Verification and Validation (V&V) of Autonomous Robots

Dr. Michaela Klauck, Bosch Research, Germany

July 3rd, 2025

ACM SIGSOFT Summer School for Software Engineering in Robotics

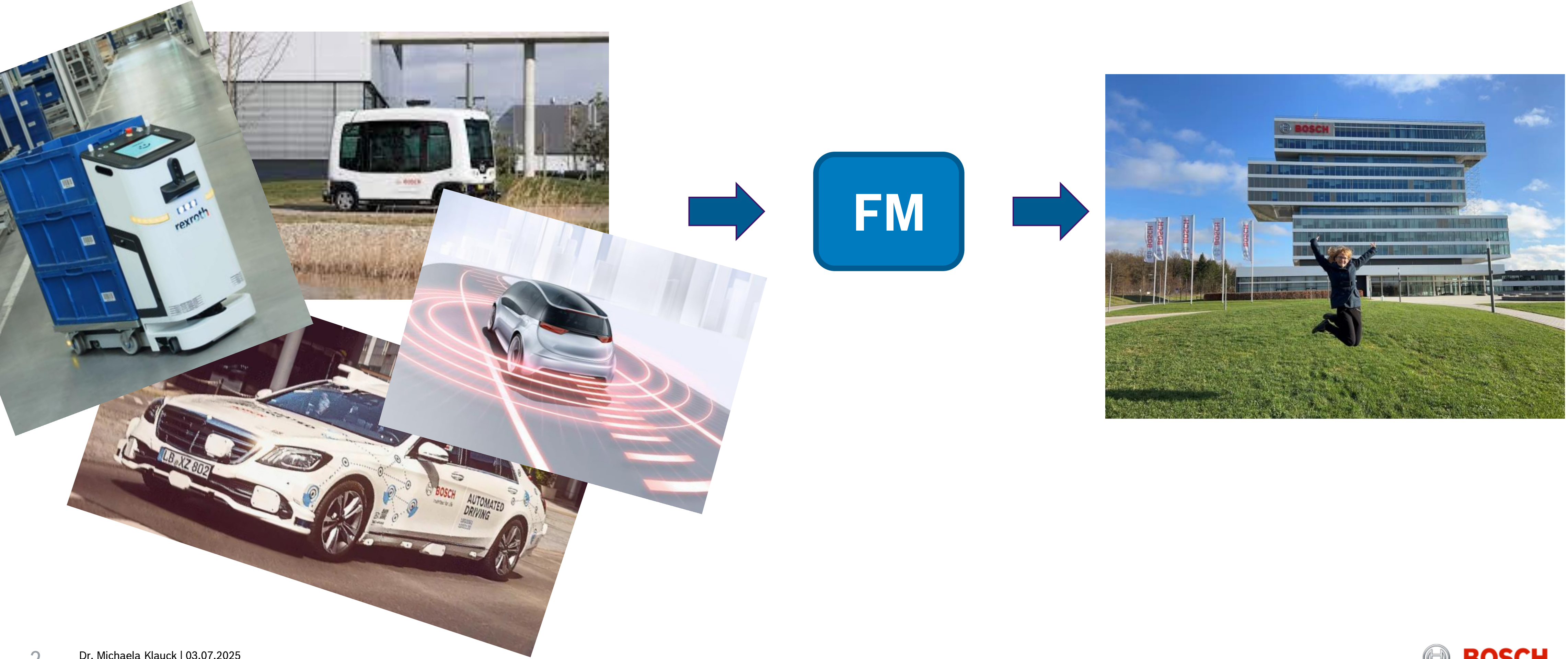
Delft, The Netherlands

CONVINCE

BOSCH

Bringing Formal Methods to Autonomous Systems Engineering

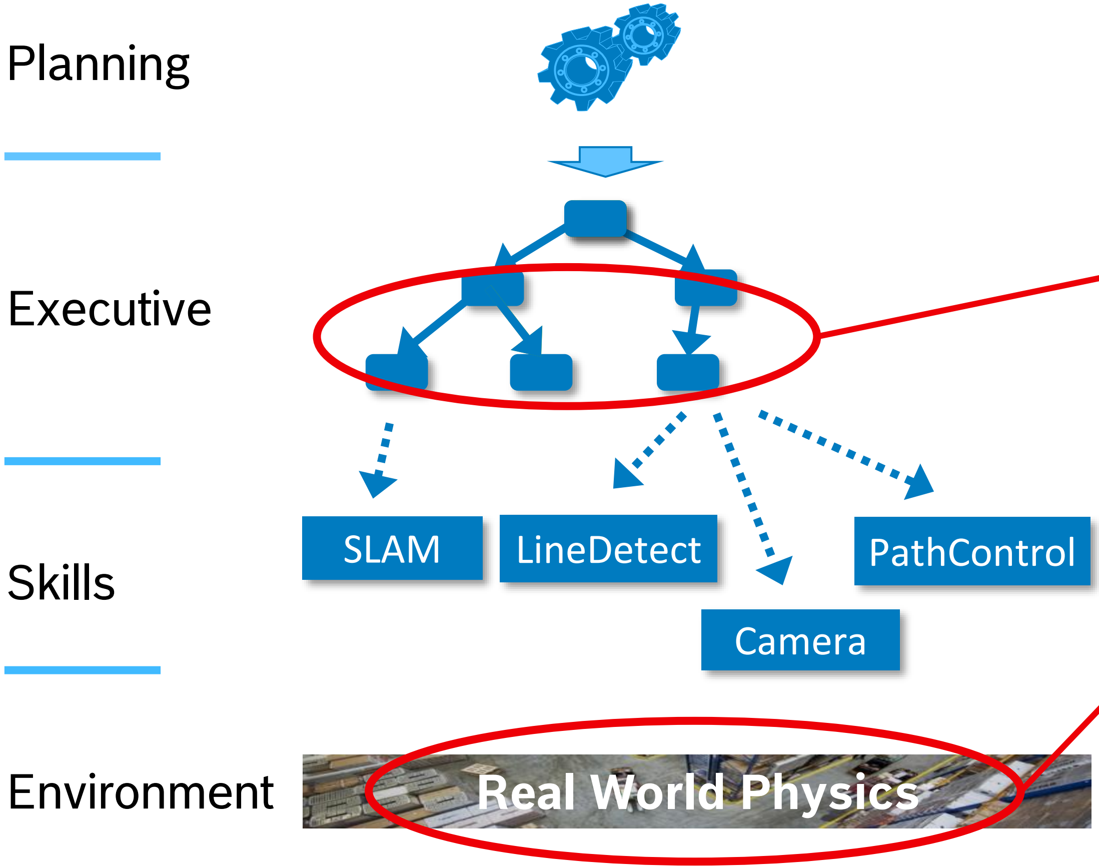
Motivation



FM

Complexity of Robotic Systems

Motivation



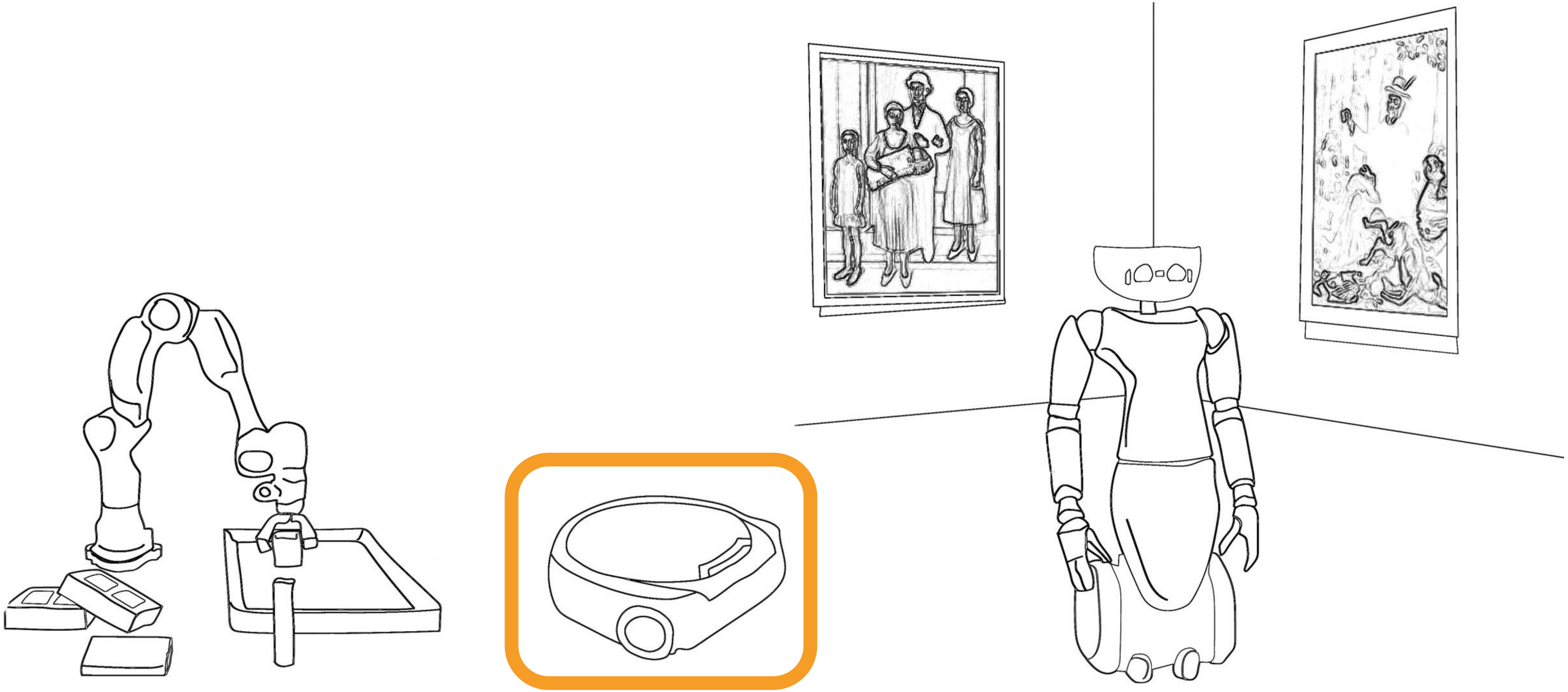
Sources of complexity

Custom (sub)tasks with platform-specific constraints and rules

Generalization to unknown environments

How to **verify** the system works robustly in **all possible** conditions?

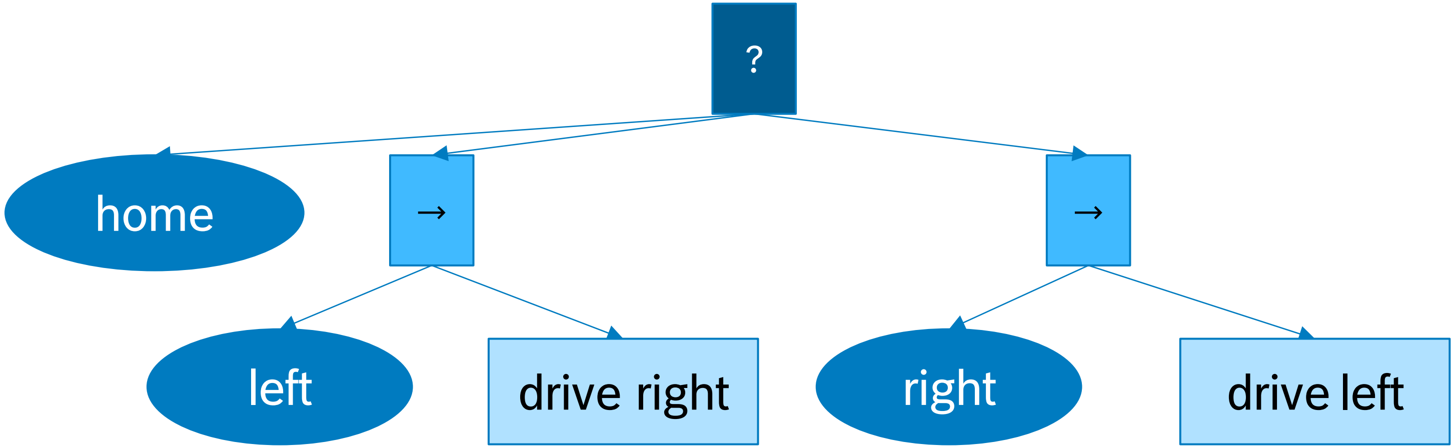
CONVINCE
Vacuum Cleaner Use Case



Funded by the European Union

Dynamic Deliberation with Behavior Trees

Defining the System‘s Behavior

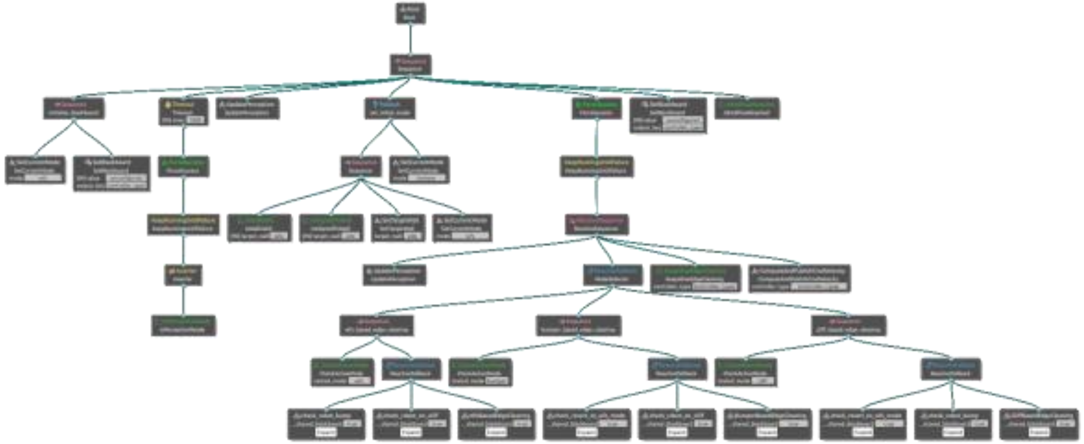


Funded by
the European Union

Use Cases

Dynamic Deliberation with Behavior Trees

Excerpt of edge cleaning BT



(blurred for confidentiality reasons)

Vacuum Cleaner Use Case Challenges

Motivation

Overall objective:

- High cleaning coverage (in given time)
  - Get into niches
  - Don’t be too conservative to clean close to obstacles
  - Depends on environment

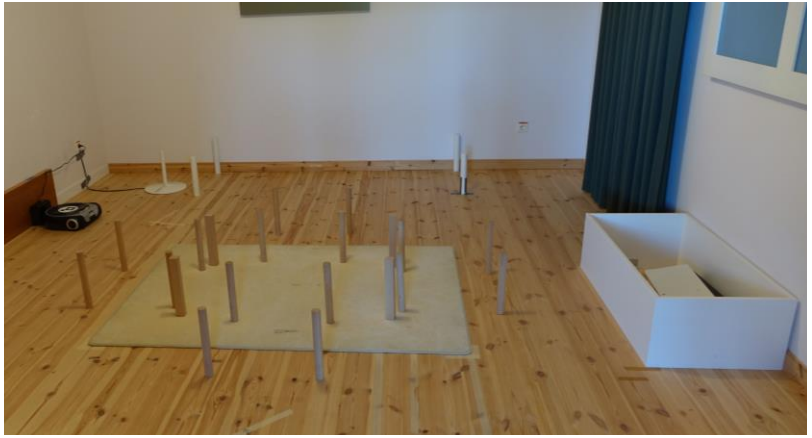
Operational Constraints:

- Prevent from getting stuck
- Don't get damaged
- Don't damage objects in the environment

Performance Measurements

Cleaning Coverage, Long-term Cleaning Task Completion, Obstacle Clearance

- Computed in percentage of room surface covered
- More complex settings by adapting test rooms
- Real-life appartments of test users
- Verify that robot learns how to deal with problematic areas over a time of two weeks
- Move small obstacles away to reach and clean additional space



IEC-62885-7 Test Room

Similar Challenges for Other Autonomous Systems

Motivation

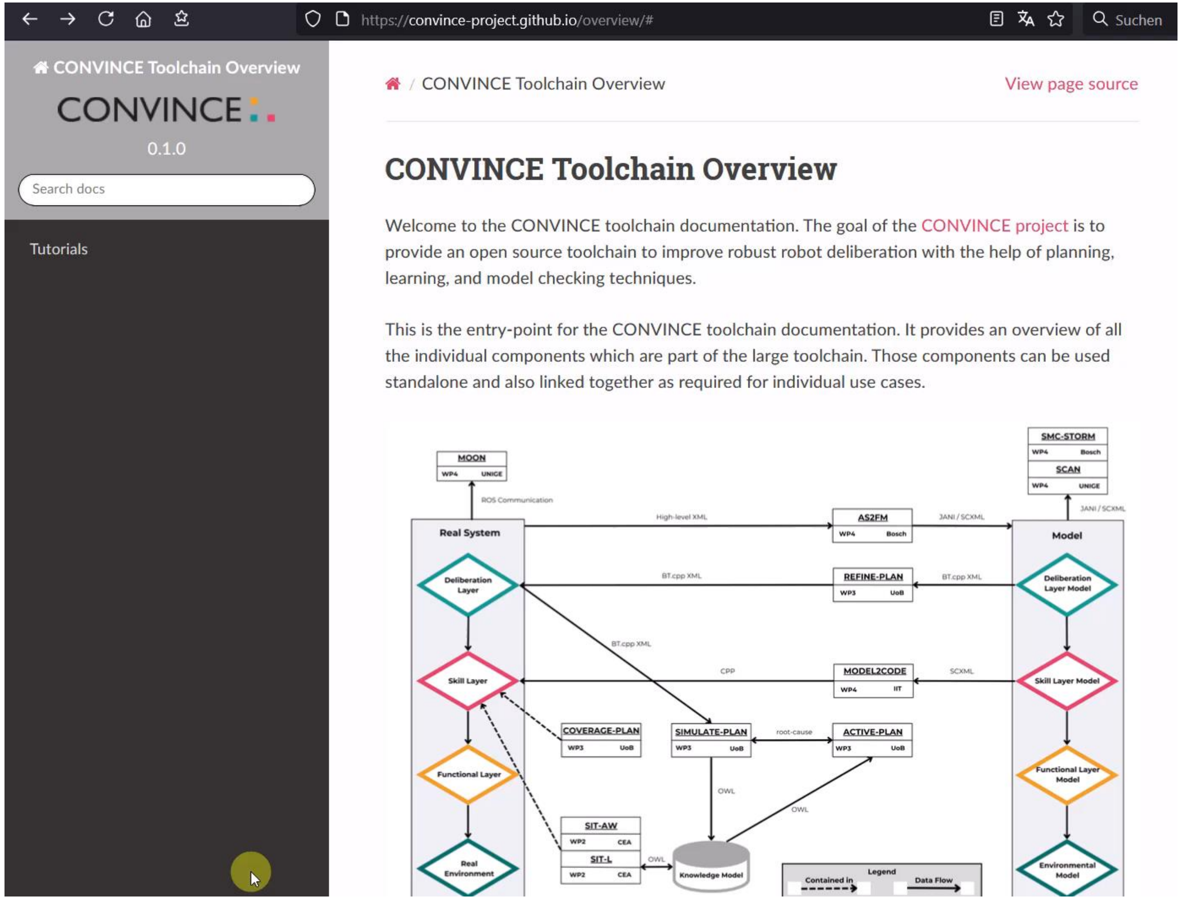
- Problem: Avoiding dangerous situations vs.
  Staying operational & having clever contingency handling
- How to make sure that anomalies are handled efficiently?
  - Preprogrammed behavior
  - Learning
  - Situation understanding & root cause analysis
  - Monitoring, testing
  - **Formal verification**
  - Or a mixture of all?

Toolbox Components

Verification, Monitoring, Planning, Situation Understanding

Street, Warsame, Mansouri, Klauck, Henkel, Lampacrescia, Palmas, Lange Ghiorzi, Tacchella, Azrou, Lallement, Morelli, Chen, Wallis, Bernagozzi, Rosa, Randazzo, Faraci, Natale.
Towards a Verifiable Toolchain for Robotics.
Proceedings of the AAAI Symposium Series 2024.

Best Paper Award


https://convince-project.github.io/overview/#
Suchen
CONVINCE Toolchain Overview
CONVINCE
0.1.0
Search docs
Tutorials
CONVINCE Toolchain Overview
View page source

CONVINCE Toolchain Overview

Welcome to the CONVINCE toolchain documentation. The goal of the CONVINCE project is to provide an open source toolchain to improve robust robot deliberation with the help of planning, learning, and model checking techniques.

This is the entry-point for the CONVINCE toolchain documentation. It provides an overview of all the individual components which are part of the large toolchain. Those components can be used standalone and also linked together as required for individual use cases.

https://convince-project.github.io/overview/

Industrial Use Cases

Vacuum Cleaning Robot & Autonomous Assembly Robot

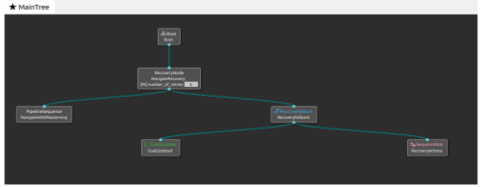
- Adapt dynamically to environment
- Model checking reduces need for timely & costly field tests

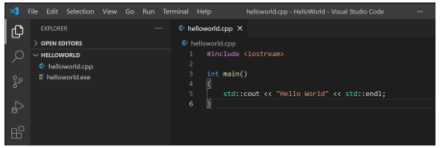
- P_{max}(stuck) < threshold
- close-to-dock → docking-time < 5 sec
- P_{min}(cleaning-coverage > thershold)

- Robust assembly: select, grasp, generate placement poses taking into account dependencies between parts & geometry of environment
- Detect defects and assembly anomalies

- no-defects → exec-time < 5 min
- not-fixable → replan ∧ part-pulled

Model Checking Tooling
Overview





```
#include <iostream>

int main()
{
    std::cout << "Hello World" << std::endl;
}
```

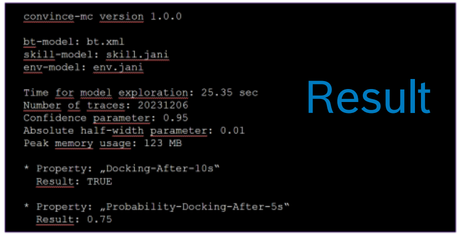
BT-Model

Skill Model

Environment Model

Properties

Model Checker



```
convince-mc version 1.0.0

bt-model: bt.xml
skill-model: skill.jani
env-model: env.jani

Time for model exploration: 25.35 sec
Number of traces: 20231206
Confidence parameter: 0.95
Absolute half-width parameter: 0.01
Peak memory usage: 123 MB

* Property: „Docking-After-10s"
  Result: TRUE

* Property: „Probability-Docking-After-5s"
  Result: 0.75
```

Result

Outline
How to Formally Verify Robotic Systems with Model Checking?

