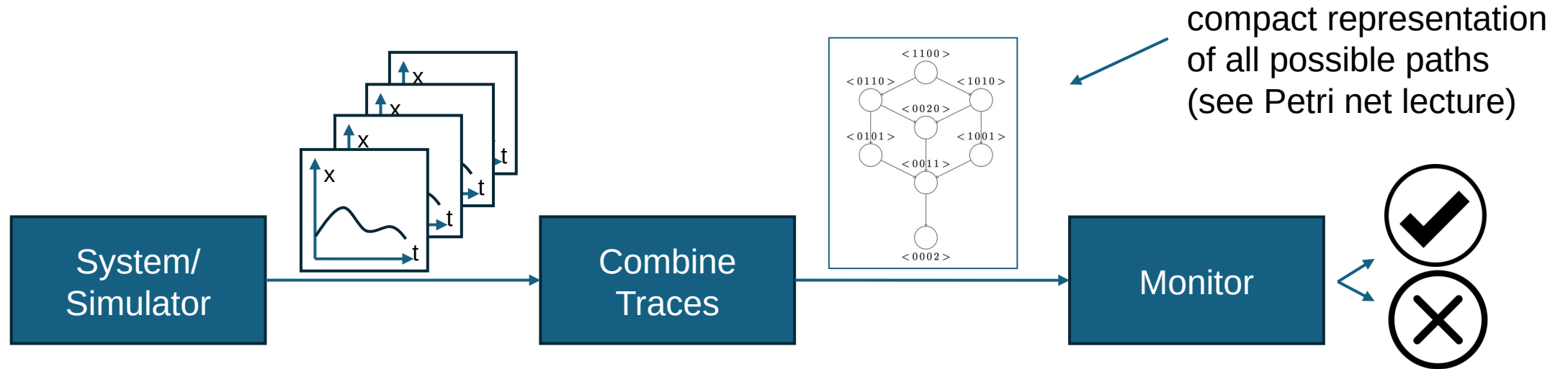
Examples

After the crane starts lowering a load, it must maintain a slow descent speed (less than 0.4 m/s) continuously for at least 5 seconds before touching down.

$G(\text{start_lowering} \rightarrow G[0,5](\text{descent_velocity} < 0.4))$



What if we have nondeterminism?



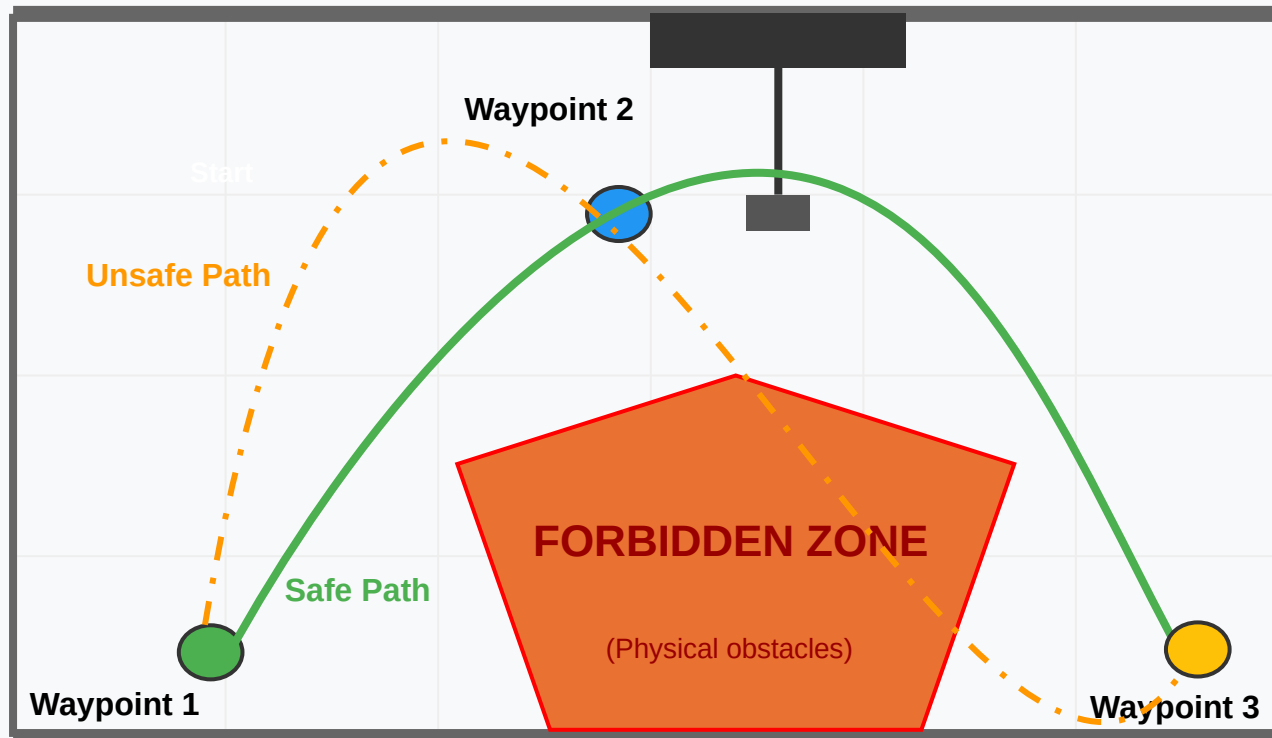
Not a single trace, but a **set of possible traces**