- Theoretical Foundations
  - Transition Systems
  - Markov Decision Processes (MDPs)
  - Linear Temporal Logic (LTL)
- Model Checking (MC)
  - Probabilistic Full State Space MC with Value Iteration
  - Statistical Model Checking
- Modeling Formats & Languages: JANI & SCXML
- Real World Application Example: Model Checking in Industry
- The CONVINCE Project
- Model Checker Tool Demo: The Modest Toolset, Storm & SMC Storm
- Hands-on: Modeling & Model Checking of Robot Behavior

Theoretical Foundations

Transition Systems,
Markov Decision Processes,
Linear Temporal Logic

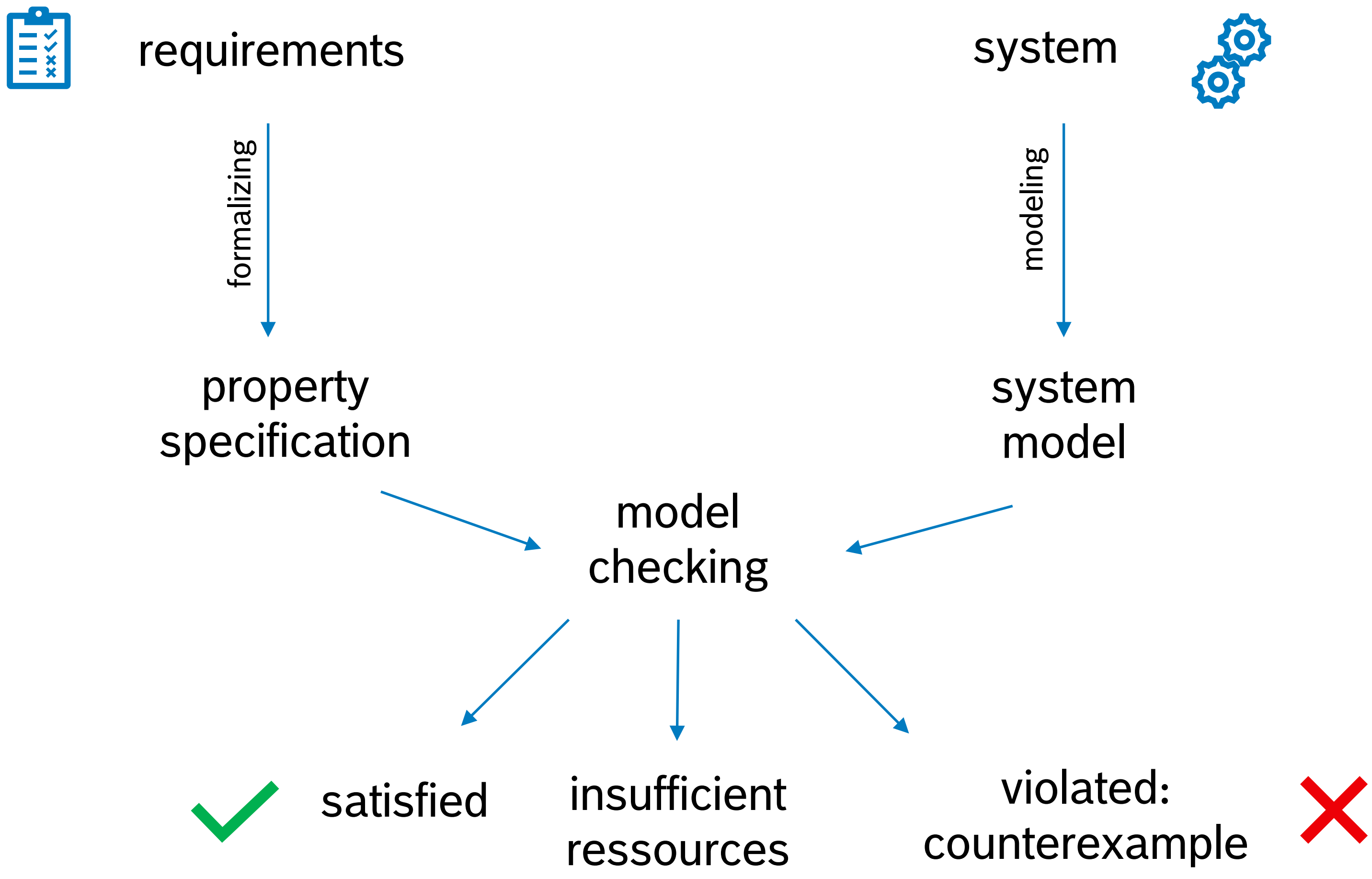
Formal Models

History

A Long Journey in the Footsteps of Famous Personalities

- Ensuring correctness in programs with mathematical techniques: Turing, 1949
- Proof rules for sequential programs: Hoare, 1969
  - Correct ouput for given input?
  - Predicate logic to formulate proof rules
- Proof rules for concurrent programs: Pnueli, 1977
  - Check correctness of infinte runs
  - Temporal logic to formulate proof rules
- Automated verification: Emerson, Clarke, Sifakis 1981
  - Systematic state space exploration
  - Model checking:
    - Given model of a system & formal property
    - Automatically and systematically check if property holds on model

Classical Model Checking
Overview



requirements

system

formalizing

modeling

property specification

system model

model checking

✓ satisfied

insufficient ressources

violated: counterexample ✗

Classical Model Checking

System Types → Model Types

- Finite-state reactive systems: Markov Chains (MC), Markov Decision Processes (MDP), Discrete-Time Markov Chains (DTMC)
  - Hardware, distributed protocols, discrete controllers
  - Non-terminating, concurrent, cooperating
- Timed automata
  - Real-time controllers, embedded systems
  - Finite-state systems + clocks
  - Infinite but underlying state space is finite
- Infinite-state systems
  - Software
  - Finite-state systems + data

How to Model System Behavior?
Transition Systems

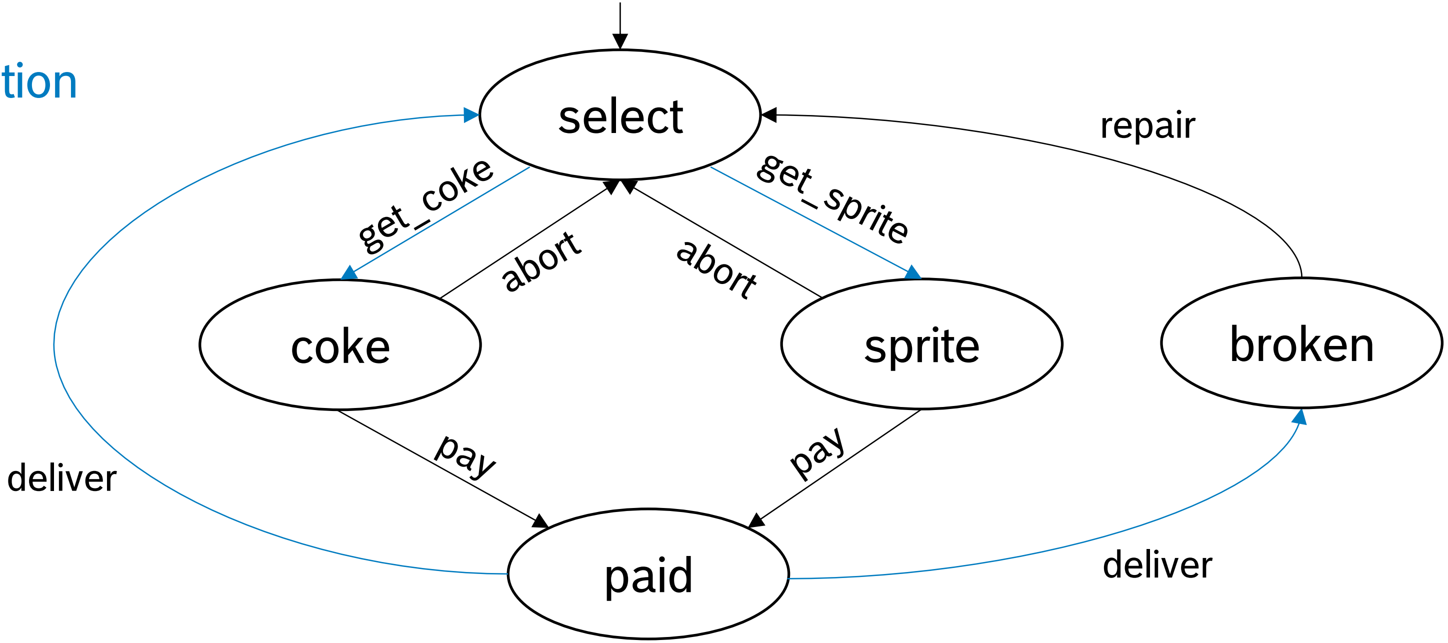
- Directed graph
  - Nodes to represent system state
  - Edges to represent transitions between states
- States:
  - Hardware: current value of registers + values of input bits
  - Software: current values of program variables + program counter
- Transitions:
  - Hardware: change of registers and output bits for new input
  - Software: execution of program statement

How to Model System Behavior?

Transition Systems

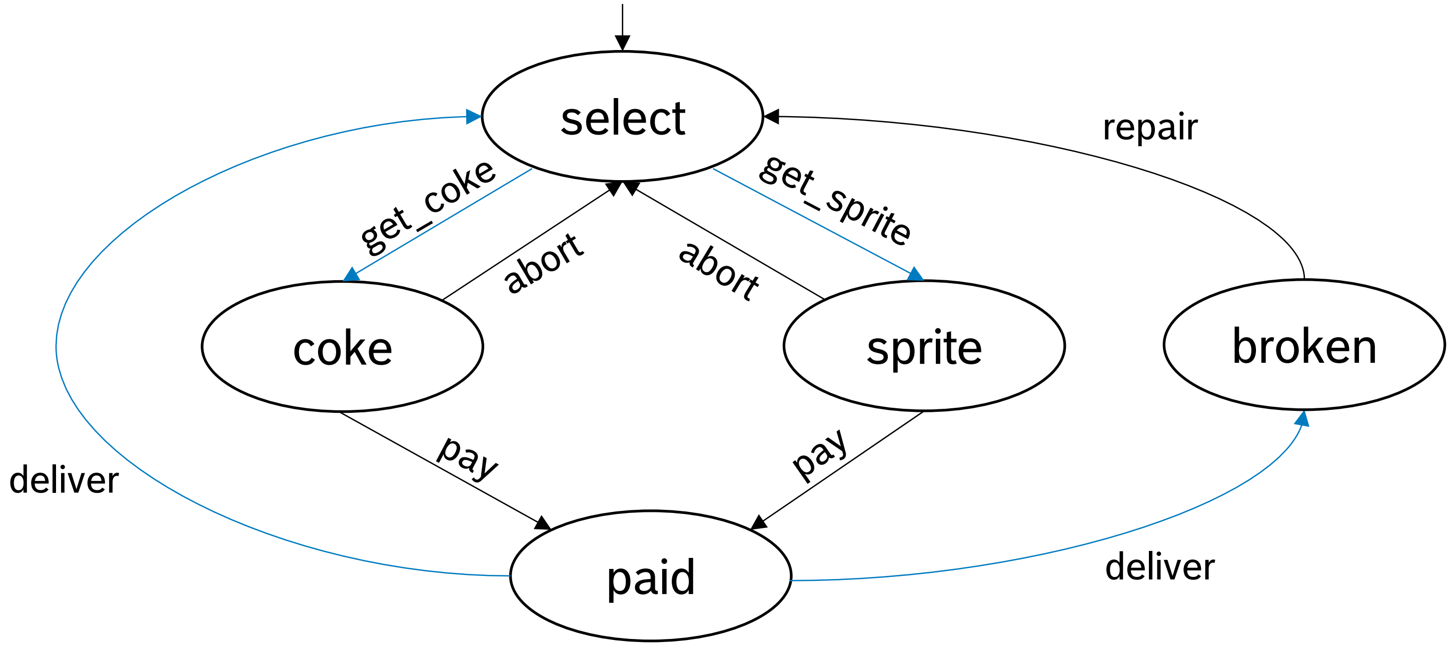
A transition system TS is a tuple (S, Act, →, I, AP, L) where:

- S is a set of states
- Act is a set of actions
- → ⊆ S x Act x S is a transition relation
- I ⊆ S is a set of initial states
- AP is a set of atomic propositions
- L: S → 2^{AP} is a labeling function

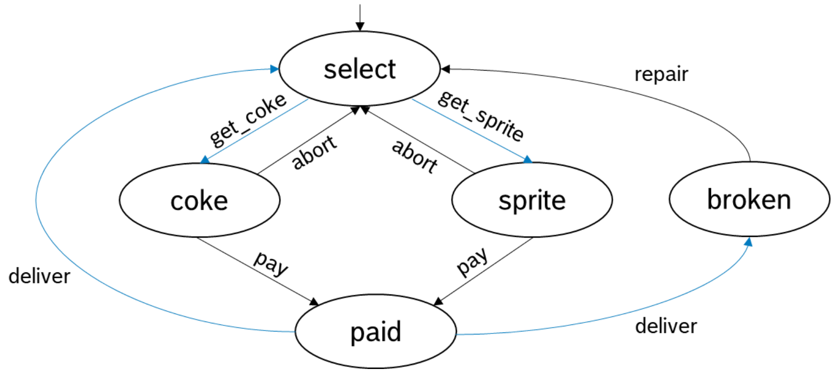


Example: Transition System

Beverage Vending Machine with Nondeterminism



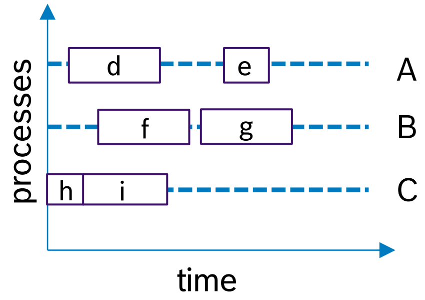
Nondterminism and it‘s Flavors
How to use it?



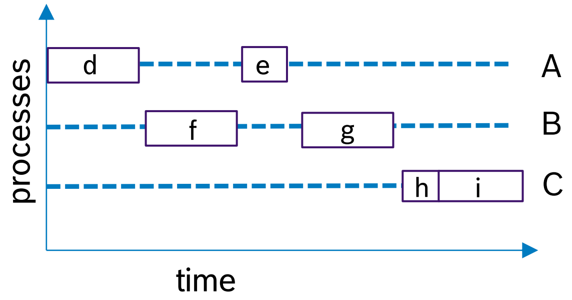
Concurrency: logically simultaneuos, most general

Parallelism: actually simultaneous

Separate processors available:

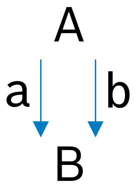


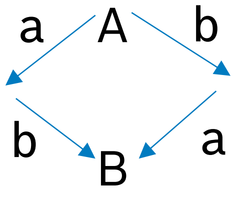
Only one processor available:



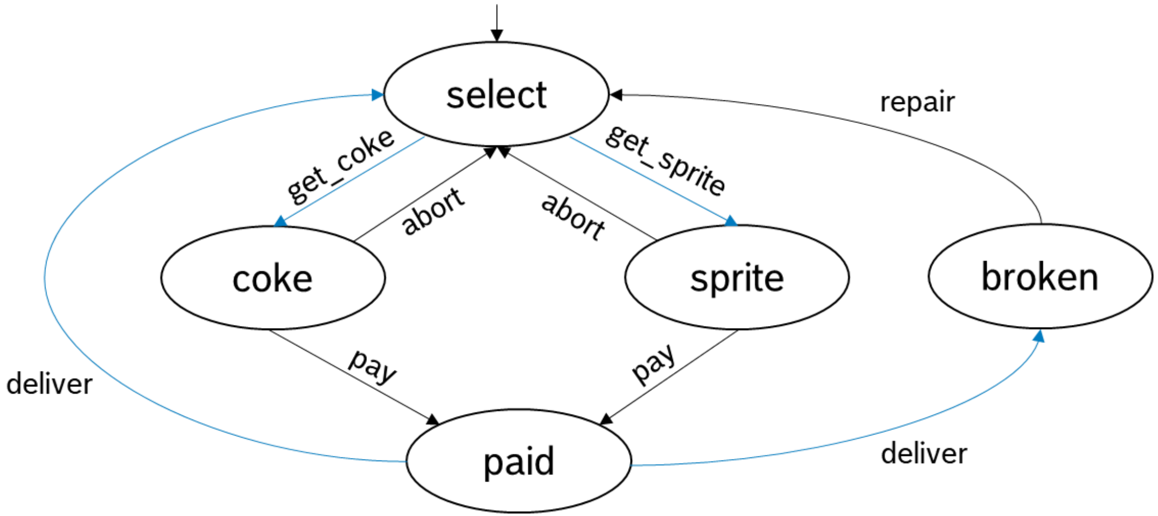
To model:

- Concurrency by interleaving
- Implementation freedom
- Under-specified or abstract systems





Example: Transition System
Beverage Vending Machine with Nondeterminism



Example Executions:

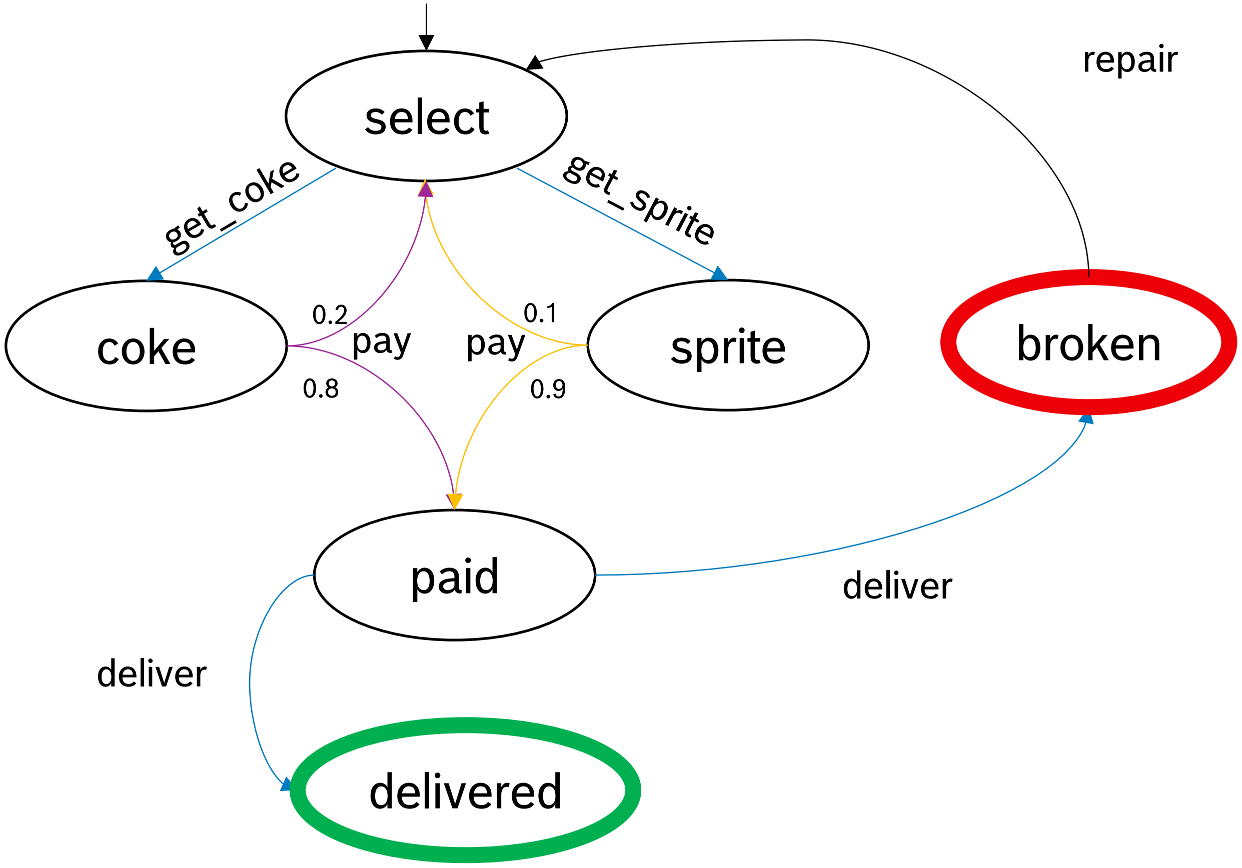
- select $\xrightarrow{\text{get\_sprite}}$ sprite $\xrightarrow{\text{pay}}$ paid $\xrightarrow{\text{deliver}}$ …
- select $\xrightarrow{\text{get\_coke}}$ coke $\xrightarrow{\text{abort}}$ select $\xrightarrow{\text{get\_coke}}$ coke $\xrightarrow{\text{pay}}$ paid $\xrightarrow{\text{deliver}}$ select $\xrightarrow{\text{get\_sprite}}$ …
- select $\xrightarrow{\text{get\_coke}}$ coke $\xrightarrow{\text{pay}}$ paid $\xrightarrow{\text{deliver}}$ broken terminal

How to Model System Behavior?
Markov Decision Processes

A Markov decision process MDP is a tuple (S, Act, $\rightarrow$, s_0, G) where:

- S is a set of states
- Act is a set of actions
- $\rightarrow \subseteq$ S x Act x D(S) is a partial transition probability function
  (into the discrete probability distributions D(S) over S)
- s_0 is the single initial state
- G $\subseteq$ S is a set of goal states

Example: Markov Decision Process
Beverage Vending Machine with Nondeterministic & Probabilistic Actions



Example: Markov Decision Process

Beverage Vending Machine with Nondeterministic & Probabilistic Actions

Variables:

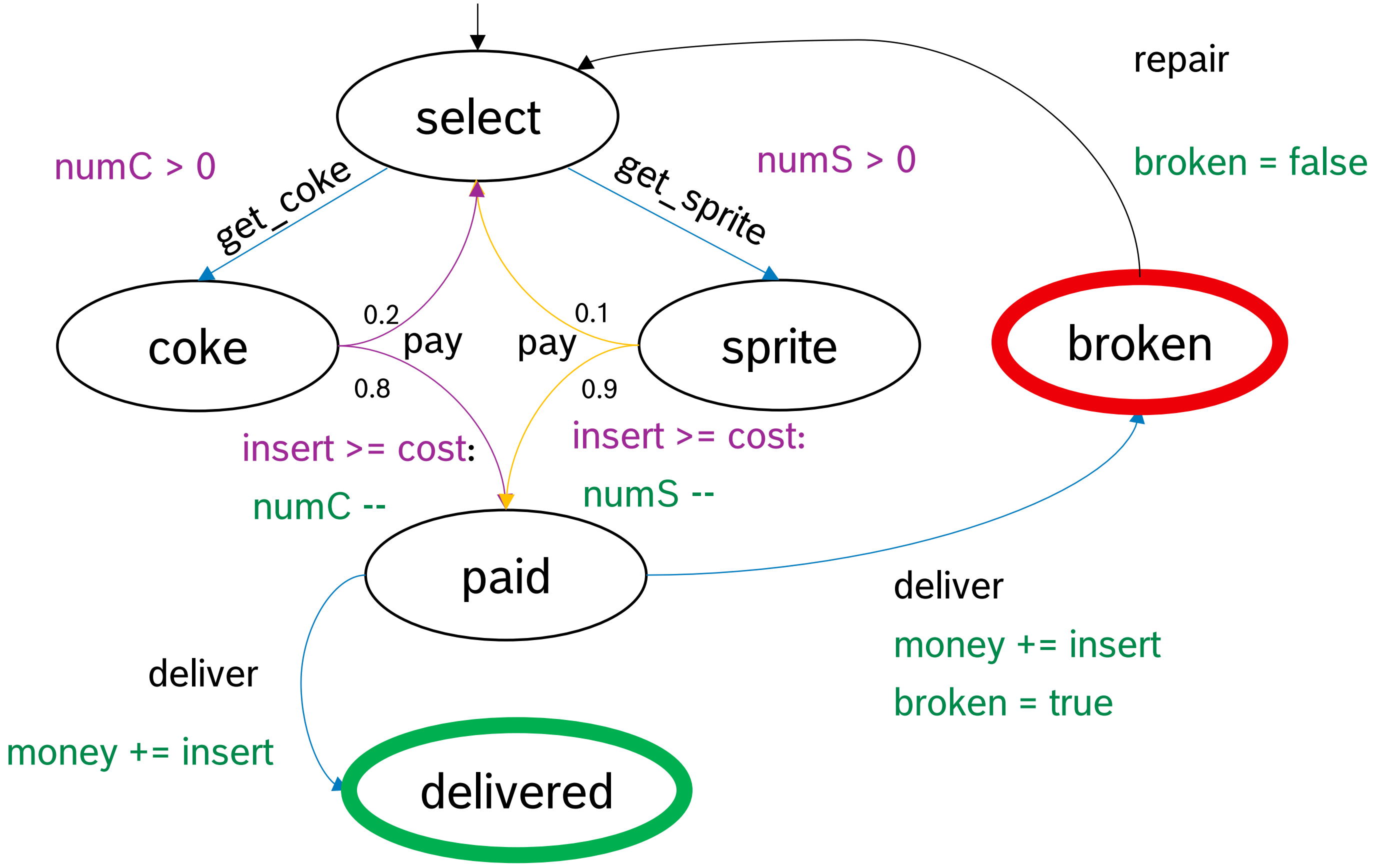
numS: int

numC: int

money: float

cost: float

broken: bool



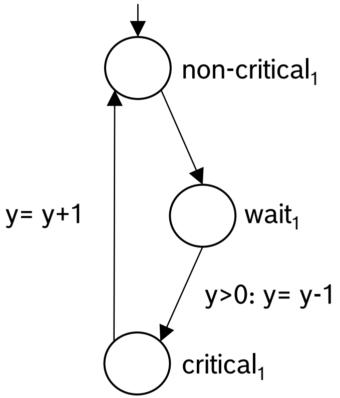
guards

assignments

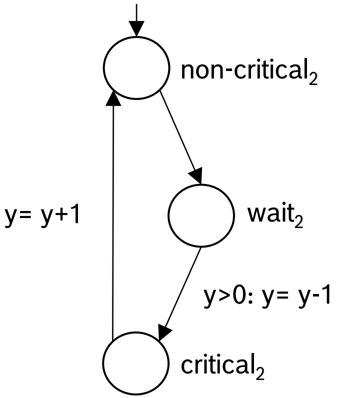
Parallelization & Interleavings

Systems with Multiple Processes

Process 1:



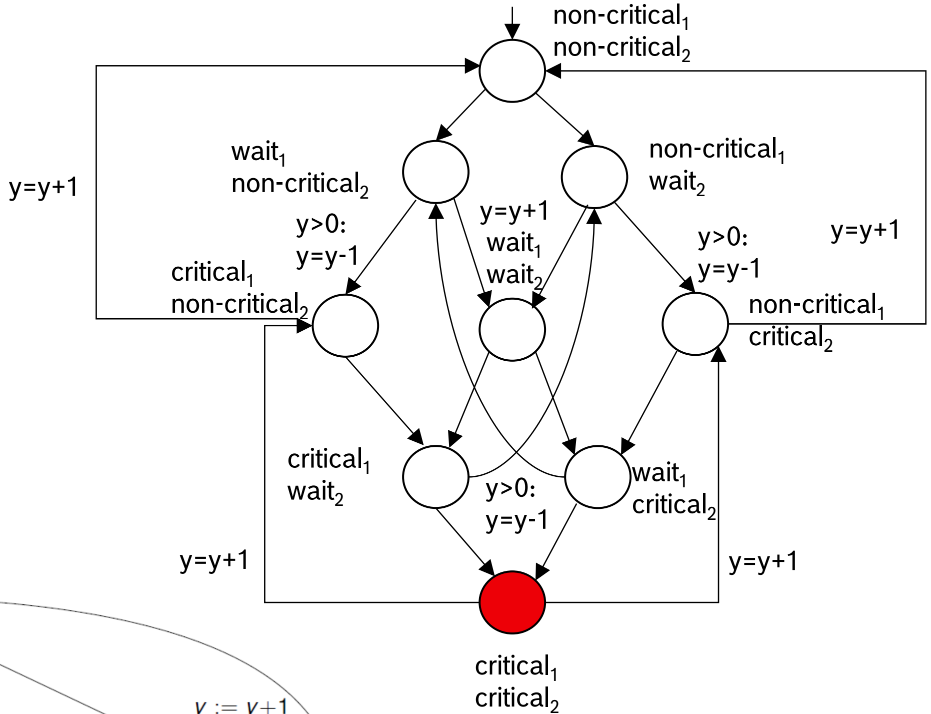
Process 2:

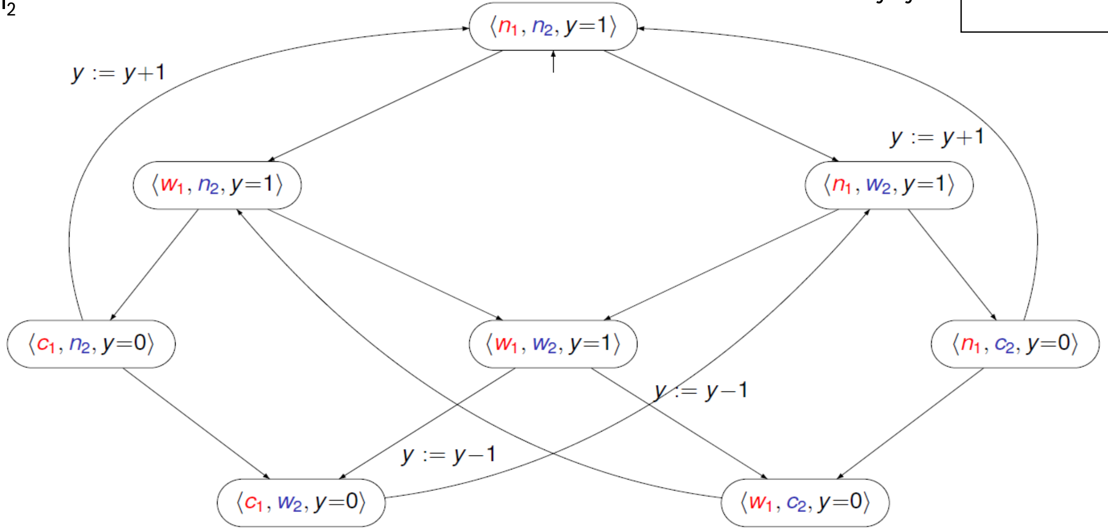


y=0: lock is currently taken

y=1: lock is free

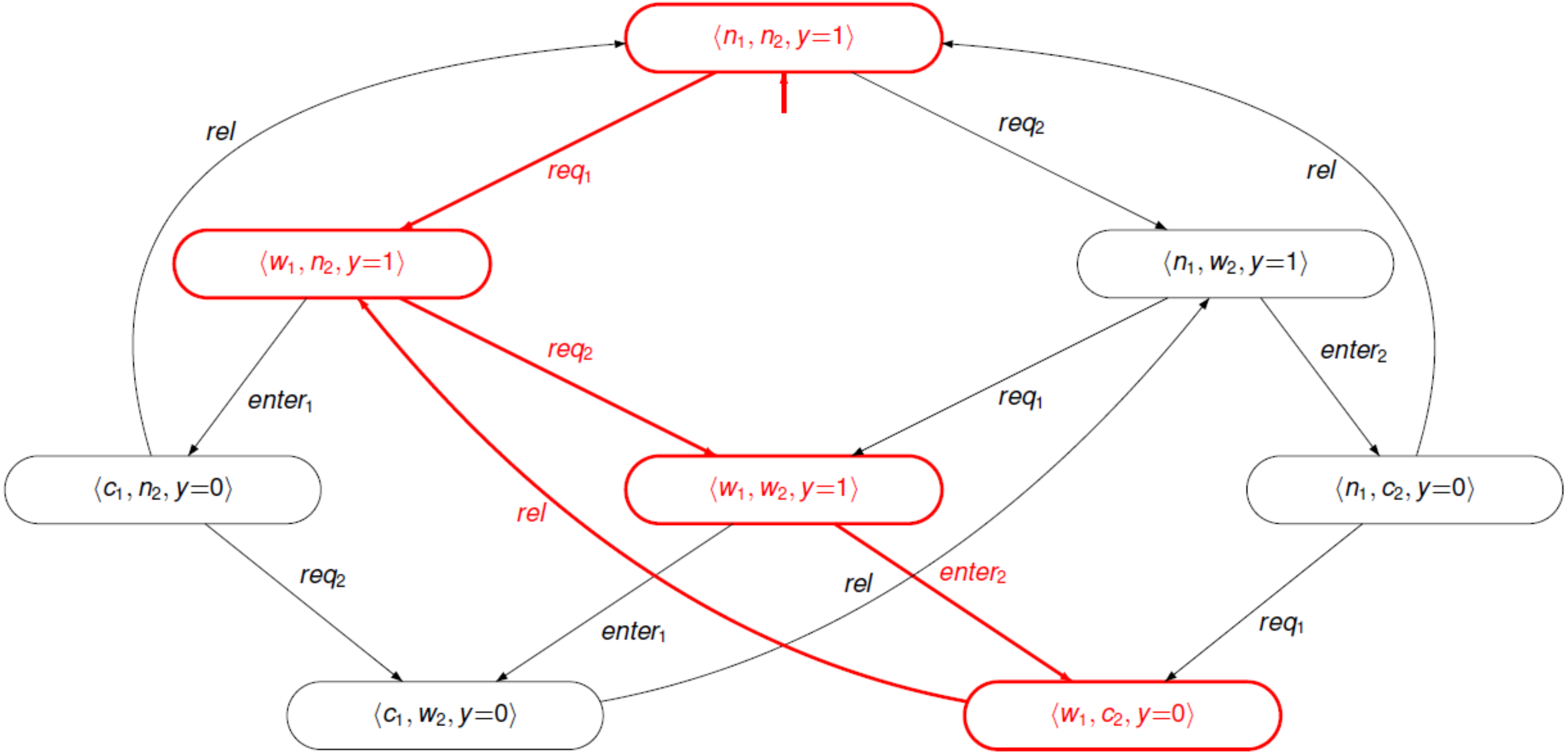
Process 1 || Process 2:


non-critical$_1$ non-critical$_2$
wait$_1$ non-critical$_2$
non-critical$_1$ wait$_2$
y=y+1
y>0: y=y-1
y=y+1 wait$_1$ wait$_2$
y>0: y=y-1
y=y+1
critical$_1$ non-critical$_2$
non-critical$_1$ critical$_2$
critical$_1$ wait$_2$
wait$_1$ critical$_2$
y>0: y=y-1
y=y+1
y=y+1
critical$_1$ critical$_2$


$\langle n_1, n_2, y{=}1\rangle$
$y := y{+}1$
$y := y{+}1$
$\langle w_1, n_2, y{=}1\rangle$
$\langle n_1, w_2, y{=}1\rangle$
$\langle c_1, n_2, y{=}0\rangle$
$\langle w_1, w_2, y{=}1\rangle$
$\langle n_1, c_2, y{=}0\rangle$
$y := y{-}1$
$y := y{-}1$
$\langle c_1, w_2, y{=}0\rangle$
$\langle w_1, c_2, y{=}0\rangle$

Parallelization & Interleavings

Problems to Avoid



$\langle n_1, n_2, y=1\rangle$
rel
req$_1$
req$_2$
rel
$\langle w_1, n_2, y=1\rangle$
$\langle n_1, w_2, y=1\rangle$
req$_2$
enter$_2$
enter$_1$
req$_1$
$\langle c_1, n_2, y=0\rangle$
$\langle w_1, w_2, y=1\rangle$
$\langle n_1, c_2, y=0\rangle$
rel
req$_2$
rel
enter$_2$
enter$_1$
req$_1$
$\langle c_1, w_2, y=0\rangle$
$\langle w_1, c_2, y=0\rangle$

Parallelization & Interleavings

Problems to Avoid

Fairness

Does the program Inc || Reset terminate (x shared, initially 0)?

Inc := while (x >= 0) : x = x+1

Reset := x = x-1

No, it is not guaranteed that Reset is executed eventually.

Fairness constraints needed: Concurrency = interleaving + fairness

Fair resolution of nondeterminism to rule out unrealistic runs

Parallelization & Interleavings

Problems to Avoid

Fairness Types:

- Unconditional fairness: actions are executed infinitely often
- Strong fairness: if an action is infinitely often enabled (not necessarily always)
  it has to be executed infinitely often
- Weak fairness: if an action is continuously enabled (no temporary disabling)
  it has to be executed infinitely often

Unconditional fairness ➔ strong fairness ➔ weak fairness

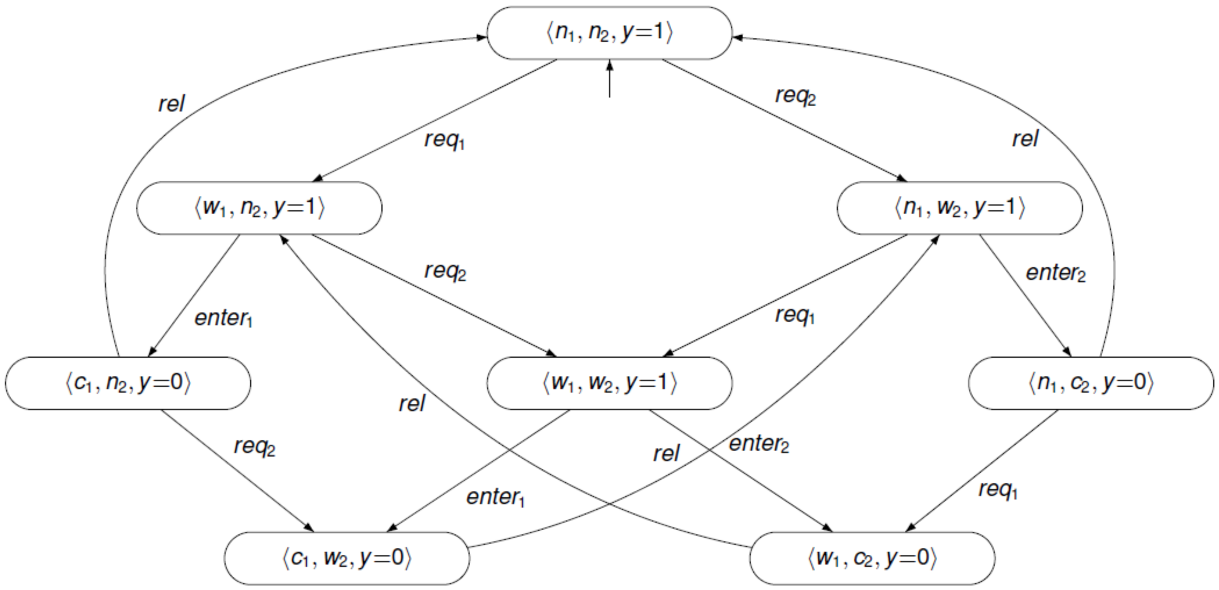
Choose the right fairness for your case:

Assumption too strong: verification could indicate that everything works as expected but some relevant execution breaking it could be ruled out

Assumption too weak: verification could indicate a problem in an unreasonable run

Parallelization & Interleavings

Problems to Avoid



$\langle n_1, n_2, y{=}1\rangle$
rel
req₂
req₁
rel
$\langle w_1, n_2, y{=}1\rangle$
$\langle n_1, w_2, y{=}1\rangle$
req₂
enter₂
enter₁
req₁
$\langle c_1, n_2, y{=}0\rangle$
$\langle w_1, w_2, y{=}1\rangle$
$\langle n_1, c_2, y{=}0\rangle$
rel
req₂
rel
enter₂
enter₁
req₁
$\langle c_1, w_2, y{=}0\rangle$
$\langle w_1, c_2, y{=}0\rangle$

$$\mathcal{F}' = \Big(\emptyset, \underbrace{\{\{\,enter_1\,\},\{\,enter_2\,\}\}}_{\mathcal{F}_{strong}}, \underbrace{\{\{\,req_1\,\},\{\,req_2\,\}\}}_{\mathcal{F}_{weak}}\Big)$$

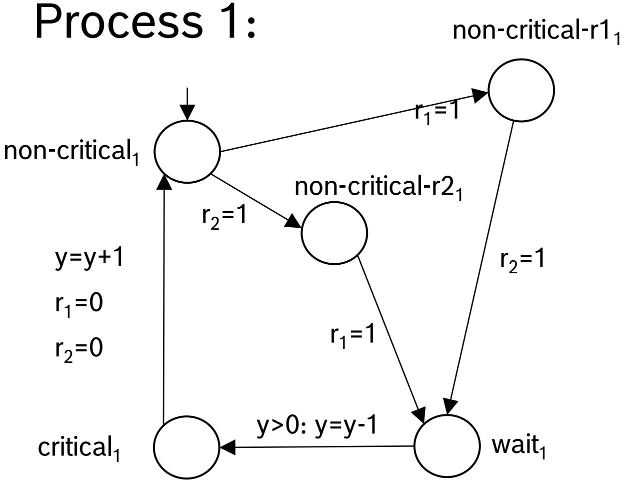
in any $\mathcal{F}'$-fair execution each process infinitely often requests access

Parallelization & Interleavings

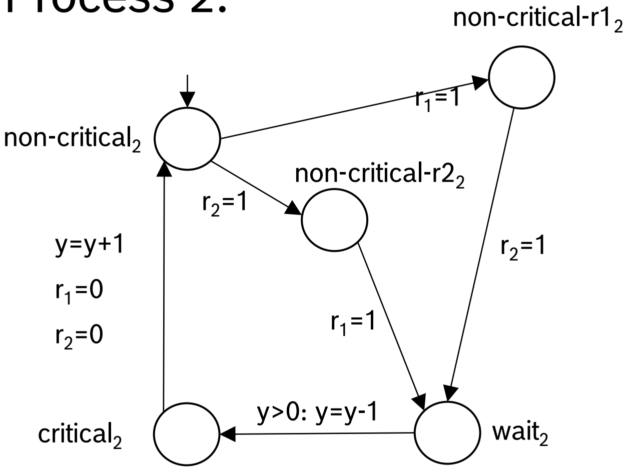
Problems to Avoid

Deadlocks

Process 1:



Process 2:



y=0: lock is currently taken

y=1: lock is free

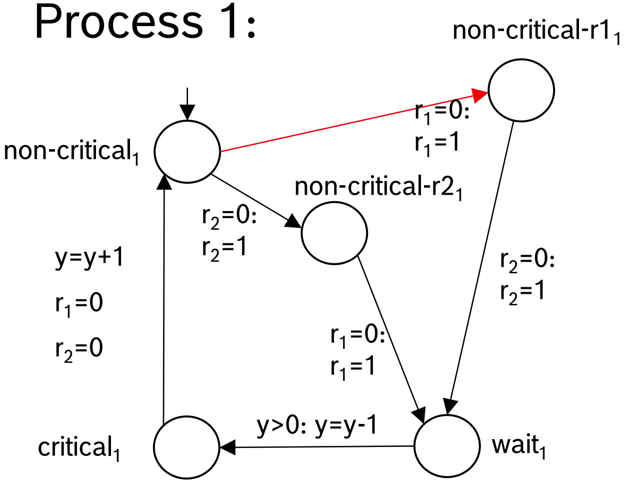
r_1, r_2 = 0/1: ressource free/ taken

Parallelization & Interleavings

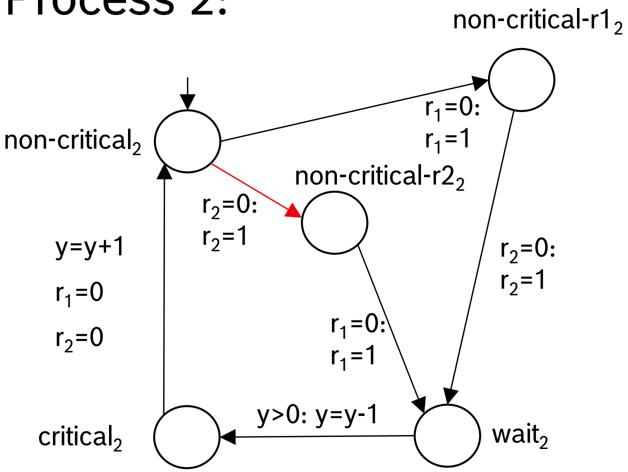
Problems to Avoid

Deadlocks

Process 1:



Process 2:



y=0: lock is currently taken

y=1: lock is free

r_1, r_2 = 0/1: ressource free/ taken

Sufficient & Necessary Criteria for a Deadlock:

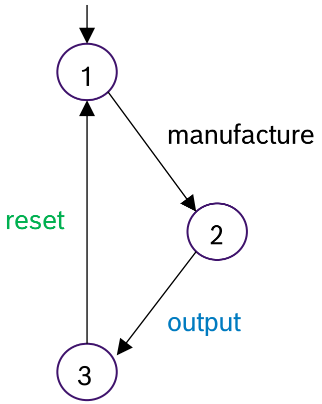
- Mutual Exclusion:
  Ressources are used exclusively by agents, they can only be used by one at a time
- No Preemption:
  Ressources can only be freed by the agents holding them
- Hold and Wait:
  Agents need access to other ressources but already hold exclusive acces to some of them
- Circular Wait:
  There is a circularity of needs

Parallelization & Interleavings
Composition by Handshaking

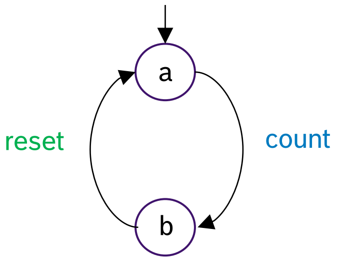
Some actions can be performed with a handshake between multiple systems/automata:

- Set of handshake actions: need to be performed synchronously together in all participating systems
- Set of independent actions: are executed independently in the automata, interleaved

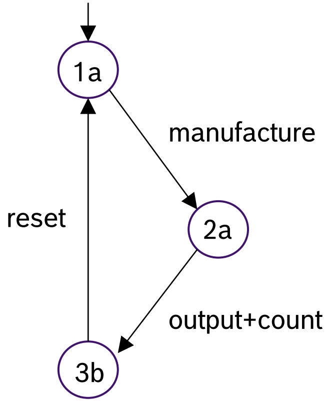
Production-Line:



Counter:



Production-Line $||_{\text{Handshake}}$ Counter
with Handshake = {{output, count}, {reset, reset}}



Properties

Linear Temporal Logic

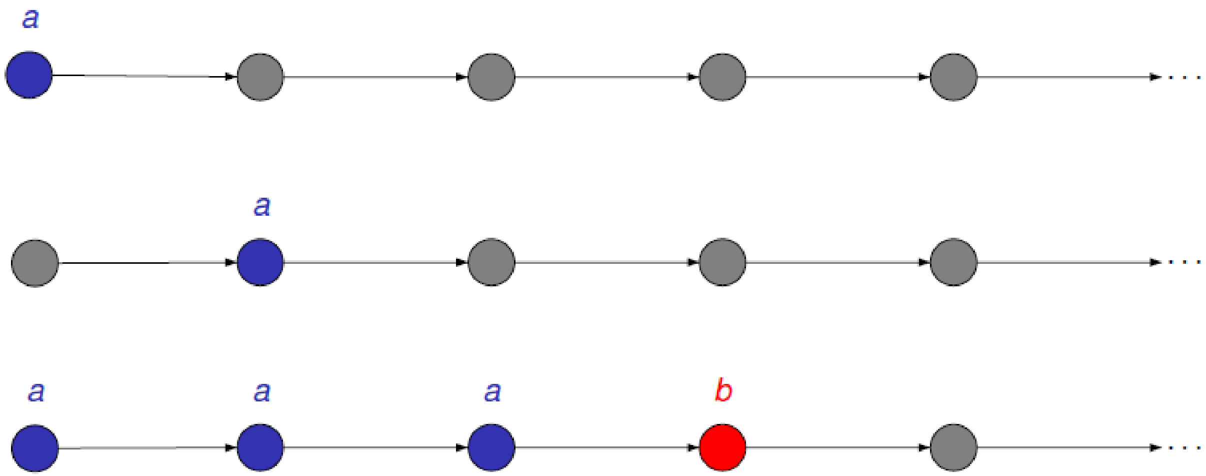
Expressing Properties
Linear-Time Temporal Logic (Pnueli 1977)

$$\varphi ::= a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2$$

atomic proposition
$a \in AP$

next operator
$\bigcirc a$

until operator
$a \,\mathsf{U}\, b$

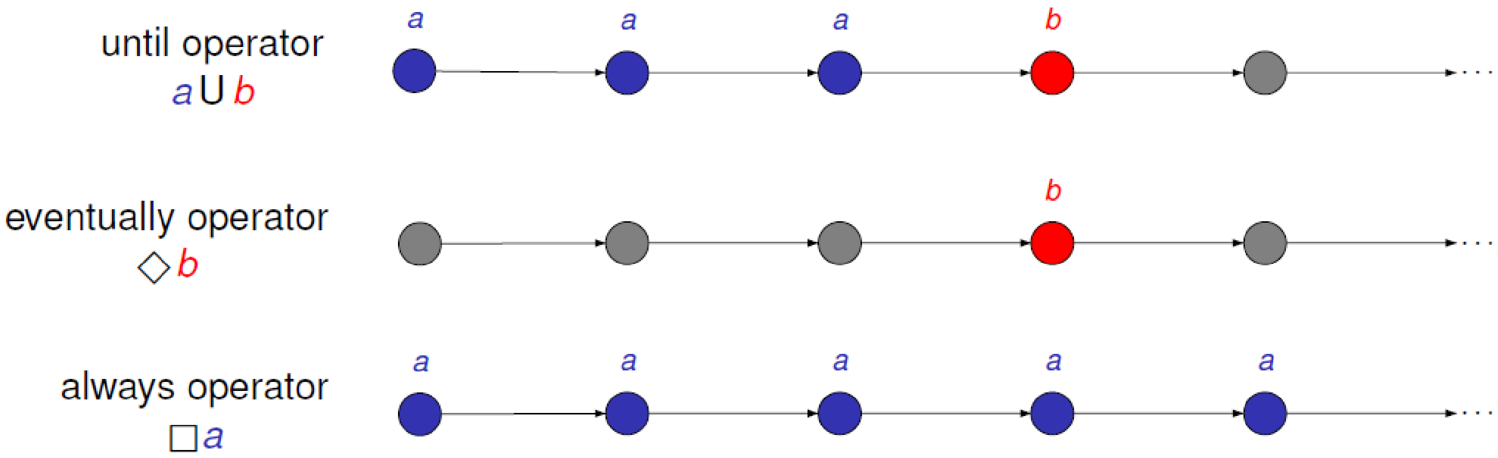


Expressing Properties
Linear-Time Temporal Logic – Derived Operators

$$\Diamond\varphi \equiv \text{true}\,\mathsf{U}\,\varphi$$

$$\Box\varphi \equiv \neg\Diamond\neg\varphi$$

boolean connectives $\vee, \Rightarrow, \Leftrightarrow, \ldots$ as usual



until operator
$a\,\mathsf{U}\,b$

eventually operator
$\Diamond b$

always operator
$\Box a$

Linear Temporal Logic
Example: Traffic Light

- It always holds that the light does not become green immediately when beeing red:

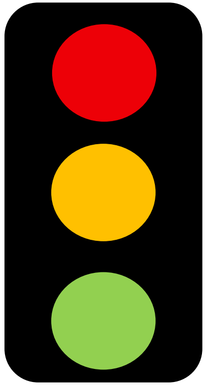
$$\Box(red \Rightarrow \neg \bigcirc green)$$

- Eventually, the light becomes green again: $\Diamond green$
- It always holds that the light always becomes green eventually when beeing red:

$$\Box(red \Rightarrow \Diamond green)$$

- When beeing red the light always becomes green eventually after being yellow for some time inbetween:

$$\Box(red \Rightarrow \bigcirc(red\, \mathrm{U}\, (yellow \wedge \bigcirc(yellow\, \mathrm{U}\, green))))$$



Linear Temporal Logic
Fairness Properties

Fairness Types: E and T: propositional logic formulas, E: something is enabled, T: something is taken

- Unconditional fairness: actions are executed infinitely often

  $\Box \Diamond T$

- Strong fairness: if an action is infinitely often enabled (not necessarily always)
  it has to be executed infinitely often

  $\Box \Diamond E \rightarrow \Box \Diamond T$

- Weak fairness: if an action is continuously enabled (no temporary disabling)
  it has to be executed infinitely often

  $\Diamond \Box E \rightarrow \Box \Diamond T$

Linear Temporal Logic
Safety & Liveness Properties

Safety Properties:

- Something bad will never happen
- Parallel Processes with write access to the same variable:
  Process 1 and process 2 are never in their critical sections at the same time

  $\Box\neg(\text{critical}_1 \wedge \text{critical}_2)$
- Are violated in finite time, finite counter example traces can be found

Liveness Properties:

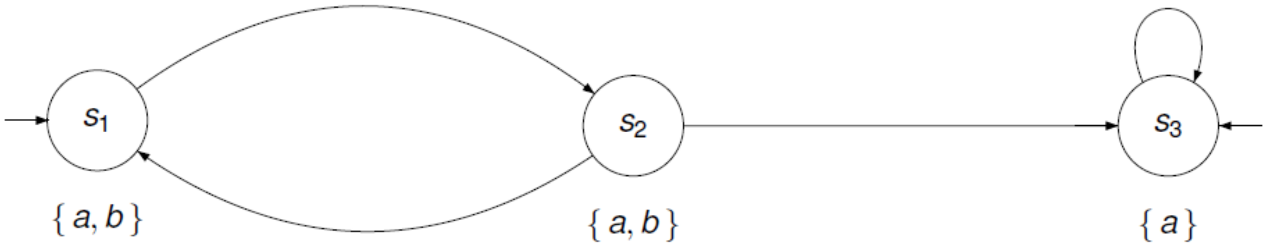
- Eventually something good will happen
- Whenever a process waits to enter its critical section, it will eventually be able to enter it.