→ can make statements about about behavioural properties
under all possible circumstances

What if we have nondeterminism?

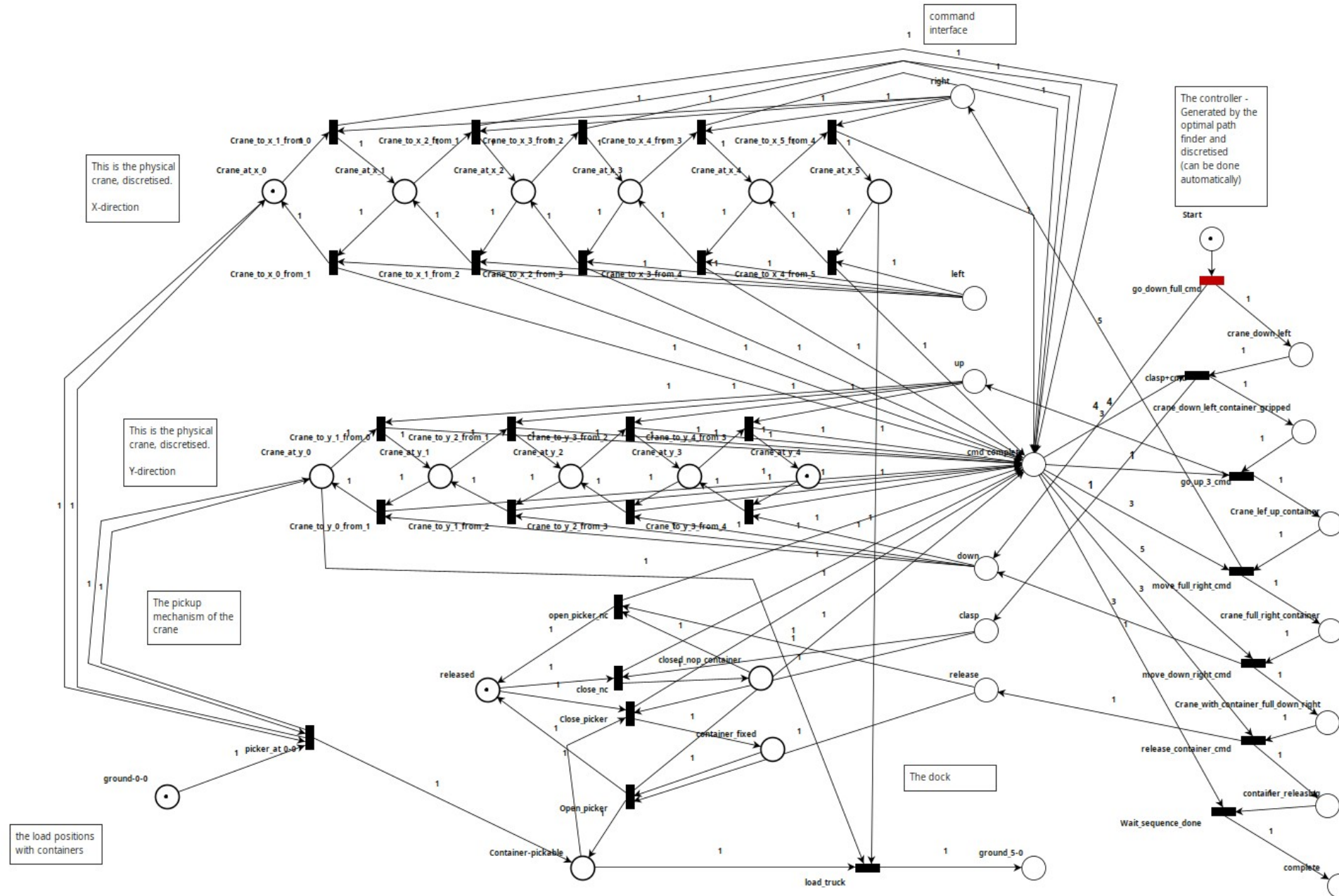
Model Checking for Gantry Crane Path Verification