  $\Box(\text{wait}_i \rightarrow \Diamond\text{critical}_i)$
- Are violated in infinite time, finite traces do not help to decide, no prefix is ruled out
- To prove liveness, fairness is typically needed (to prove progess, progess needs to be possible)

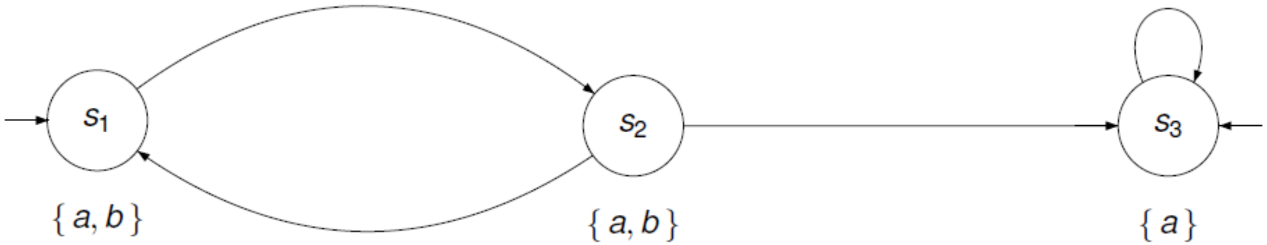
Model Checking

A First Glimps on Model Checking

Example on a Transition System

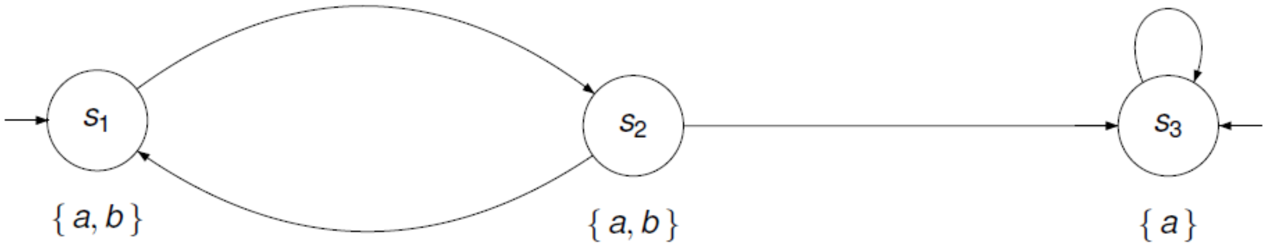


A First Glimps on Model Checking
Example on a Transition System



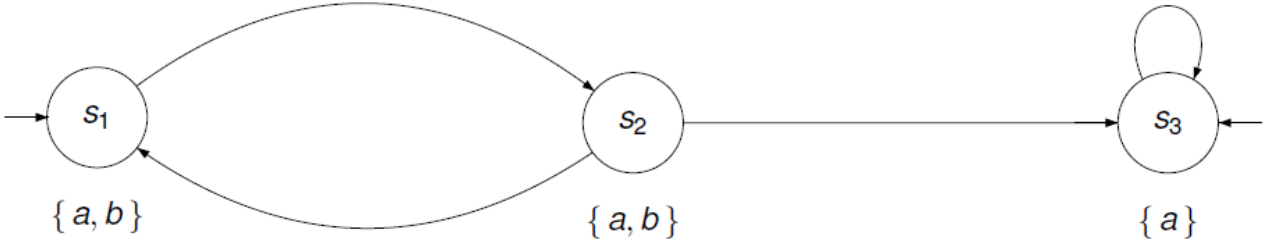
- $TS \models \square a$?

A First Glimps on Model Checking
Example on a Transition System



- $TS \models \Box a$? Yes

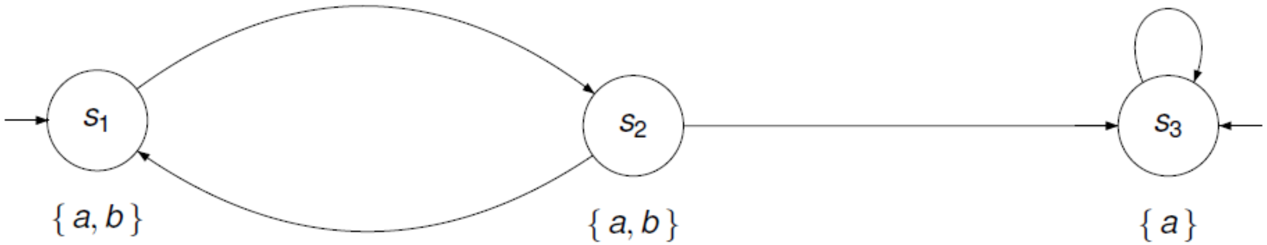
A First Glimps on Model Checking
Example on a Transition System



- $TS \models \Box a$? Yes
- $s_1 \models \bigcirc(a \wedge b)$?

A First Glimps on Model Checking

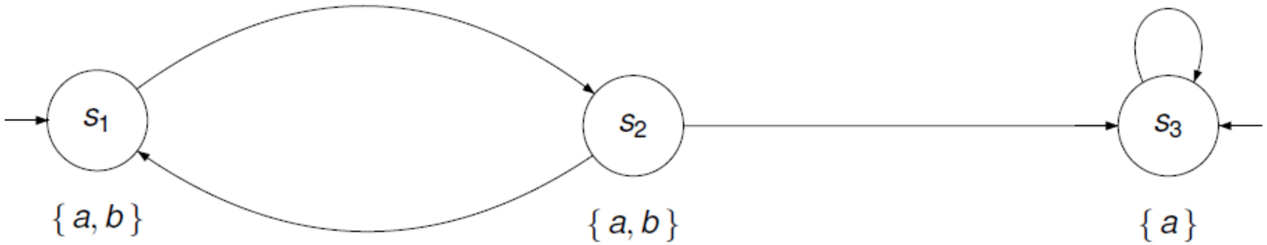
Example on a Transition System



- $TS \models \Box a$? Yes
- $s_1 \models \bigcirc(a \wedge b)$? Yes

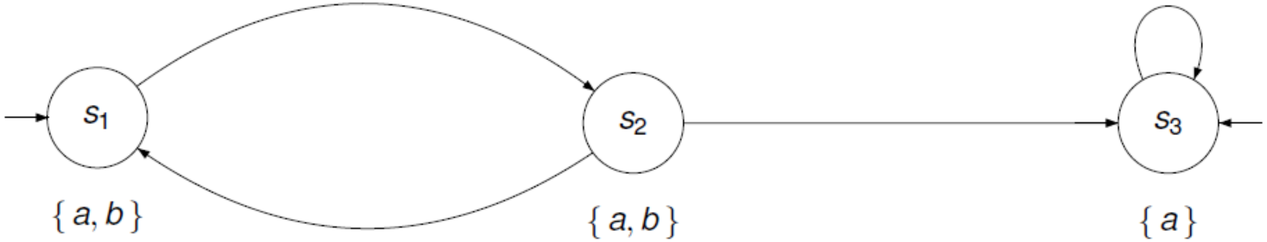
A First Glimps on Model Checking

Example on a Transition System



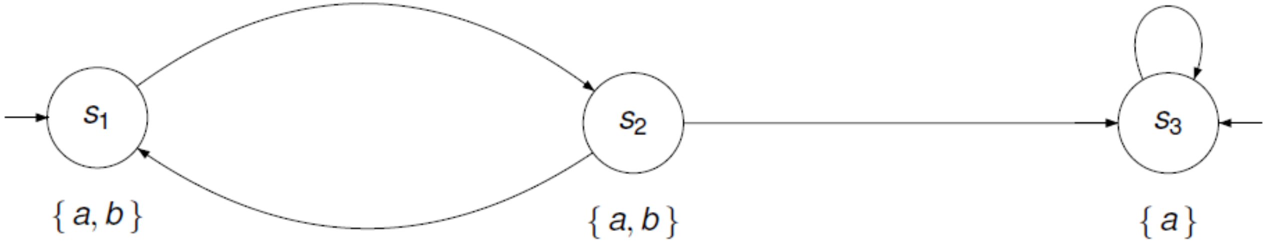
- $TS \models \Box a$? Yes
- $s_1 \models \bigcirc(a \wedge b)$? Yes
- $TS \models \bigcirc(a \wedge b)$?

A First Glimps on Model Checking
Example on a Transition System



- $TS \models \Box a$? Yes
- $s_1 \models \bigcirc(a \wedge b)$? Yes
- $TS \models \bigcirc(a \wedge b)$? No, because $s_3 \not\models \bigcirc(a \wedge b)$

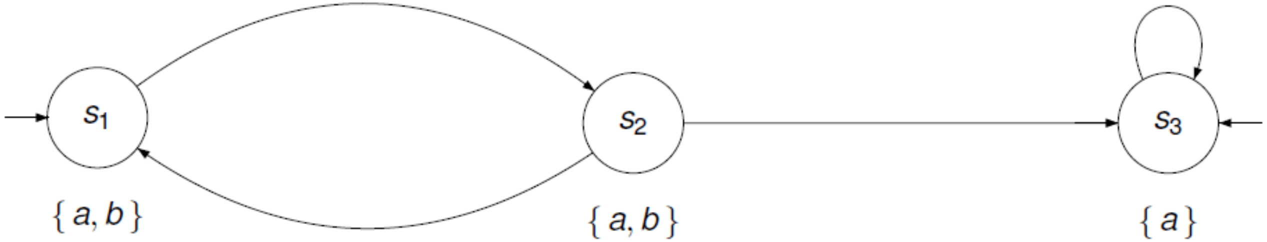
A First Glimps on Model Checking
Example on a Transition System



- $TS \models \Box a$? Yes
- $s_1 \models \bigcirc(a \wedge b)$? Yes
- $TS \models \bigcirc(a \wedge b)$? No, because $s_3 \not\models \bigcirc(a \wedge b)$
- $TS \models \Box(\neg b \Rightarrow \Box(a \wedge \neg b))$?

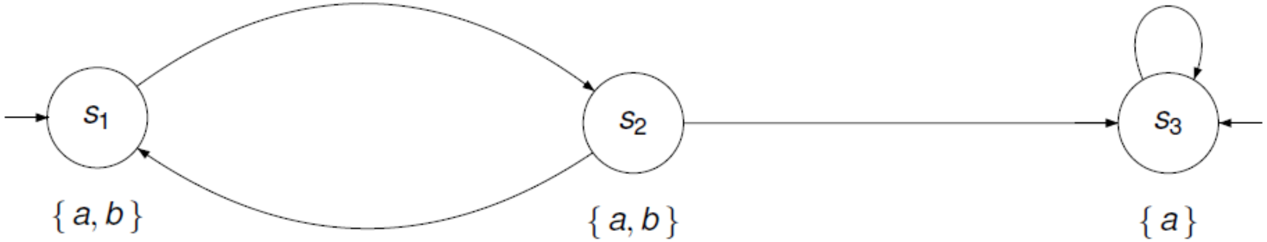
A First Glimps on Model Checking

Example on a Transition System



- $TS \models \Box a$? Yes
- $s_1 \models \bigcirc(a \wedge b)$? Yes
- $TS \models \bigcirc(a \wedge b)$? No, because $s_3 \not\models \bigcirc(a \wedge b)$
- $TS \models \Box(\neg b \Rightarrow \Box(a \wedge \neg b))$? Yes

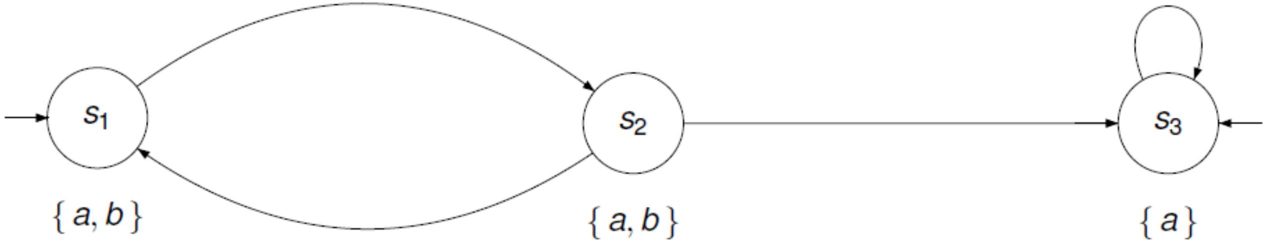
A First Glimps on Model Checking
Example on a Transition System



- $TS \models \Box a$? Yes
- $s_1 \models \bigcirc(a \wedge b)$? Yes
- $TS \models \bigcirc(a \wedge b)$? No, because $s_3 \not\models \bigcirc(a \wedge b)$
- $TS \models \Box(\neg b \Rightarrow \Box(a \wedge \neg b))$? Yes
- $TS \models b \, \mathsf{U} \, (a \wedge \neg b)$?

A First Glimps on Model Checking

Example on a Transition System



- $TS \models \Box a$? Yes
- $s_1 \models \bigcirc(a \wedge b)$? Yes
- $TS \models \bigcirc(a \wedge b)$? No, because $s_3 \not\models \bigcirc(a \wedge b)$
- $TS \models \Box(\neg b \Rightarrow \Box(a \wedge \neg b))$? Yes
- $TS \models b \, \mathsf{U} \, (a \wedge \neg b)$? No, because $(s_1 s_2)^\omega \not\models b \, \mathsf{U} \, (a \wedge \neg b)$

Example: Markov Decision Process

Beverage Vending Machine with Nondeterministic Actions

- Immediately after selecting we have to pay:

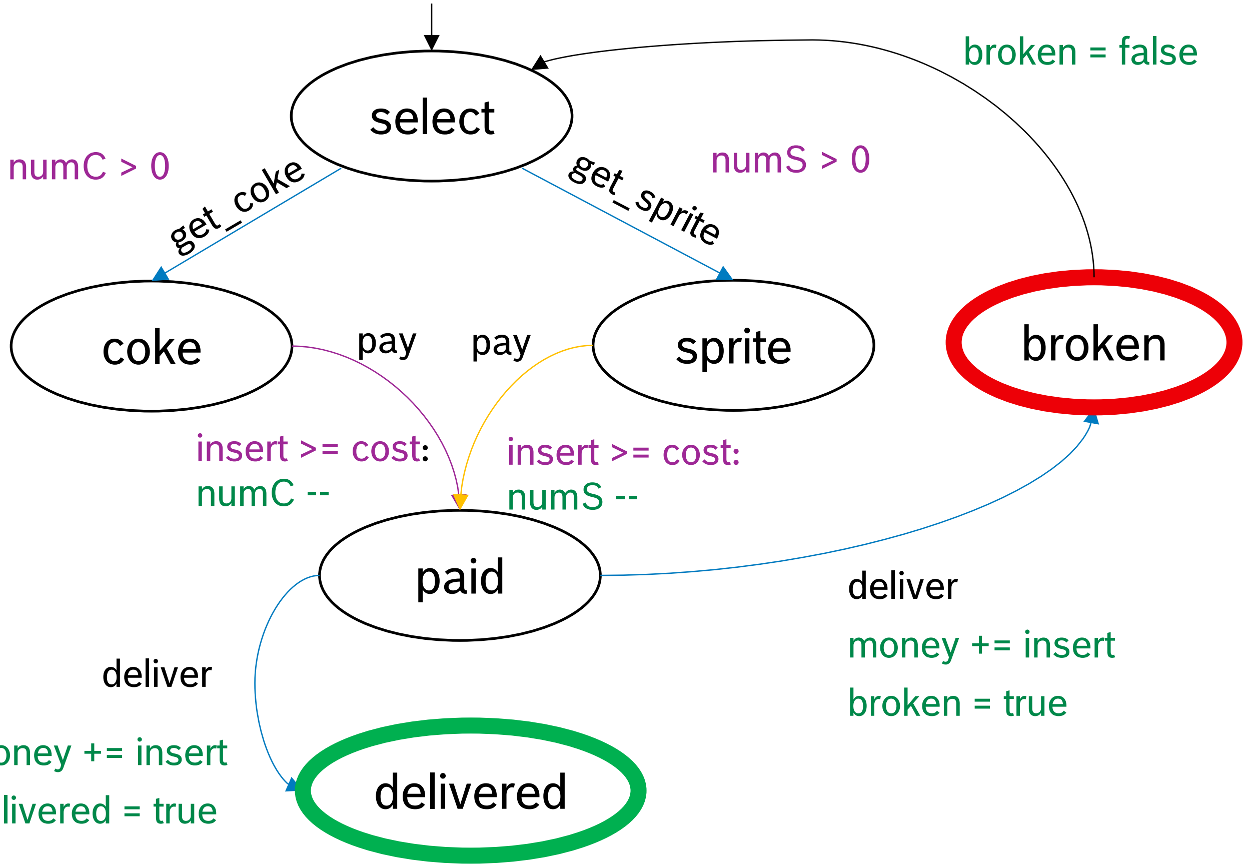
  select ➔ ○payed

- The machine will always be back in operation at some point:

  □ (broken ➔ ◇! broken)

- If we have enough money and there is coke in the machine it will be delivered at some point:

  (insert >= cost & numC >0) ➔ ◇ delivered



Variables:

numS: int numC: int

money, cost: float broken, delivered: bool

Example: Markov Decision Process

Beverage Vending Machine with Nondeterministic & Probabilistic Actions

- Probability to have to select twice in the beginning:

  P(○○ select)

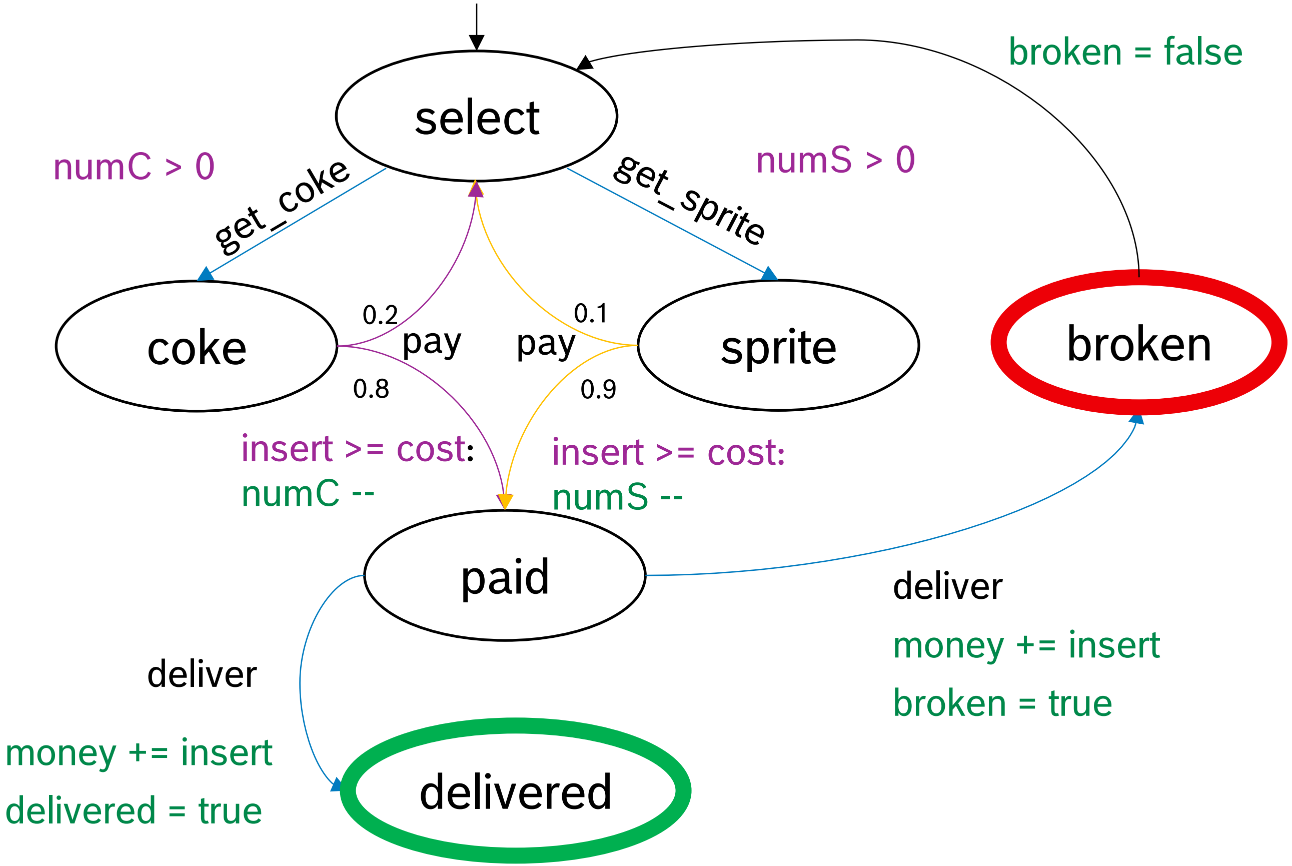
- Probability that if we have enough money and there is coke in the machine it will be delivered at some point after we selected it:

  P((insert >= cost & numC >0)
  ➔◇delivered)

Variables:

numS: int numC: int

money, cost: float broken, delivered: bool



select

broken = false

numC > 0

get_coke

get_sprite

numS > 0

coke

0.2

pay

pay

0.1

sprite

broken

0.8

0.9

insert >= cost:
numC --

insert >= cost:
numS --

paid

deliver
money += insert
broken = true

deliver
money += insert
delivered = true

delivered

How to Model Check Timing Issues?
Discrete vs. Continuous Time

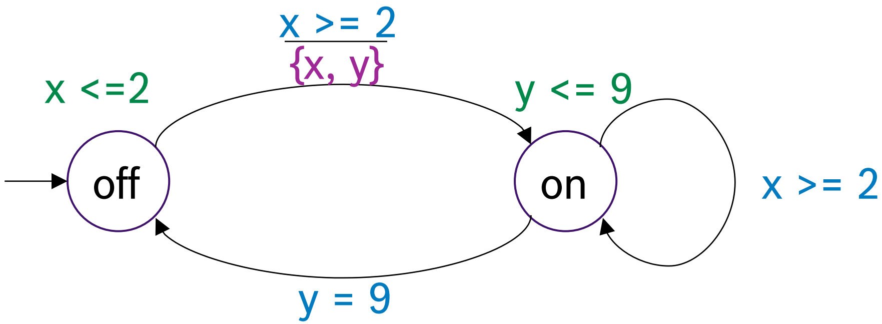
- Correctness depends not only on logical result of computation but also on time when results are produced
- Robot controllers, landing gear controller of airplane, railway crossing, communication protocols
- Discrete time:
  - Discrete steps, natural time values, tick actions
  - Minimal delay is a priori fixed and difficult to determine in practice
  - Properties: traditional temporal logic with next operator as time measure
  - Standard model checking algorithms suffice
  - Often sufficient in synchronous systems, e.g., hardware

How to Model Check Timing Issues?
Discrete vs. Continuous Time

- Continuous time:
  - State changes can happen at any point in time
  - Needed for asynchronous systems, e.g., distributed systems
  - Properties: very expressive in general, therefore restrict expressivity:
    - Only reference to natural time units: Timed CTL
    - Model timed systems symbolically rather than explicitly: Timed automata (TA)
    - Consider finite quotient of infinite state space (equivalence depending on property + TA): Region automata

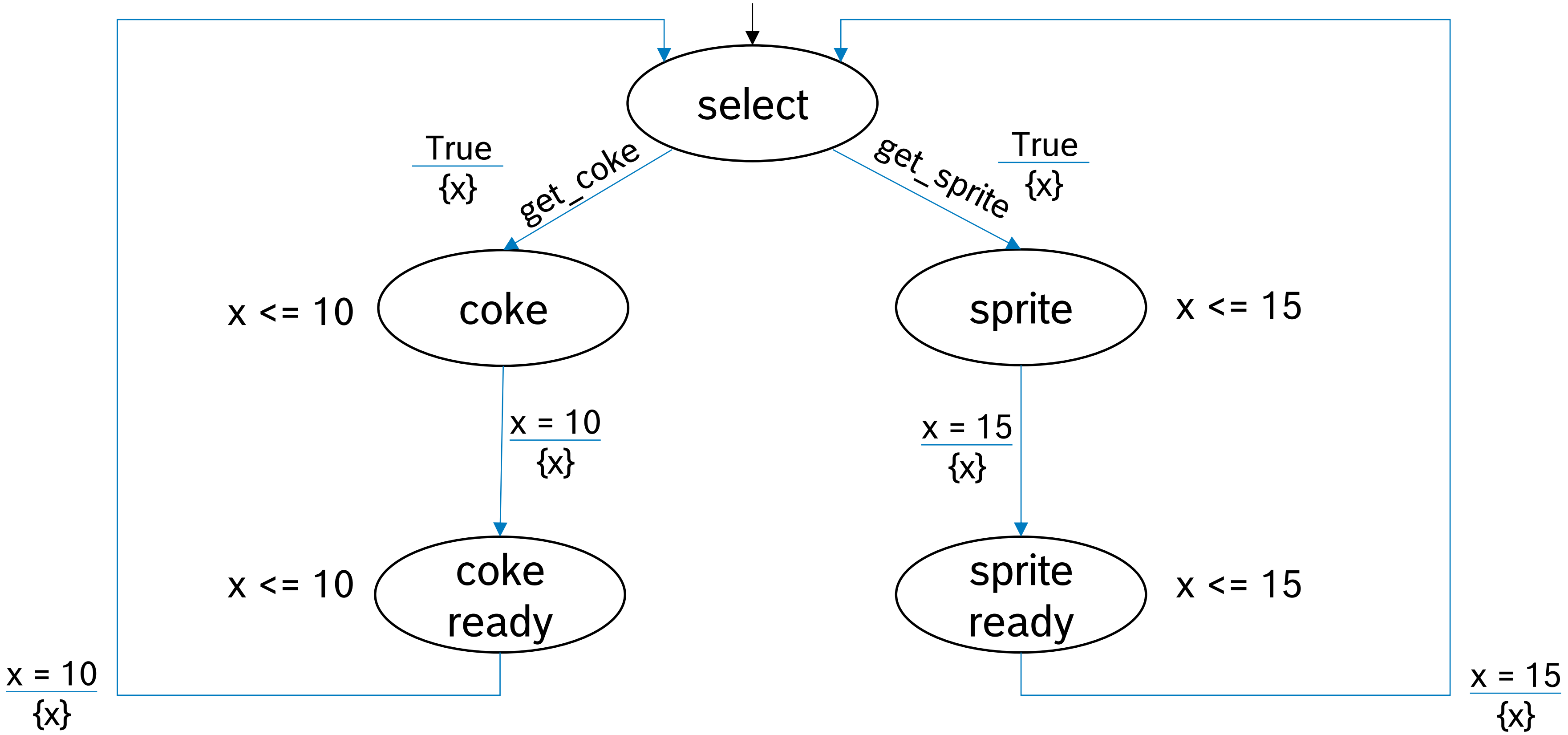
Timed Automata
How to Handle Time?