Verifying that no possible execution will enter the forbidden zone



*"The crane will never enter the forbidden zone **on any possible behaviour path**"*

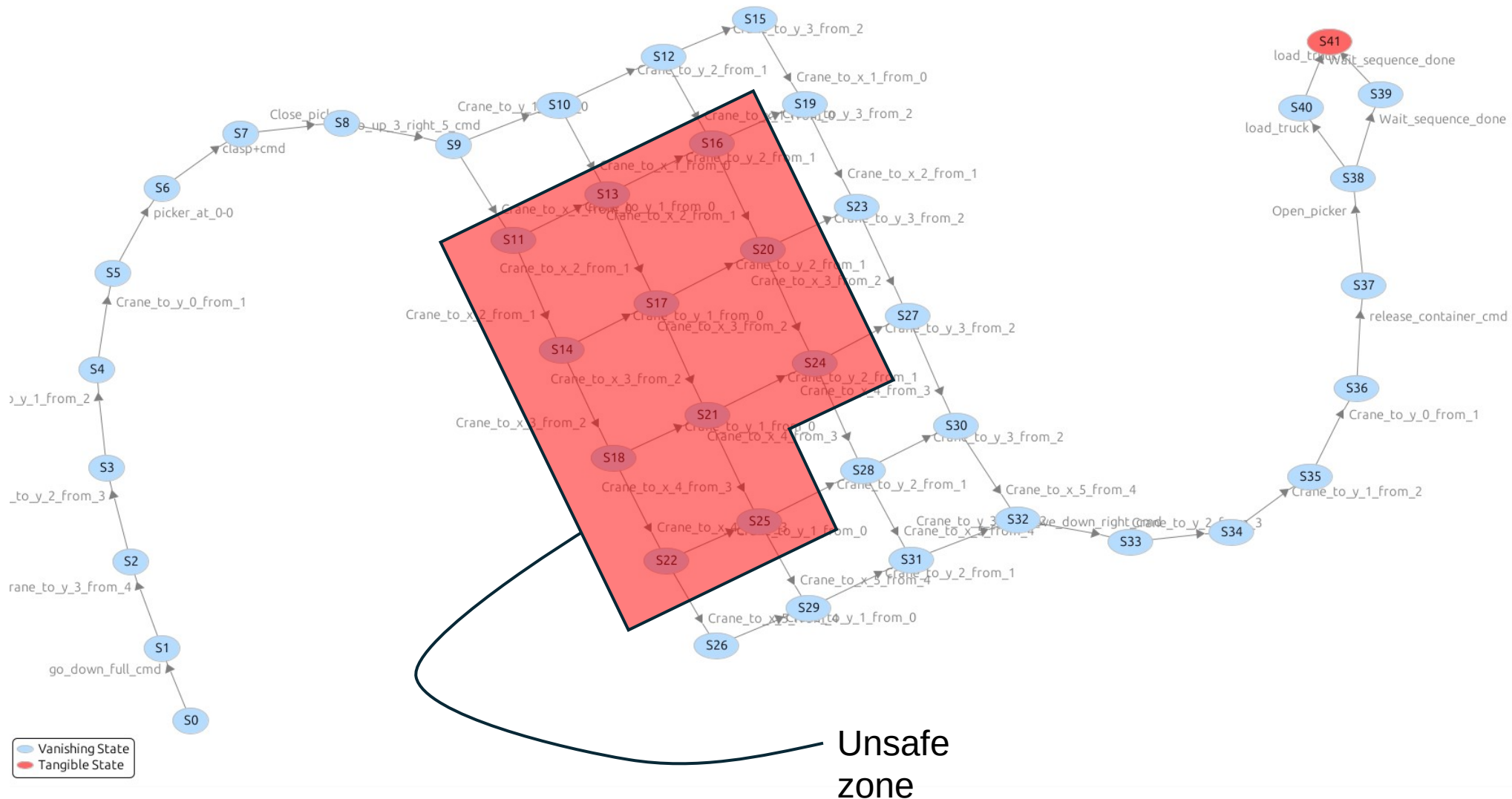
Petri net model of (discretized) Gantry Crane behaviour