- Clocks x, y: real
- Advance implicitly at same speed
- Clock constraints: guards of actions
- Guards indicate when edge **may** be taken
- Clocks can be reset to initial value 0
- Location invariants: time that may be spent in location
- Before location invariant gets invalid, edge **must** be taken



- Problems:
  - Time convergence: time advances only up to certain value (convergent sum)
  - Timelocks: no behavior where time can progress ad infinitum in a state
  - Zenoness: infinite number of actions in finite time

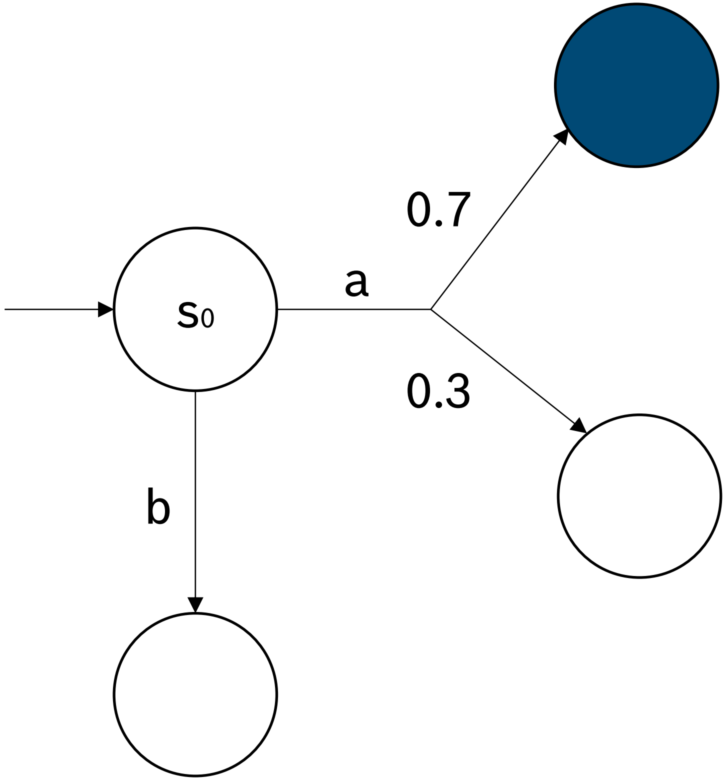
Timed Automata
Example



Model Checking

Probabilistic Full State Space MC with Value Iteration,
Statistical Model Checking (SMC)

Markov Decision Processes (MDPs)
Recap



Probabilistic Reachability:

$$P_{max}(\diamond \text{Goal})$$

Interface:

```
- s0
- actions(s)
- sample(s,a)
- distr(s,a)
- goal(s)
```

Model Checking
Introduction

%

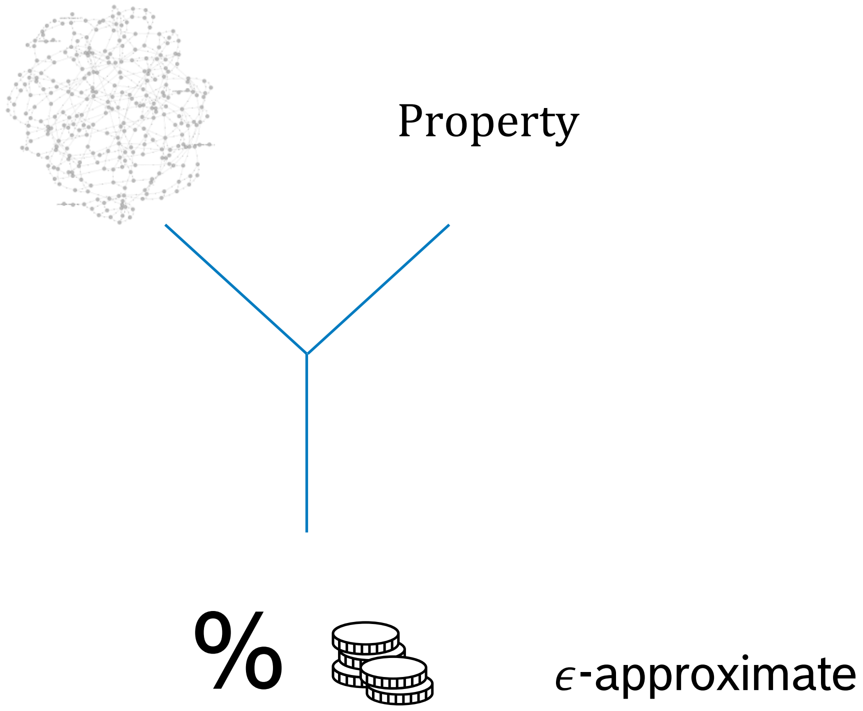
Probabilistic Model Checking (PMC)

- Automated technique for formally verifying quantitative properties of stochastic and non-deterministic systems
- Giving results with predefined error ranges

Statistical Model Checking (SMC)

- Always depending on strategy:
  - Lightweight Scheduler Sampling (LSS)
  - Deep Learning (DSMC)
- Giving results with statistical guarantees

Probabilistic Model Checking
Introduction



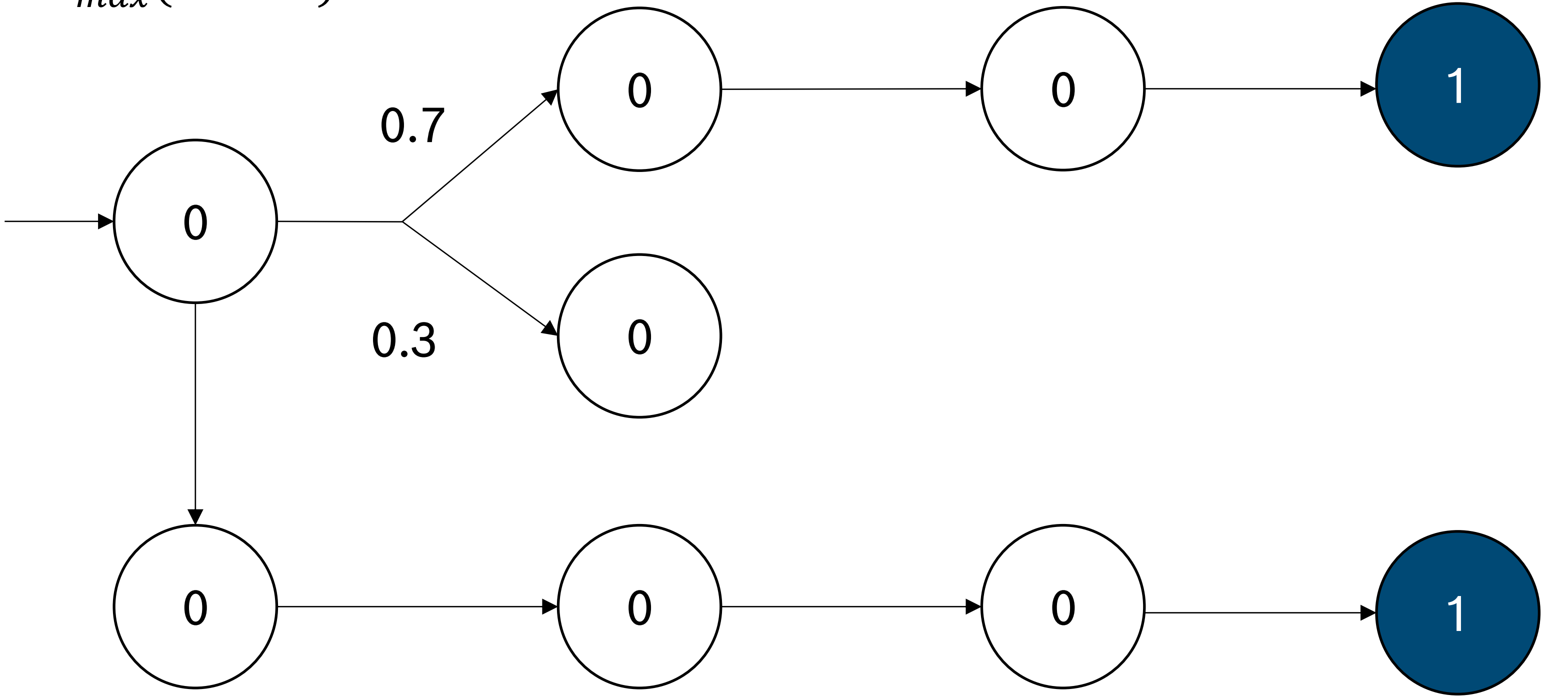
Property

%

ϵ-approximate

Probabilistic Model Checking
Value Iteration

$P_{max}(\diamond \text{Goal})$

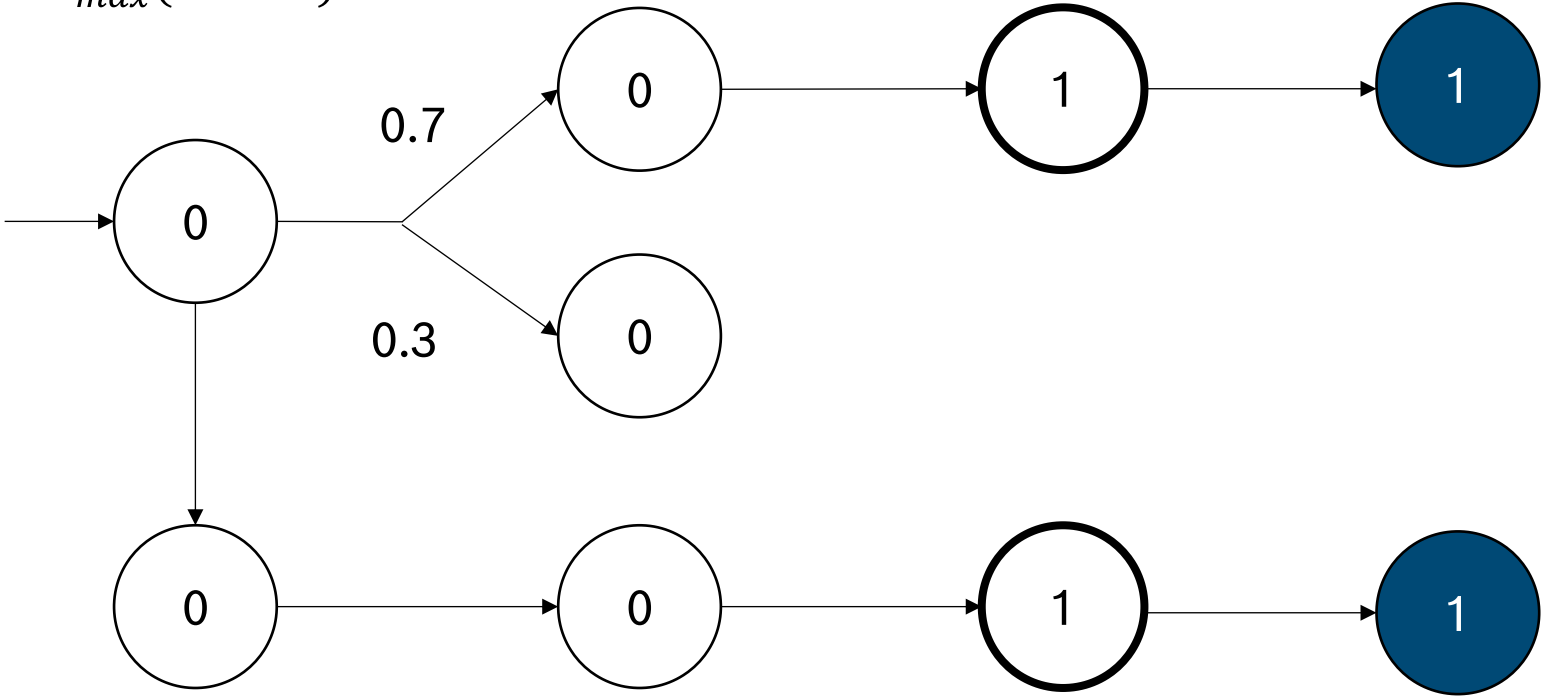


Bellman function: $v_{i+1}(s) = \max_{a \in A(s)} \Sigma_{s'} T(s,a,s') \cdot v_i(s')$

Probabilistic Model Checking

Value Iteration

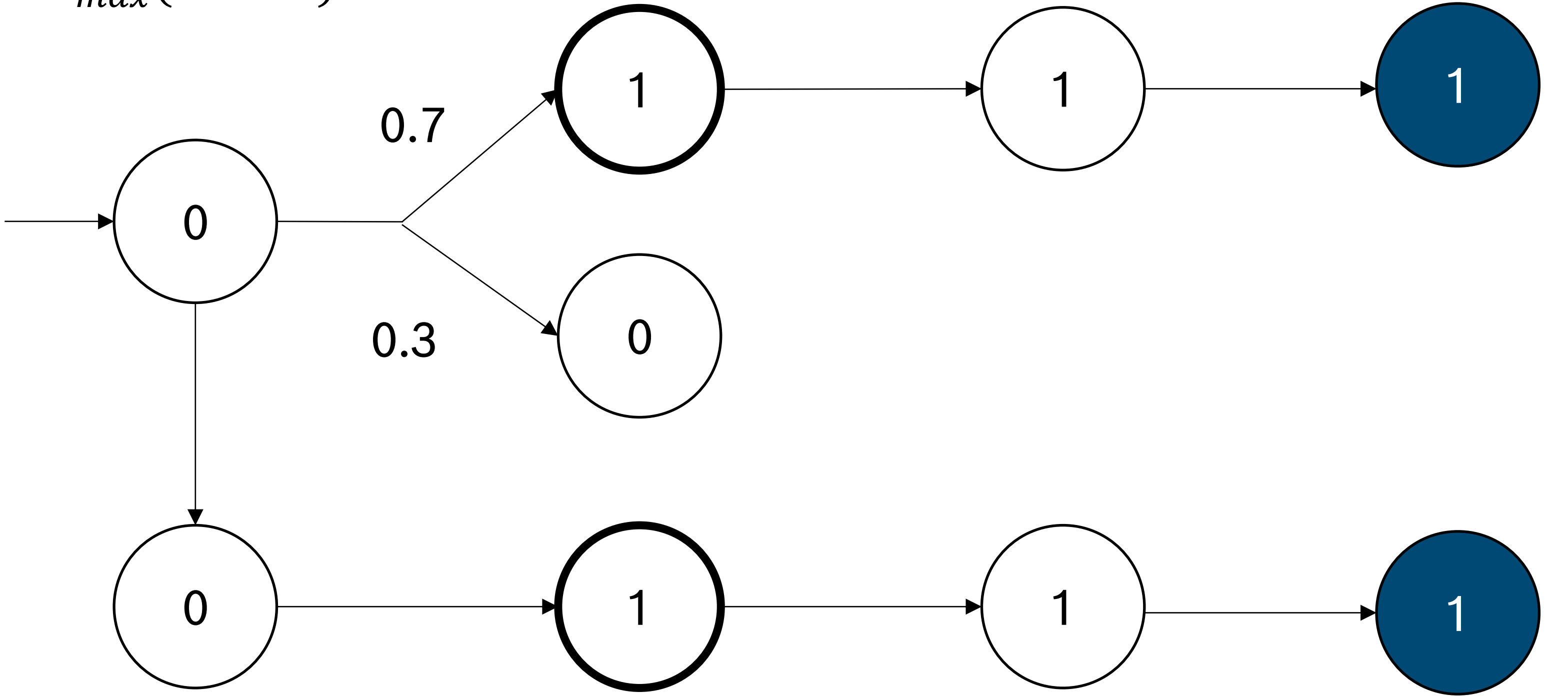
$P_{max}(\diamond \text{Goal})$



Bellman function: $v_{i+1}(s) = \max_{a \in A(s)} \Sigma_{s'} T(s, a, s') \cdot v_i(s')$

Probabilistic Model Checking
Value Iteration

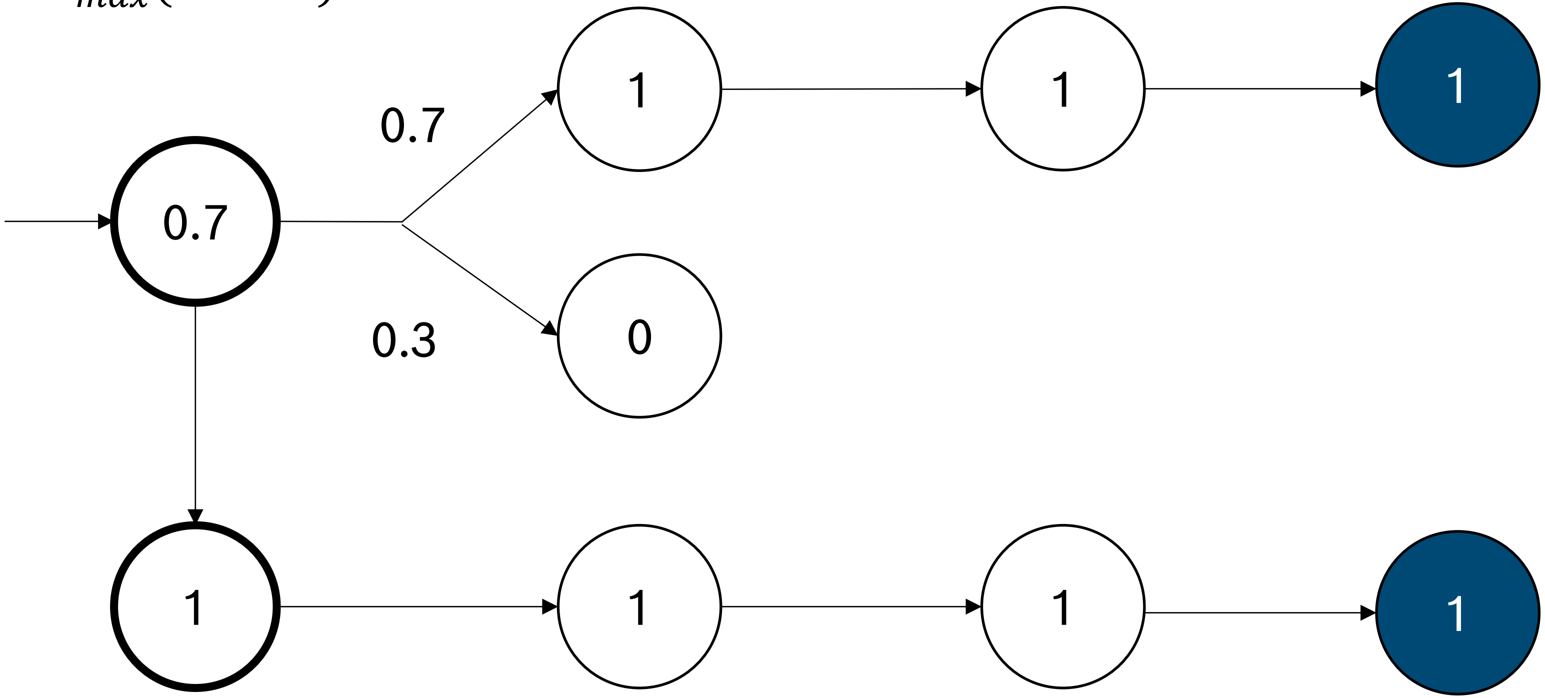
$P_{max}(\Diamond \text{Goal})$



Bellman function: $v_{i+1}(s) = \max_{a \in A(s)} \Sigma_{s'} T(s,a,s') \cdot v_i(s')$

Probabilistic Model Checking
Value Iteration

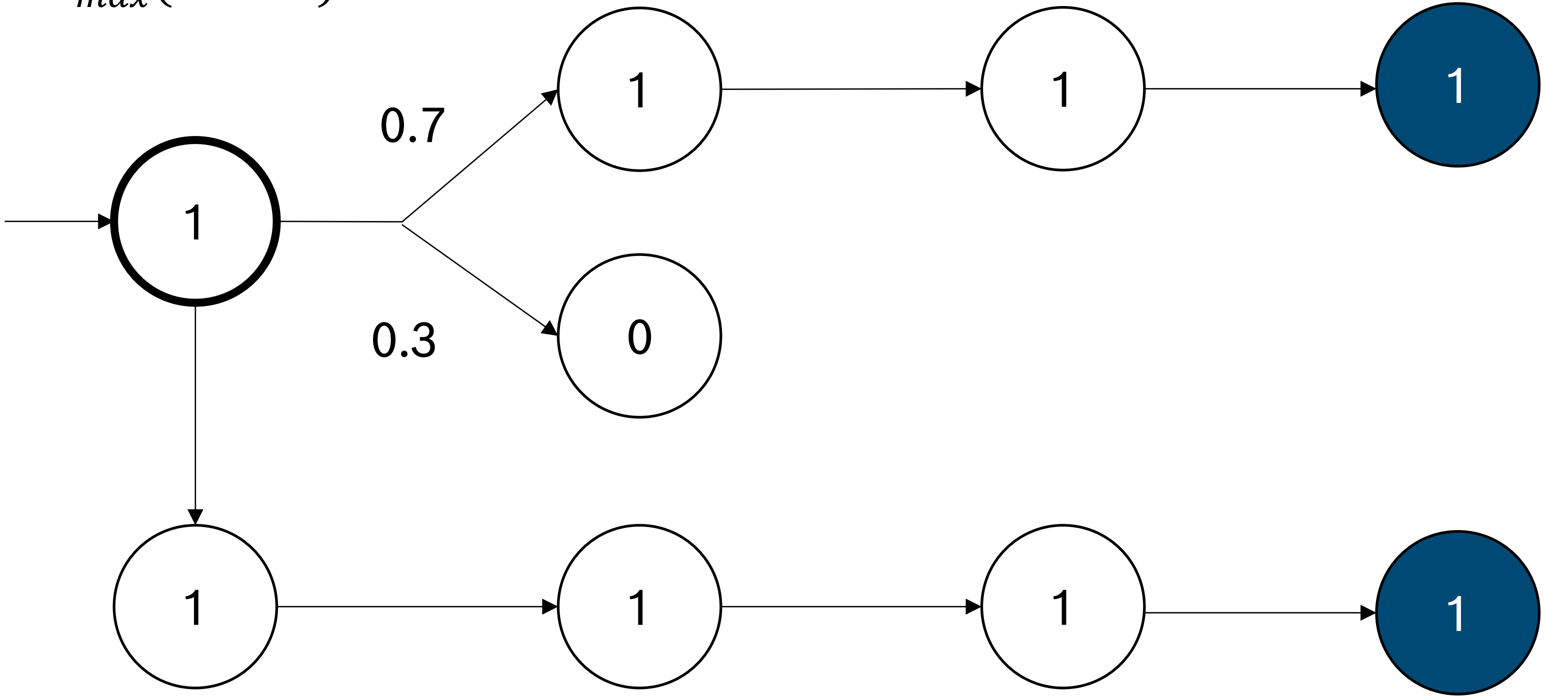
$P_{max}(\diamond \text{Goal})$



Bellman function: $v_{i+1}(s) = \max_{a \in A(s)} \Sigma_{s'} T(s,a,s') \cdot v_i(s')$

Probabilistic Model Checking
Value Iteration

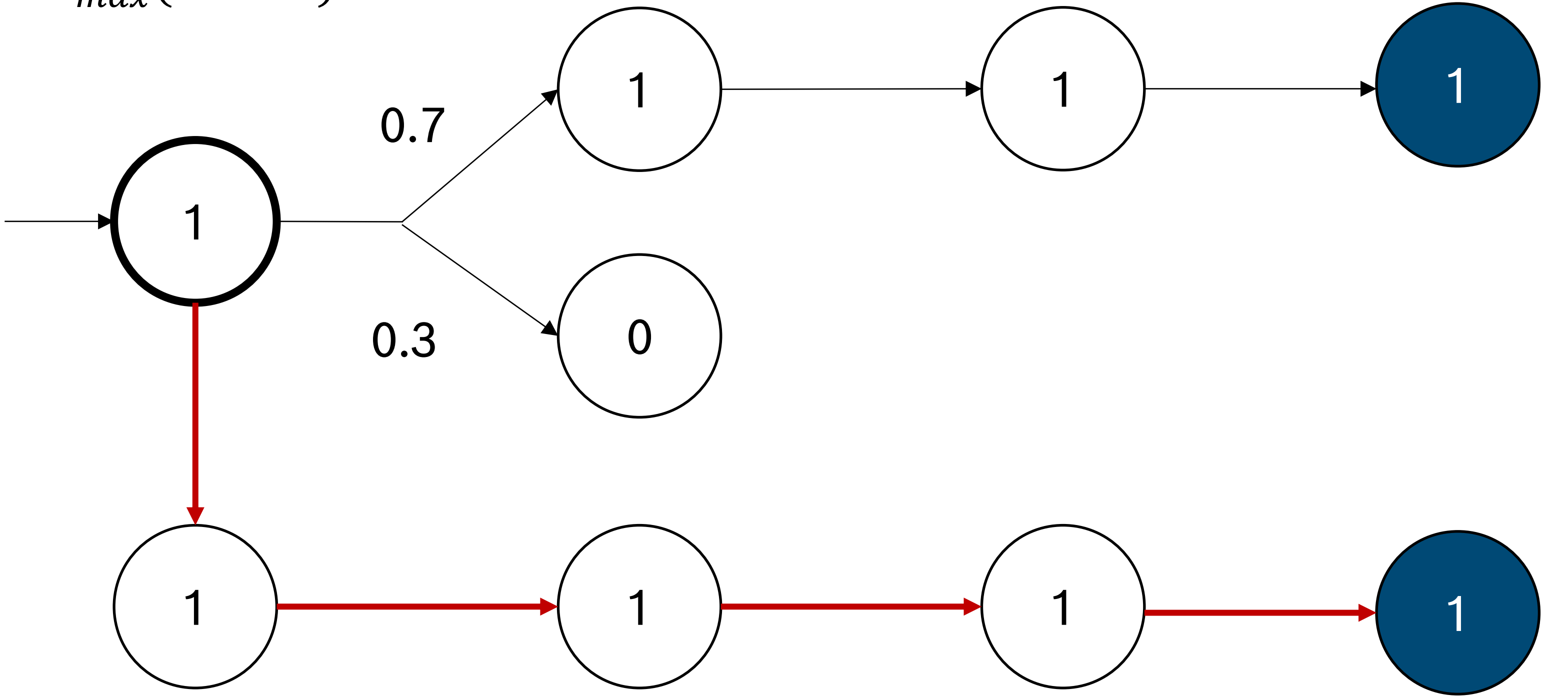
$P_{max}(\diamond \text{Goal})$



Bellman function: $v_{i+1}(s) = \max_{a \in A(s)} \Sigma_{s'} T(s,a,s') \cdot v_i(s')$

Probabilistic Model Checking
Value Iteration

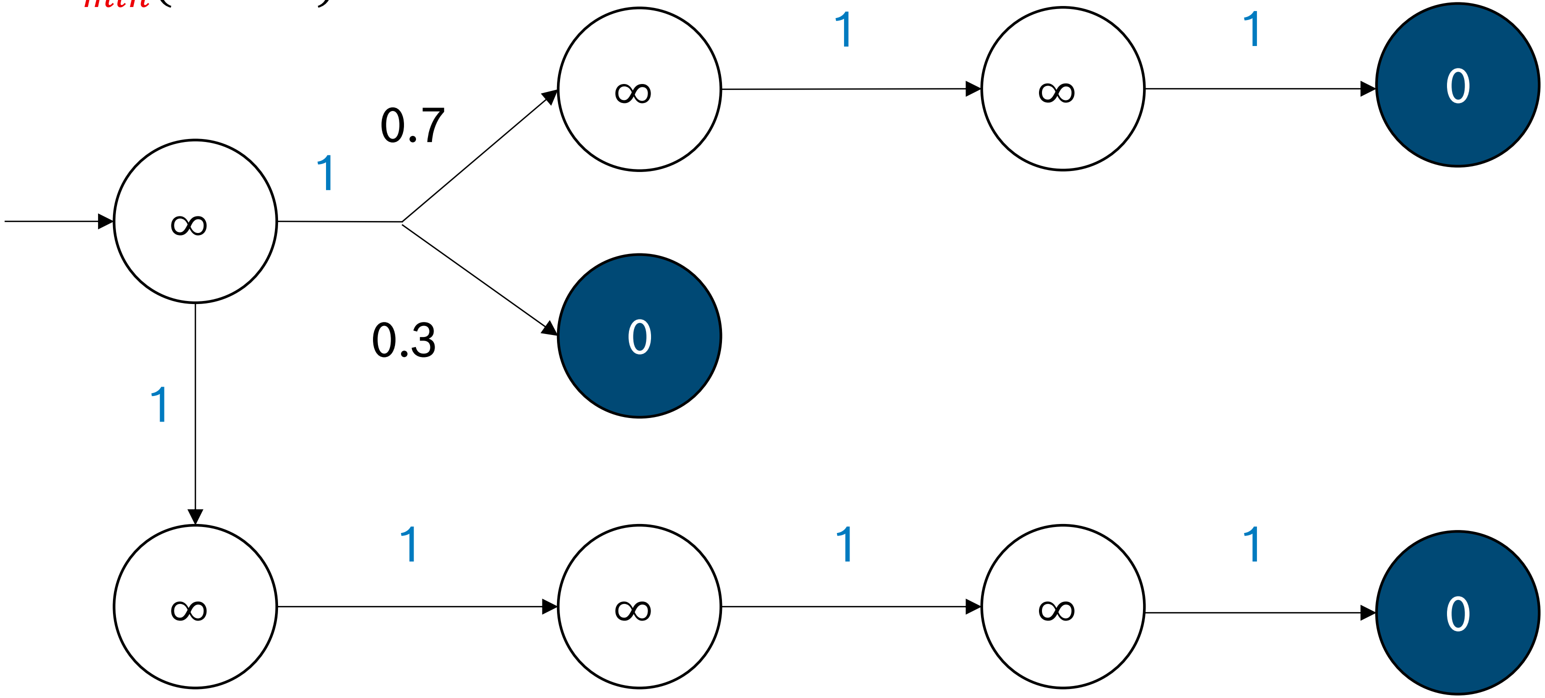
$P_{max}(\diamond \text{Goal})$



Bellman function: $v_{i+1}(s) = \max_{a \in A(s)} \Sigma_{s'} T(s,a,s') \cdot v_i(s')$

Probabilistic Model Checking
Value Iteration

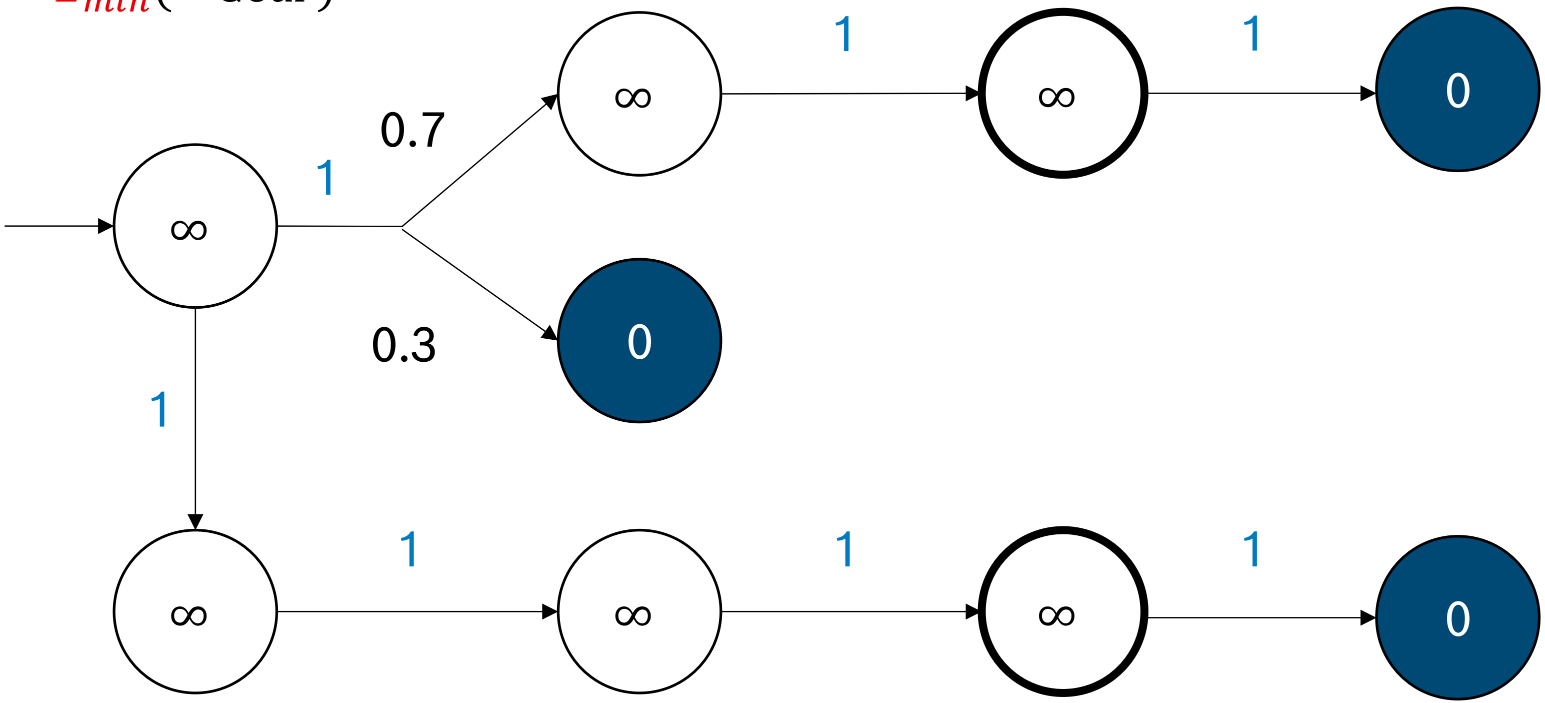
$E_{min}(\Diamond \text{Goal})$



Bellman function: $v_{i+1}(s) = min_{a \in A(s)} \Sigma_{s'} T(s,a,s') \cdot (v_i(s') + R(s,a,s'))$

Probabilistic Model Checking
Value Iteration

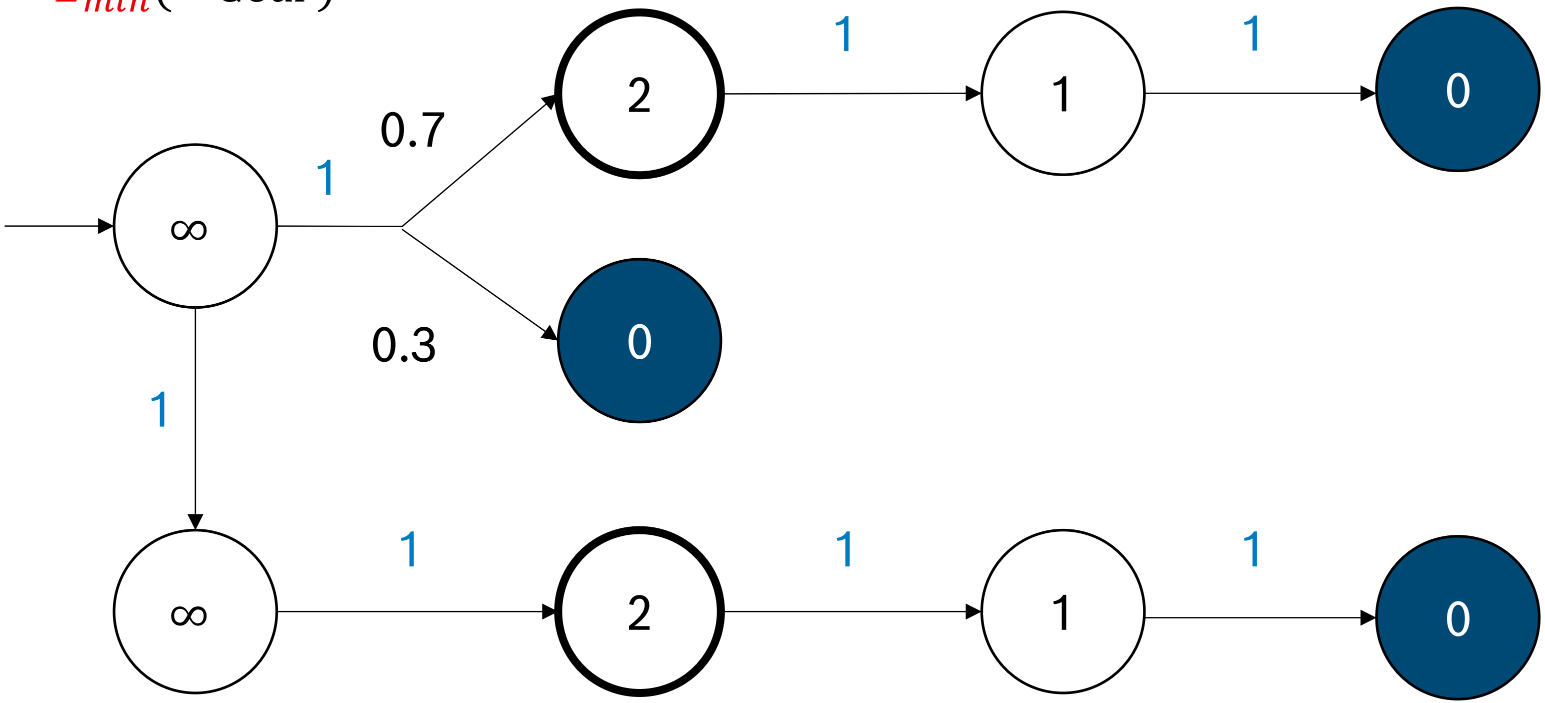
$E_{min}(\diamond \text{Goal})$



Bellman function: $v_{i+1}(s) = min_{a \in A(s)} \Sigma_{s'} T(s,a,s') \cdot (v_i(s') + R(s,a,s'))$

Probabilistic Model Checking
Value Iteration

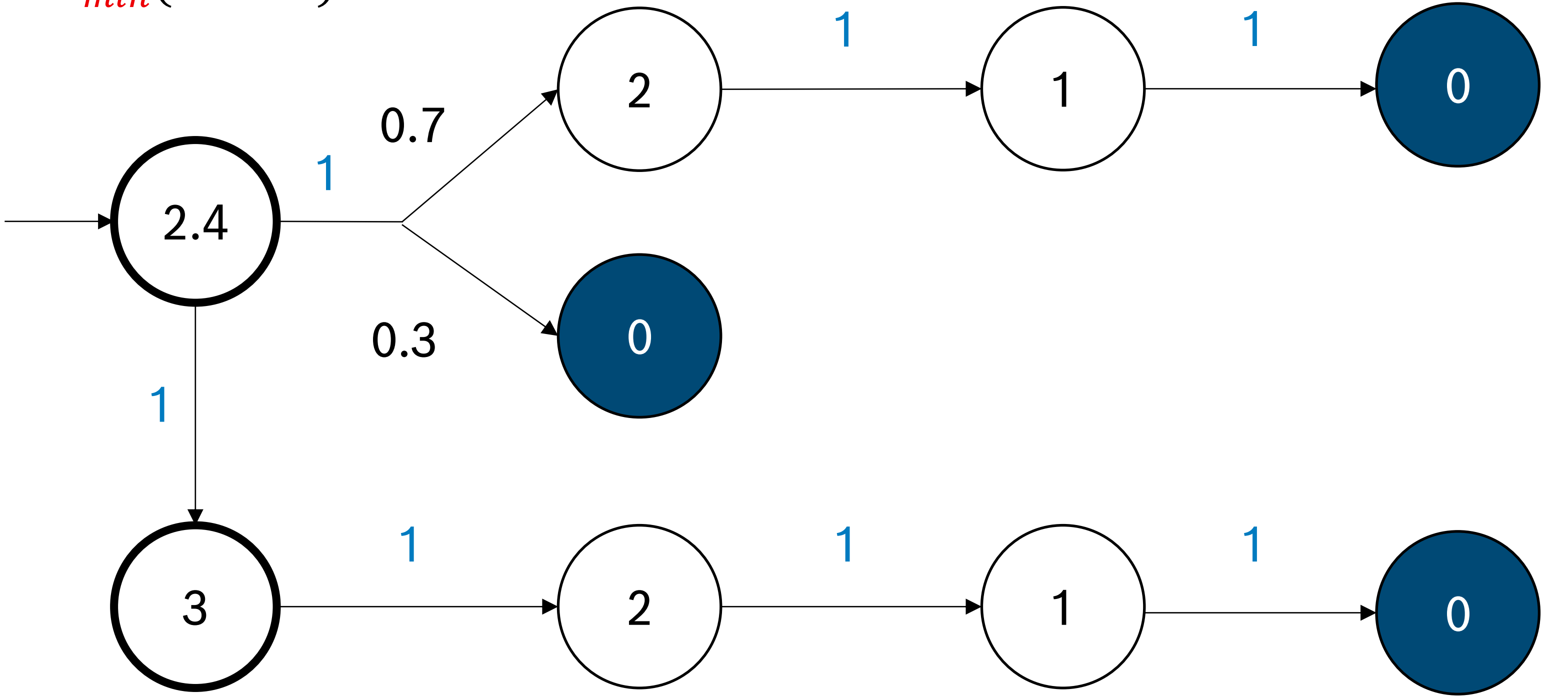
$E_{min}(\diamond \text{Goal})$



Bellman function: $v_{i+1}(s) = min_{a \in A(s)} \Sigma_{s'} T(s,a,s') \cdot (v_i(s') + R(s,a,s'))$

Probabilistic Model Checking
Value Iteration

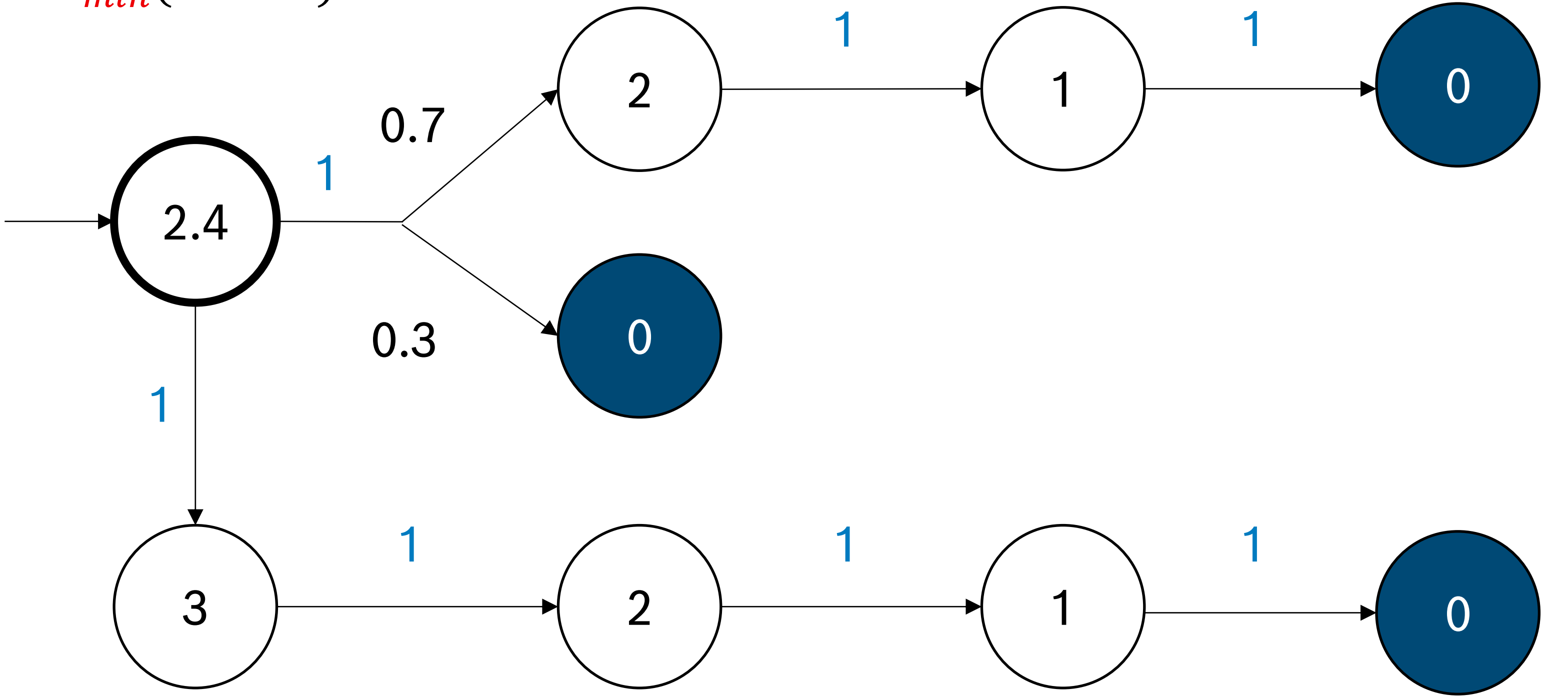
$E_{min}(\Diamond \text{ Goal })$



Bellman function: $v_{i+1}(s) = min_{a \in A(s)} \Sigma_{s'} T(s,a,s') \cdot (v_i(s') + R(s,a,s'))$