All possible paths:



All containers end at the correct location



some paths are not safe → refine the software controller

tools

TUTOR DASHBOARDLTL SyntaxGenerate ExerciseExperiments ▾ProfileLog Out

Exercise garden-chair-3095 · ClassicQuestion 3 of 6

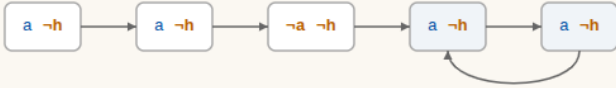
Which of the following traces satisfies the following LTL formula?

$((F\ h) \rightarrow (G\ a))$

☐

☐

☐

☐

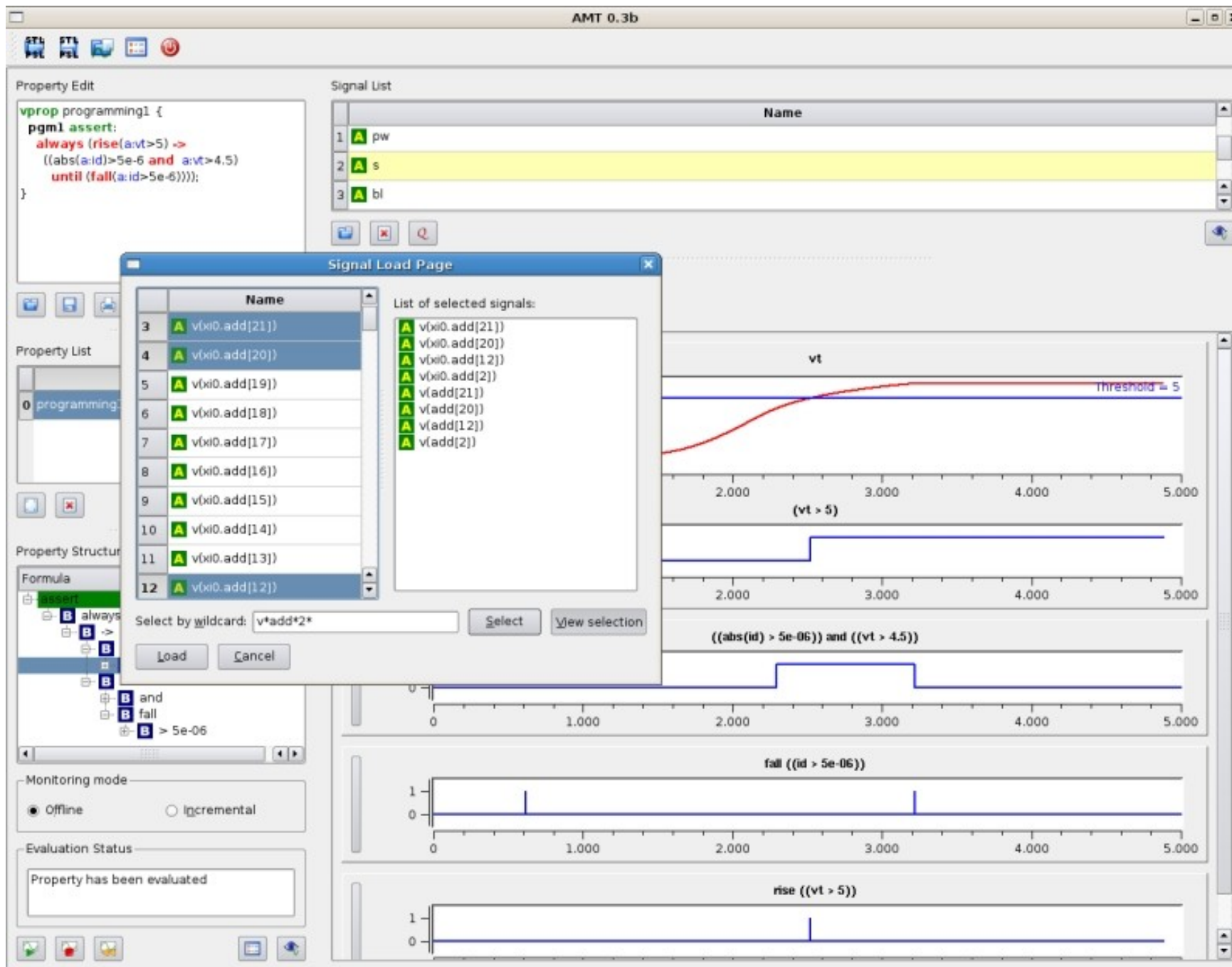
☐

Check Answer

Next Question

<https://ltl-tutor.xyz/>

tools



tools



[International Conference on Computer Aided Verification](#)

CAV 2010: [Computer Aided Verification](#) pp 167-170 | [Cite as](#)

Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems

Authors

[Authors and affiliations](#)

Alexandre Donzé

Conference paper

172

Citations

2.3k

Downloads

Part of the [Lecture Notes in Computer Science](#) book series (LNCS, volume 6174)

Abstract

We describe Breach, a Matlab/C++ toolbox providing a coherent set of simulation-based techniques aimed at the analysis of deterministic models of hybrid dynamical systems. The primary feature of Breach is to facilitate the computation and the property investigation of large sets of trajectories. It relies on an efficient numerical solver of ordinary differential equations that can also provide information about sensitivity with respect to parameters variation. The latter is used to perform approximate reachability analysis and parameter synthesis. A major novel feature is the robust monitoring of metric interval temporal logic (MITL) formulas. The application domain of Breach ranges from embedded systems design to

<https://github.com/decyphir/breach>

tools

mvcisback / py-metric-temporal-logic Public

- <> Code
- Issues 1
- Pull requests
- Actions
- Projects
- Security
- Insights

main 31 Branches 1 Tag Go to file <> Code

	chore: Bump dependencies.	ceb2567 · last year 363 Commits
assets	added Node typevar	6 years ago
mtl	feat: allow hyphens in AP names.	2 years ago
.drone.yml	chore: Bump dependencies.	last year
.gitignore	yapf + pylint + add style checks to tests	7 years ago
CITATION.cff	Create CITATION.cff	3 years ago
LICENSE	restructuring to move repo	8 years ago
README.md	Small typo	last year
poetry.lock	chore: Bump dependencies.	last year
pyproject.toml	chore: Bump dependencies.	last year

README
BSD-3-Clause license

metric-temporal-logic

A library for manipulating and evaluating metric temporal logic.

build failure
coverage 98%
package version 0.4.1
license MIT
DOI 10.5281/zenodo.2548862

Table of Contents

- About
- Installation
- Usage
 - Python Operator API
 - Propositional logic (using python syntax)
 - Modal Logic (using python syntax)

About

Python library for working with Metric Temporal Logic (MTL)

- Readme
- BSD-3-Clause license
- Cite this repository
- Activity
- 93 stars
- 6 watching
- 19 forks
- Report repository

Releases

1 tags

Packages

No packages published

Contributors

7 contributors

Languages

Python 100.0%

<https://github.com/mvcisback/py-metric-temporal-logic>