Probabilistic Model Checking
Value Iteration

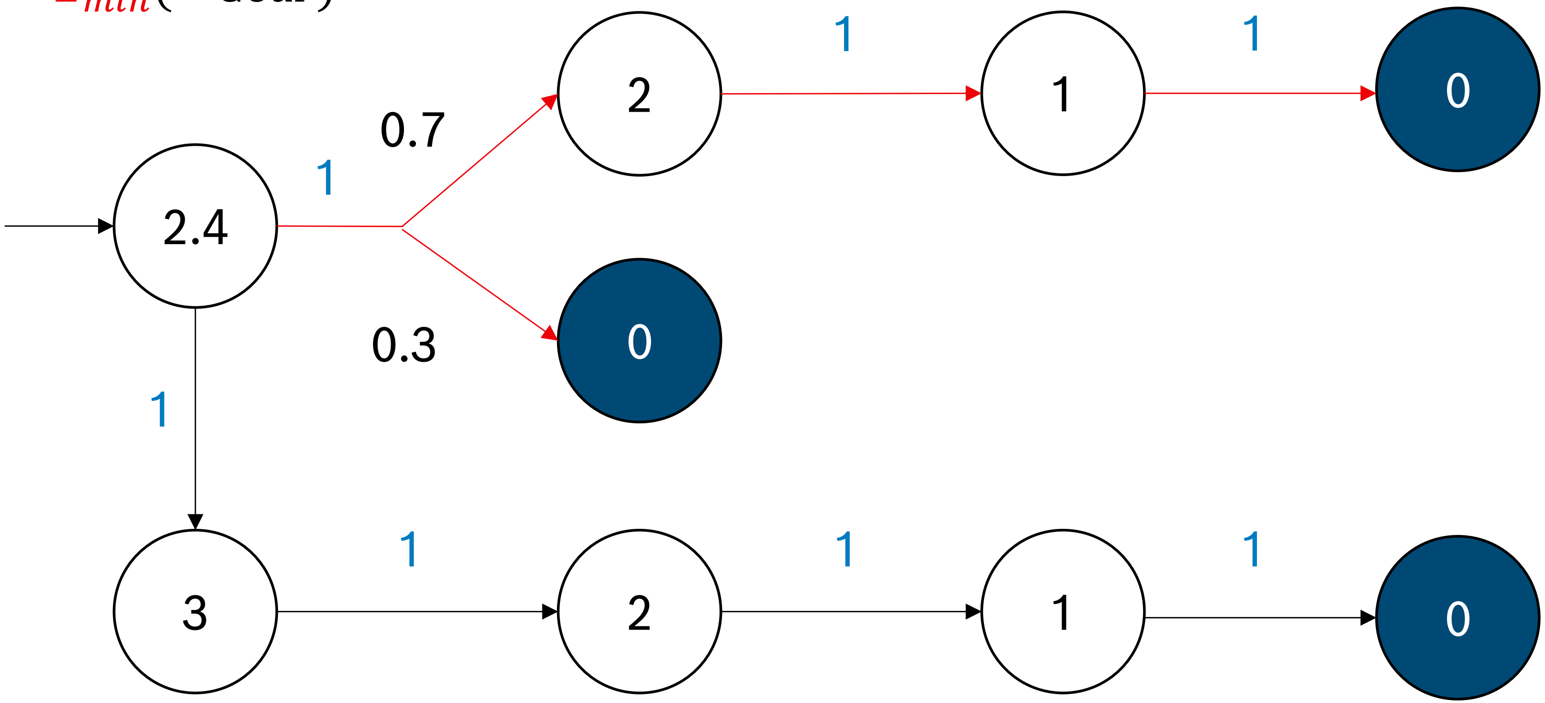
$E_{min}(\Diamond \text{Goal})$



Bellman function: $v_{i+1}(s) = min_{a \in A(s)} \Sigma_{s'} T(s,a,s') \cdot (v_i(s') + R(s,a,s'))$

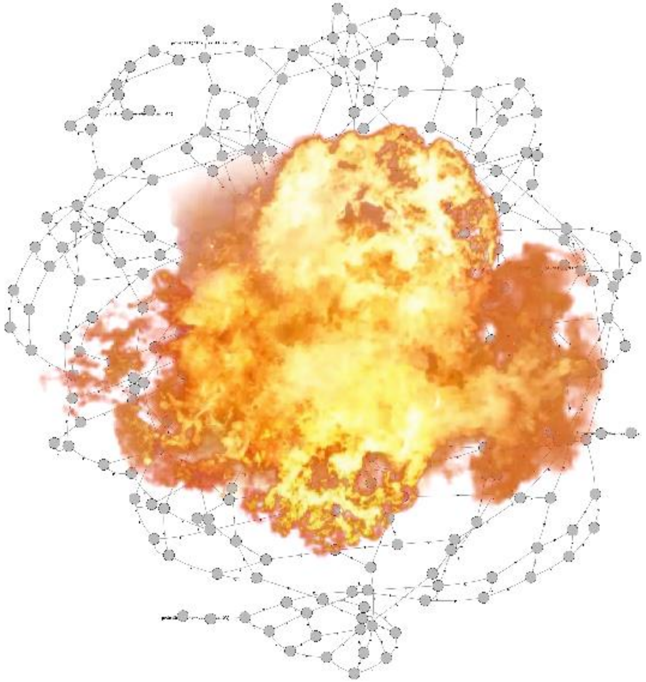
Probabilistic Model Checking
Value Iteration

$E_{min}(\diamond \text{Goal})$

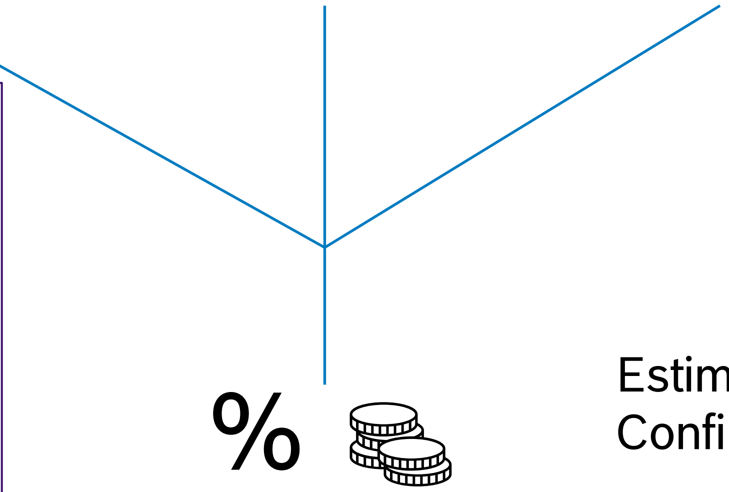


Bellman function: $v_{i+1}(s) = min_{a \in A(s)} \Sigma_{s'} T(s,a,s') \cdot (v_i(s') + R(s,a,s'))$

Probabilistic Model Checking
State Space Explosion Problem



Statistical Model Checking
Introduction



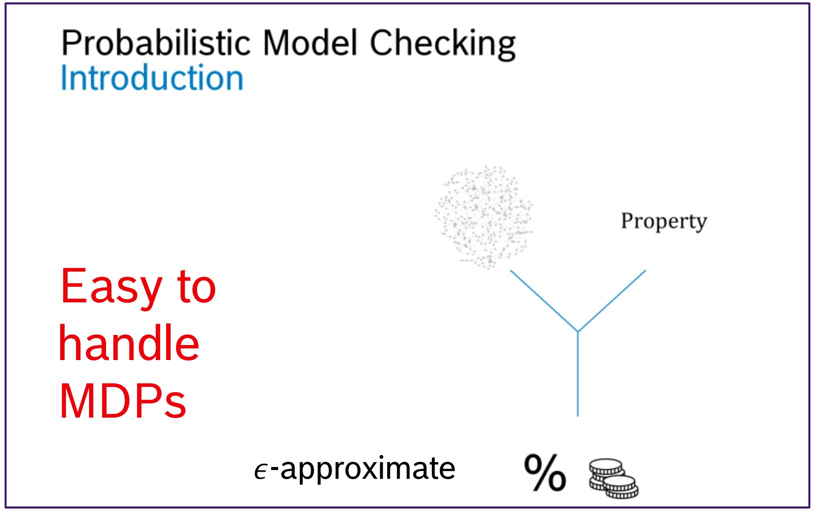
Property

Strategy σ to resolve nondeterminism

At first, only made for DTMCs

Estimator with statistical guarantees:
Confindence Intervall:

$$P(|smc_\sigma - opti_\sigma| > \epsilon) < \delta$$



Probabilistic Model Checking
Introduction

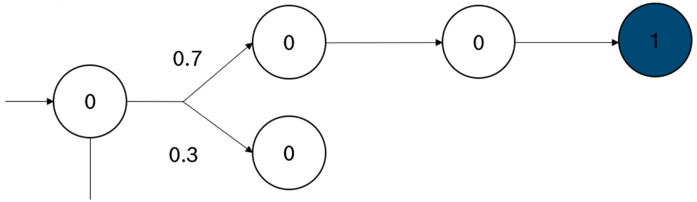
Property

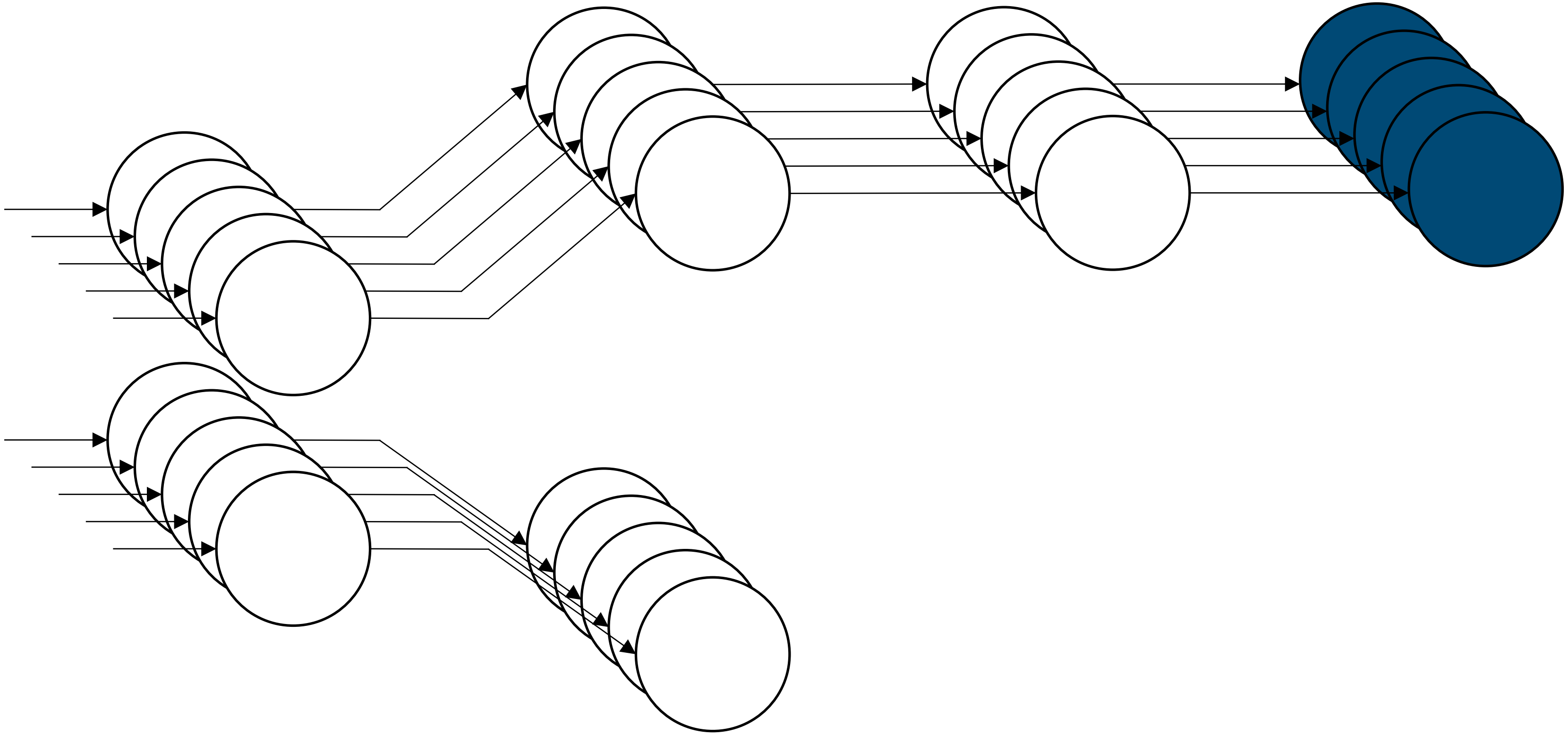
Easy to handle MDPs

ϵ-approximate

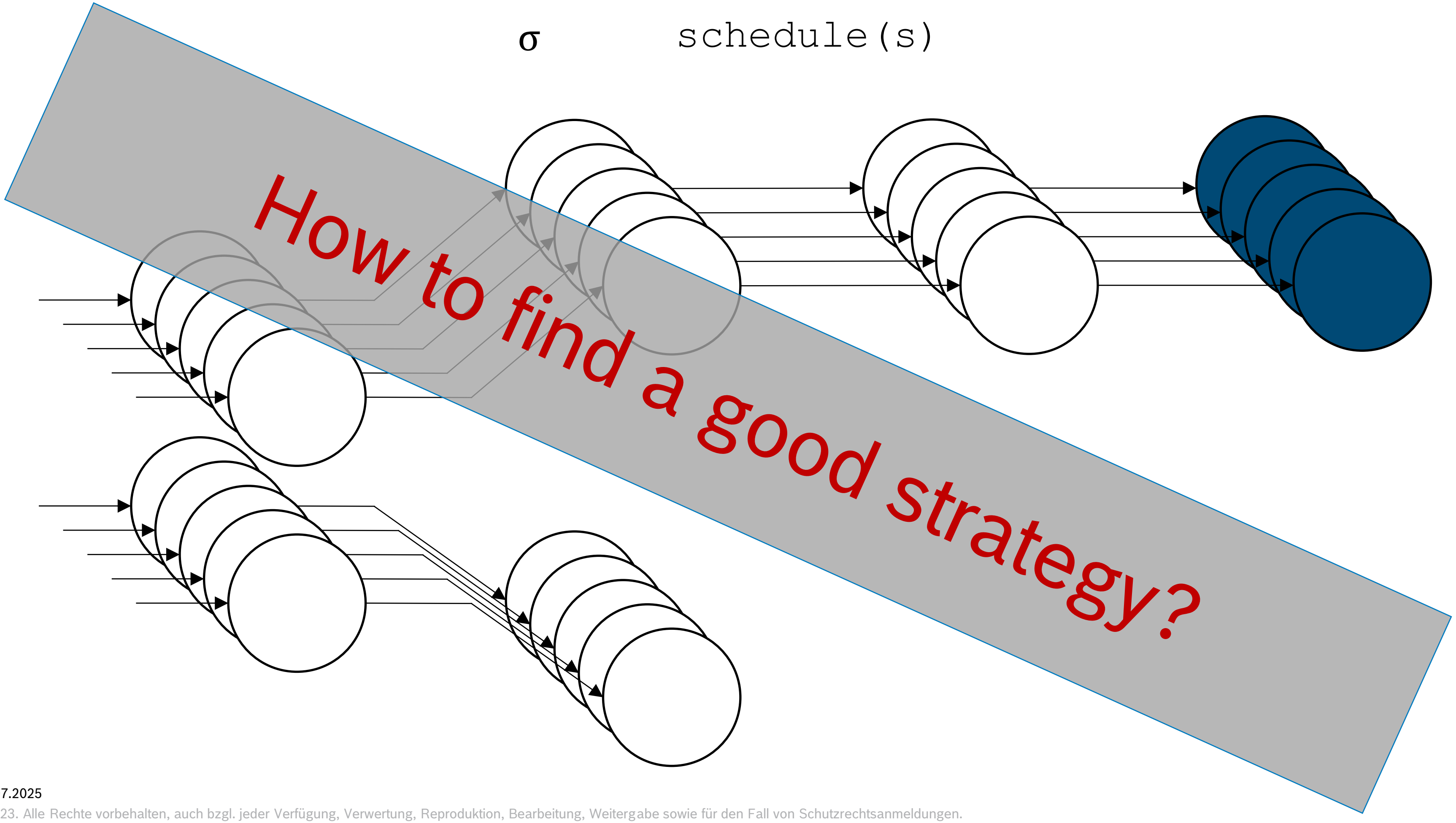
Statistical Model Checking
Monte Carlo Simulation

schedule(s) σ



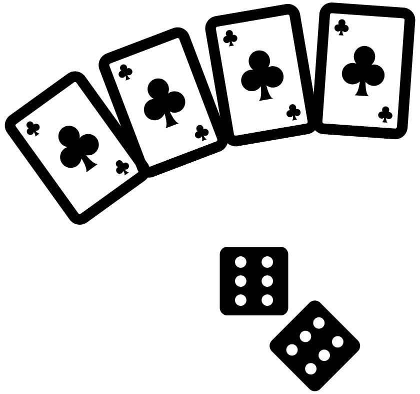


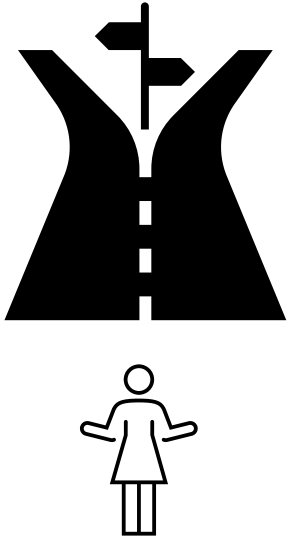
Statistical Model Checking
Schedulers

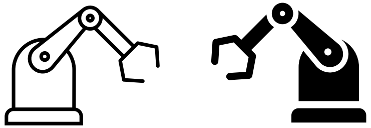


How to find a good strategy?

Statistical Model Checking
Schedulers

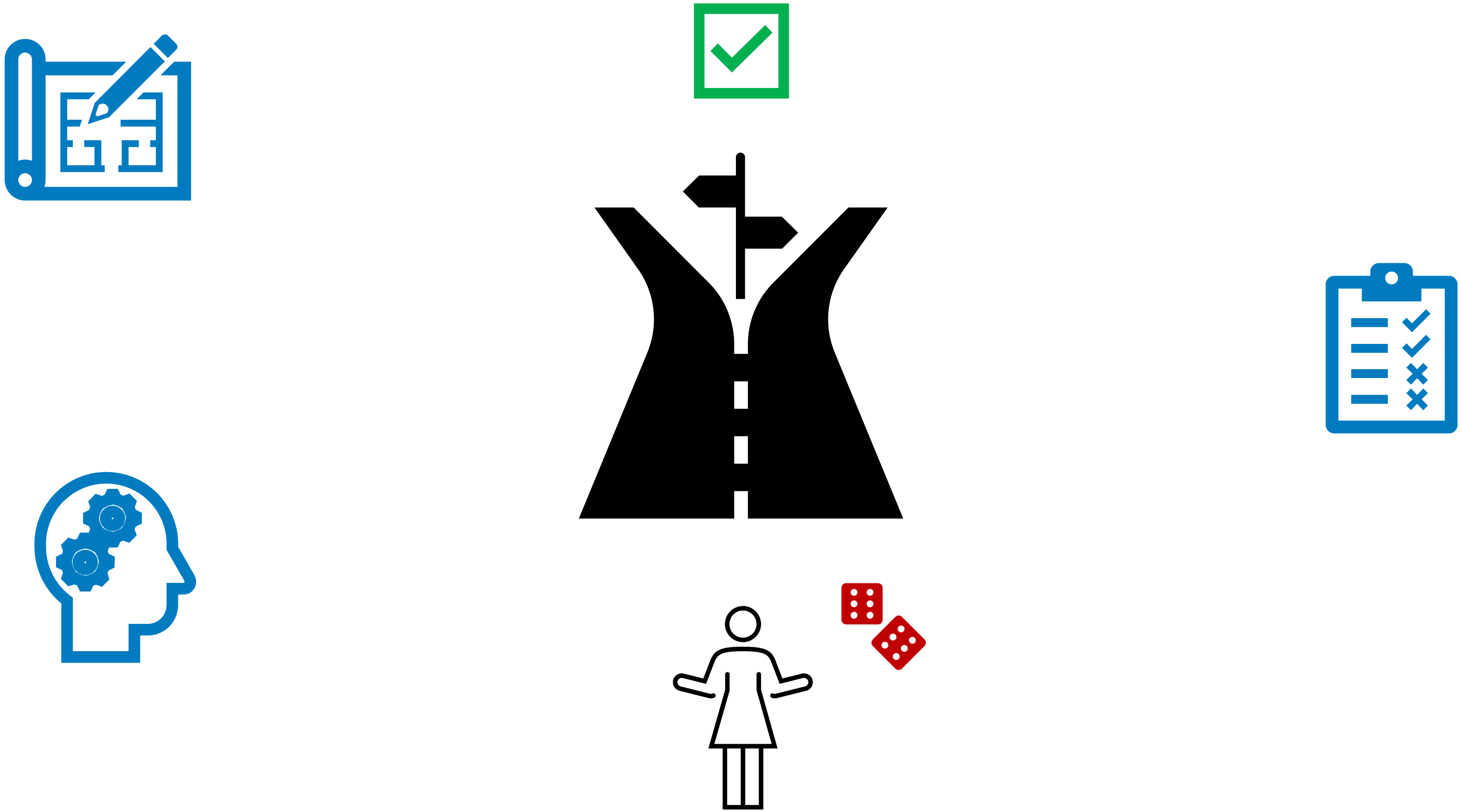






optimal, safe, performant strategies

Statistical Model Checking
Motivation



Statistical Model Checking
How to Find Good Schedulers?

- Lightweight scheduler sampling (LSS) lifts SMC from DTMCs to MDPs
- Picks set of strategies & applies SMC-based heuristic to find best strategy

What about rare events?

- Importance splitting & importance sampling

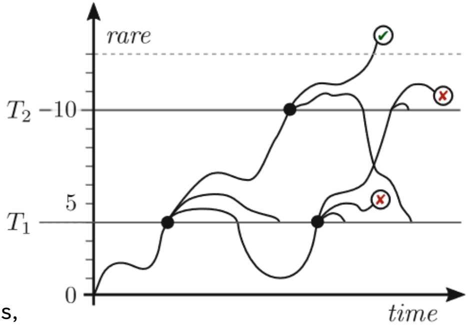


Fig. 3 Illustration of RESTART [9]

Source: An efficient statistical model checker for nondeterminism and rare events, Carlos Budde, Pedro D'Argenio, Arnd Hartmanns, Sean Sedwards, STTT 2020.

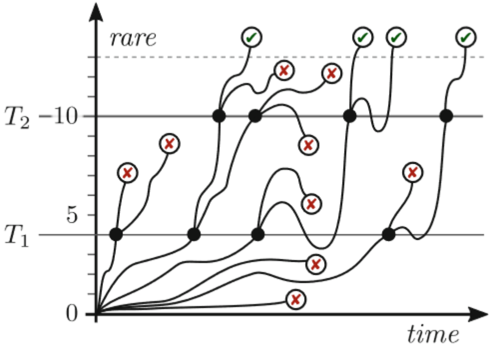


Fig. 5 Illustration of fixed success [9]

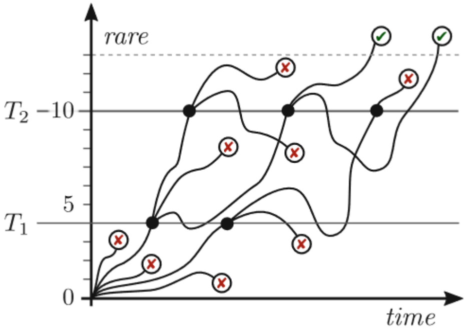


Fig. 4 Illustration of fixed effort [9]

Other Model Checking Methods
Overview

- Symbolic model checking
  - States and transition relations represented as logical formulas, e.g., in BDDs
  - Set of states and transition relations:
    Boolean formulas over vector of current state variables and next state variables
  - Model checking: mathematical reasoning over those formulas
  - Compute set of reachable states in i steps, search for fixpoint

- LTL model checking
  - Build product of TS and Büchi automaton of negated LTL formula to verify
  - Check if product automaton satisfies persistance property

Modeling Formats & Languages

JANI,
SCXML

JANI
Model Checking Format

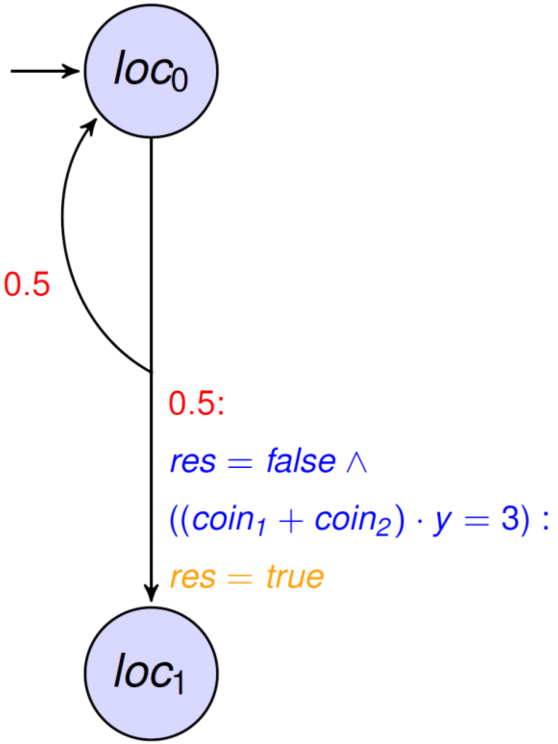
- Model exchange format for networks of quantitative automata
- Based on JSON
- Properties: temporal formulas (CTL)
- Foster verification tool interoperation & comparability
- Supported by most state-of-the-art model checkers:
  The Modest Toolset, Storm, PRISM (via converter), …

Specification: https://jani-spec.org/

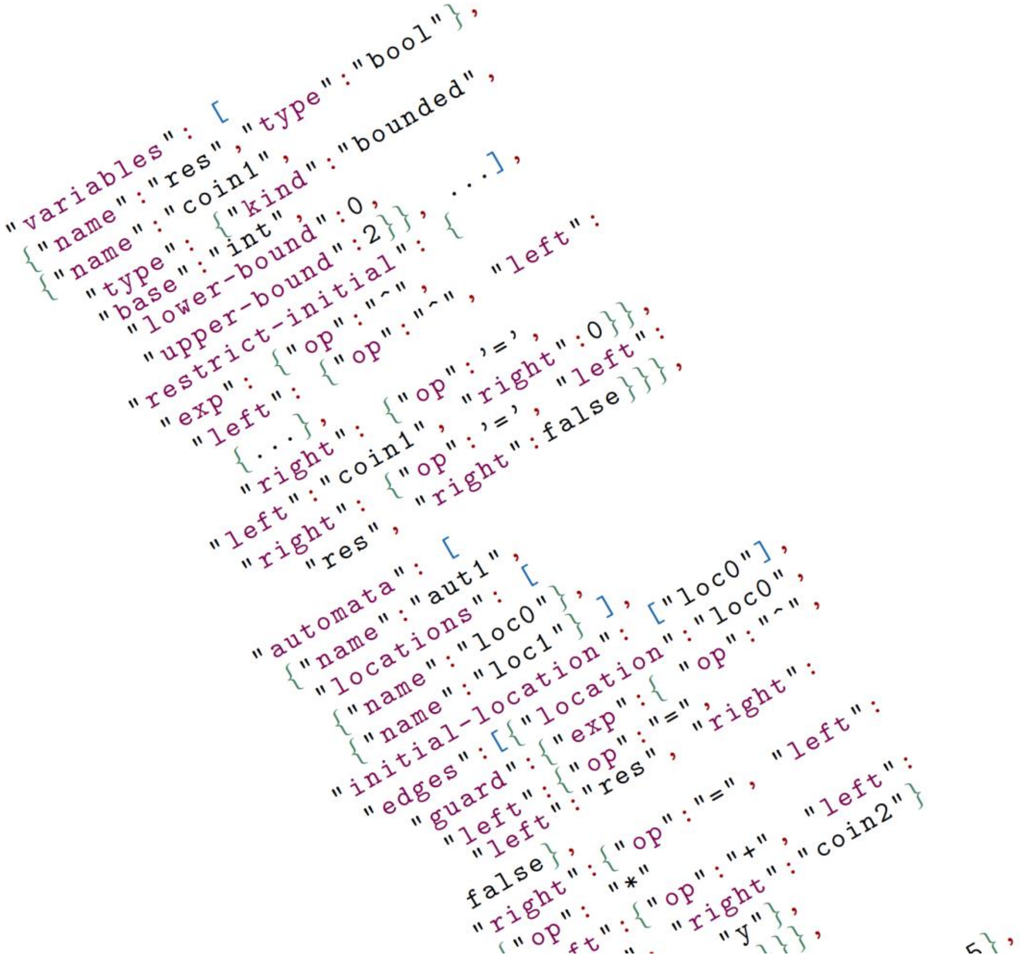
Benchmark Set:
https://qcomp.org/benchmarks/

JANI
Example

Example: Automaton "aut_1":

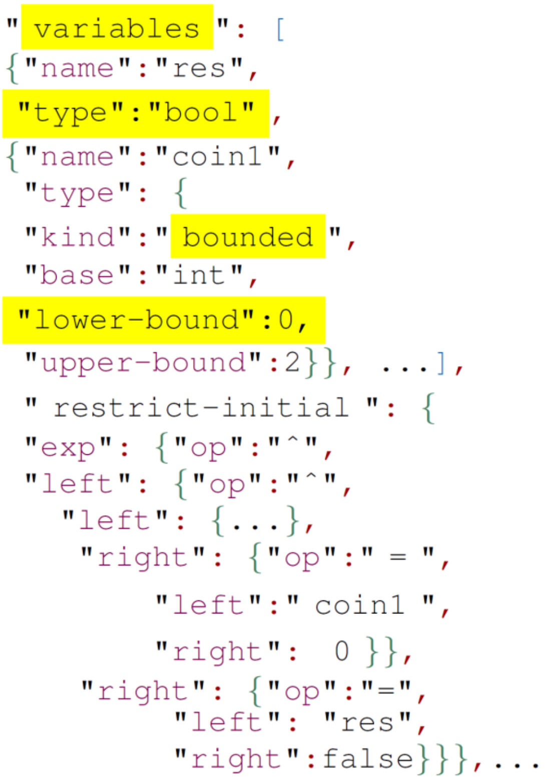


JANI
Example

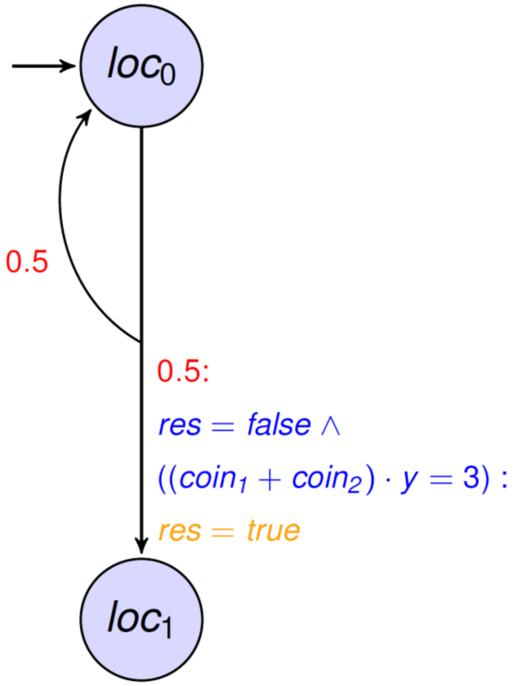


```
"variables": [
  {"name":"res","type":"bool"},
  {"name":"coin1",
    "type": {"kind":"bounded",
      "base":"int",
      "lower-bound":0,
      "upper-bound":2}}, ...],
"restrict-initial": {
  "exp": {"op":"^",
    "left": {"op":"^", "left":
      {...},
      "right": {"op":'=', "right":0}},
    "left":"coin1", "left":
    "right": {"op":'=', "right":false}}},
      "res",

"automata": [
  {"name":"aut1",
    "locations": [
      {"name":"loc0"},
      {"name":"loc1"} ],
    "initial-location": ["loc0"],
    "edges":[{"location":"loc0",
      "guard":{"exp":{ "op":"^",
        "left":{"op":"=", "right":
        "left":"res", "left":
        false}, {"op":"=", "left":
        "right":{"op":"+", "left":"coin2"}
        {"op": "*"
          ft":{"op":"+", "right":
              "y"},
              }}},
                                  5},
```

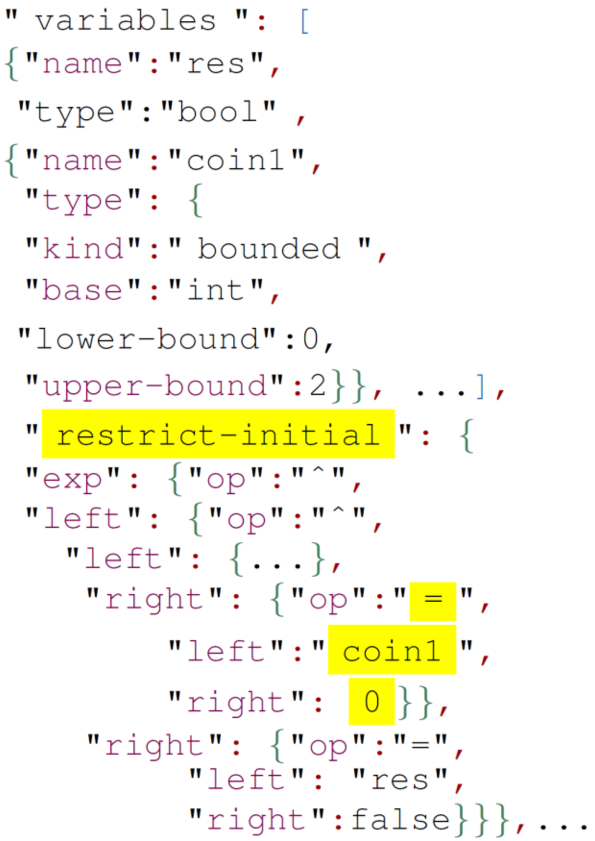
JANI
Example



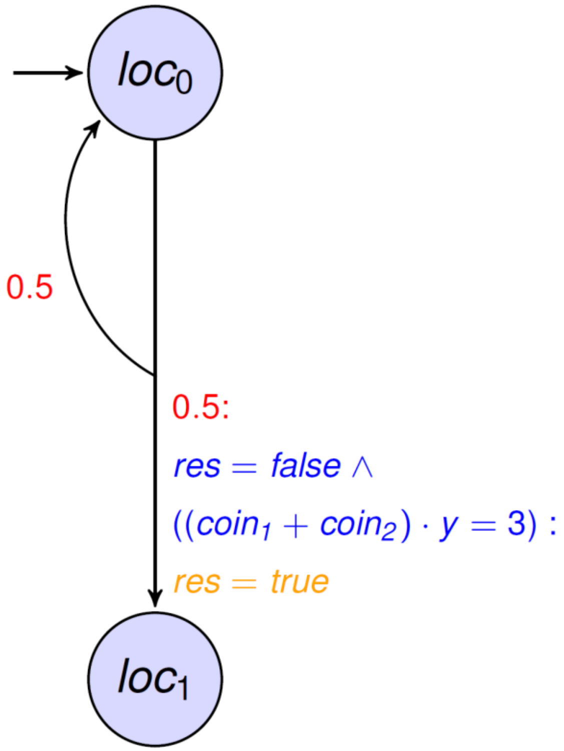
```
" variables ": [
{"name":"res",
 "type":"bool" ,
{"name":"coin1",
 "type": {
 "kind":" bounded ",
 "base":"int",
 "lower-bound":0,
 "upper-bound":2}}, ...],
 " restrict-initial ": {
 "exp": {"op":"^",
 "left": {"op":"^",
   "left": {...},
    "right": {"op":" = ",
        "left":" coin1 ",
        "right":  0 }},
    "right": {"op":"=",
         "left": "res",
         "right":false}}},...
```



JANI
Example

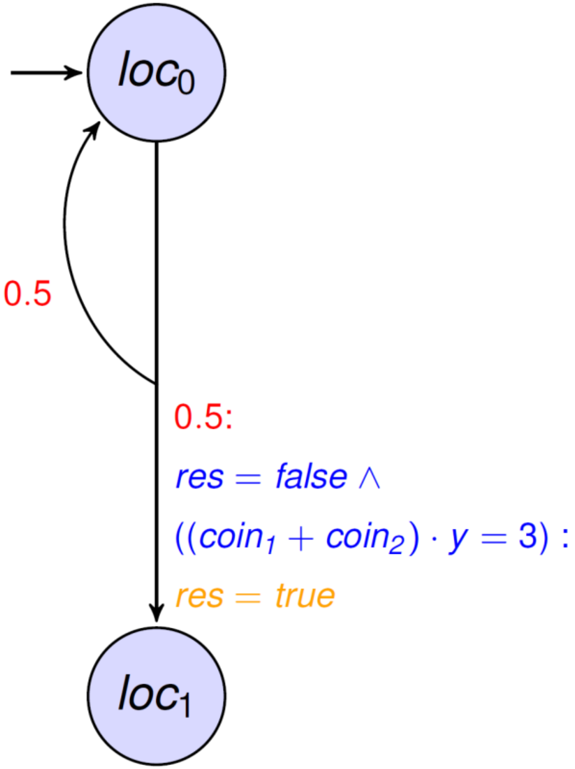


```
" variables ": [
{"name":"res",
 "type":"bool" ,
{"name":"coin1",
  "type": {
  "kind":" bounded ",
  "base":"int",
 "lower-bound":0,
  "upper-bound":2}}, ...],
  " restrict-initial ": {
  "exp": {"op":"^",
  "left": {"op":"^",
     "left": {...},
       "right": {"op":" = ",
            "left":" coin1 ",
            "right": 0 }},
       "right": {"op":"=",
              "left": "res",
              "right":false}}},...
```

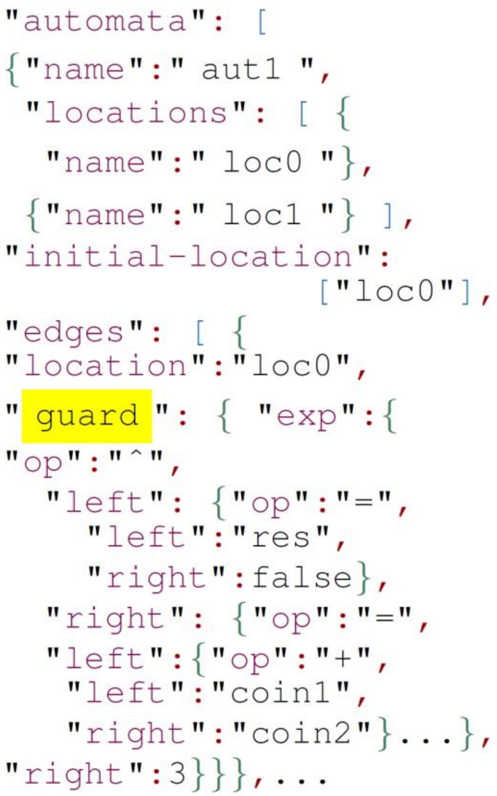


JANI
Example

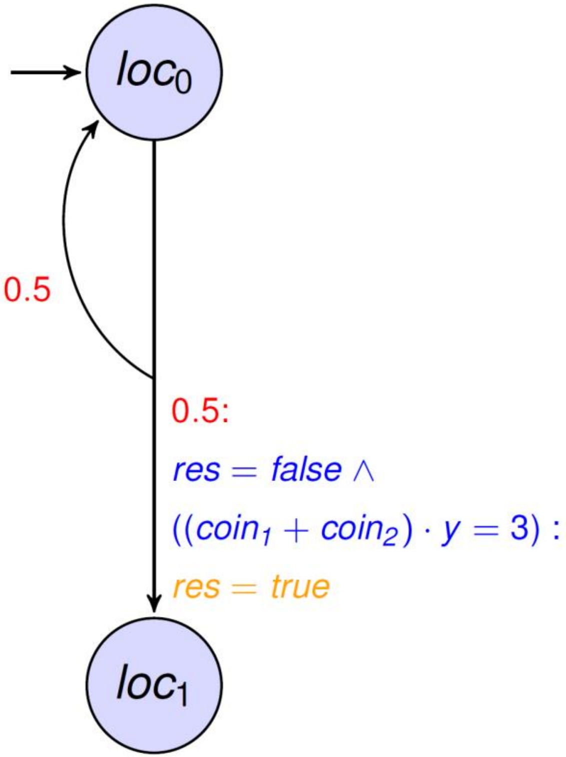
```
"automata": [
{"name":" aut1 ",
 "locations": [ {
  "name":" loc0 "},
 {"name":" loc1 "} ],
"initial-location":
               ["loc0"],
"edges": [ {
"location":"loc0",
" guard ": { "exp":{
"op":"^",
  "left": {"op":"=",
    "left":"res",
    "right":false},
  "right": {"op":"=",
  "left":{"op":"+",
   "left":"coin1",
   "right":"coin2"}...},
"right":3}}},...
```



JANI
Example

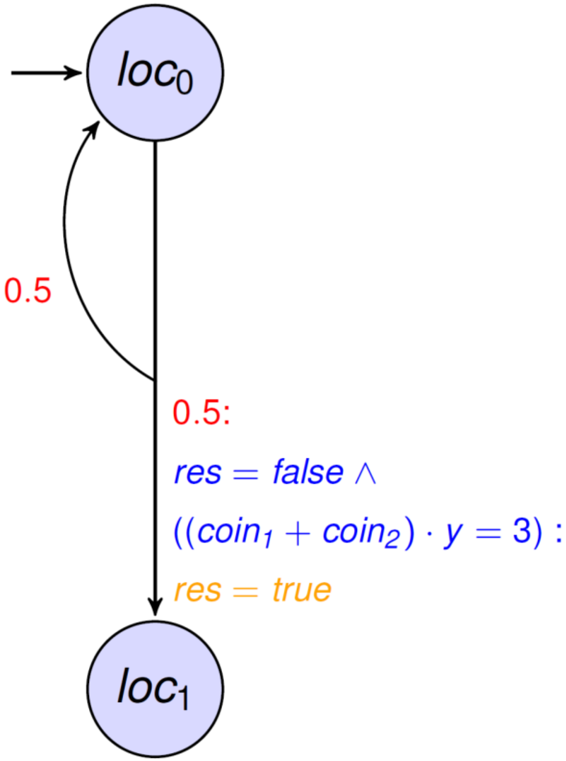


```
"automata": [
{"name":" aut1 ",
 "locations": [ {
  "name":" loc0 "},
 {"name":" loc1 "} ],
"initial-location":
                ["loc0"],
"edges": [ {
"location":"loc0",
" guard ": { "exp":{
"op":"^",
  "left": {"op":"=",
    "left":"res",
    "right":false},
  "right": {"op":"=",
  "left":{"op":"+",
   "left":"coin1",
   "right":"coin2"}...},
"right":3}}},...
```

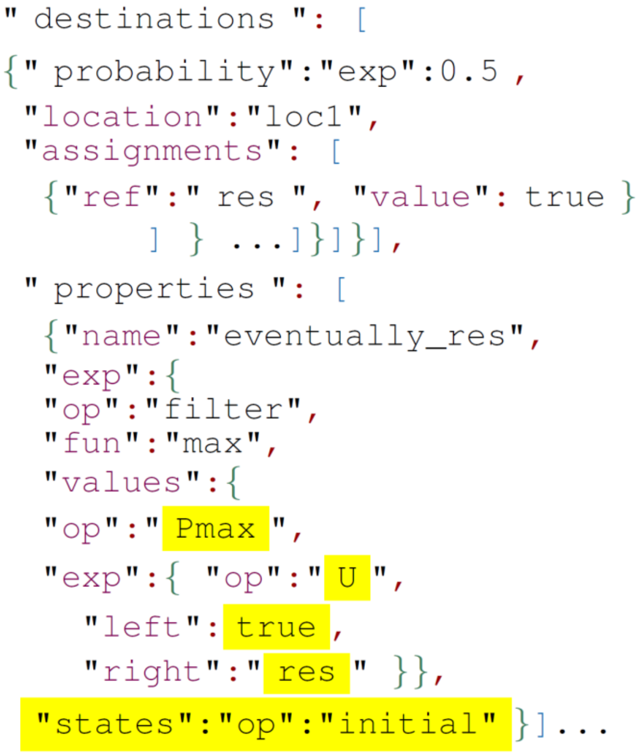


JANI
Example

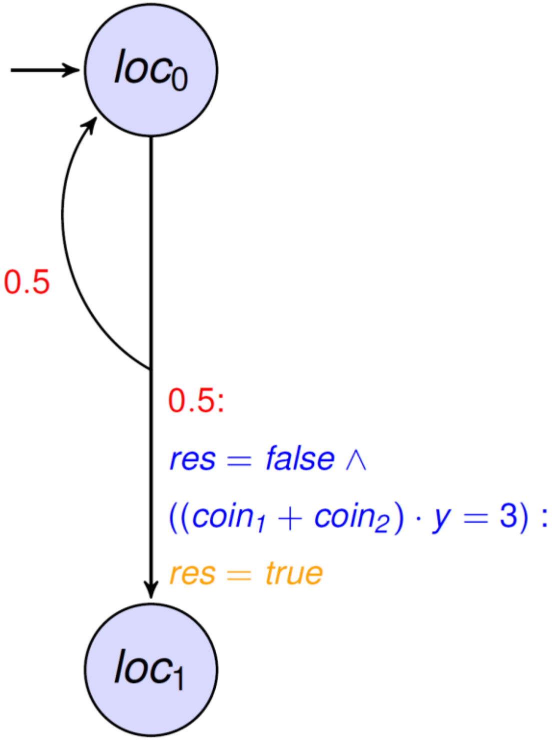
```
" destinations ": [
{" probability":"exp":0.5 ,
 "location":"loc1",
 "assignments": [
  {"ref":" res ", "value": true }
       ] } ...]}]}],
 " properties ": [
  {"name":"eventually_res",
  "exp":{
  "op":"filter",
  "fun":"max",
  "values":{
  "op":" Pmax ",
  "exp":{ "op":" U ",
    "left": true ,
    "right":" res " }},
  "states":"op":"initial" }]...
```



JANI
Example



```
" destinations ": [
{" probability":"exp":0.5 ,
 "location":"loc1",
 "assignments": [
  {"ref":" res ", "value": true }
       ] } ...]}]}],
 " properties ": [
  {"name":"eventually_res",
  "exp":{
  "op":"filter",
  "fun":"max",
  "values":{
  "op":" Pmax ",
  "exp":{ "op":" U ",
    "left": true ,
    "right":" res " }},
 "states":"op":"initial" }]...
```



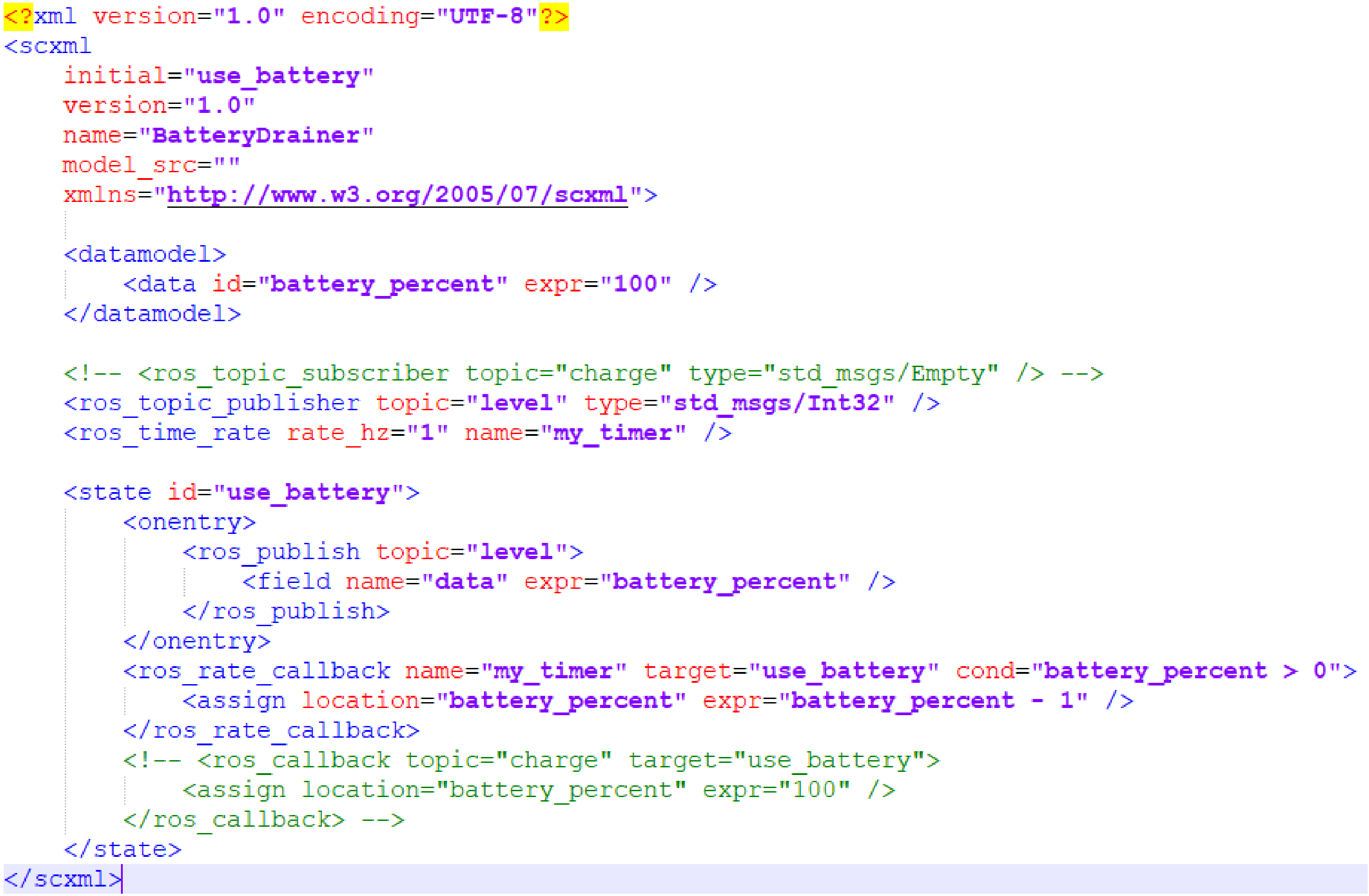
SCXML
Starting from a Running Robotic System

- SCXML: State Chart eXtensible Markup Language
- Event-based state machine language
- Genericizes state diagram notations used in other XML contexts
- Large tool support: SCXML to C++ compiler, Java & Python libraries to parse & execute SCXML programs, ...
- Compatible with other XML specifications already available for robotic systems

Standard: https://www.w3.org/TR/scxml/

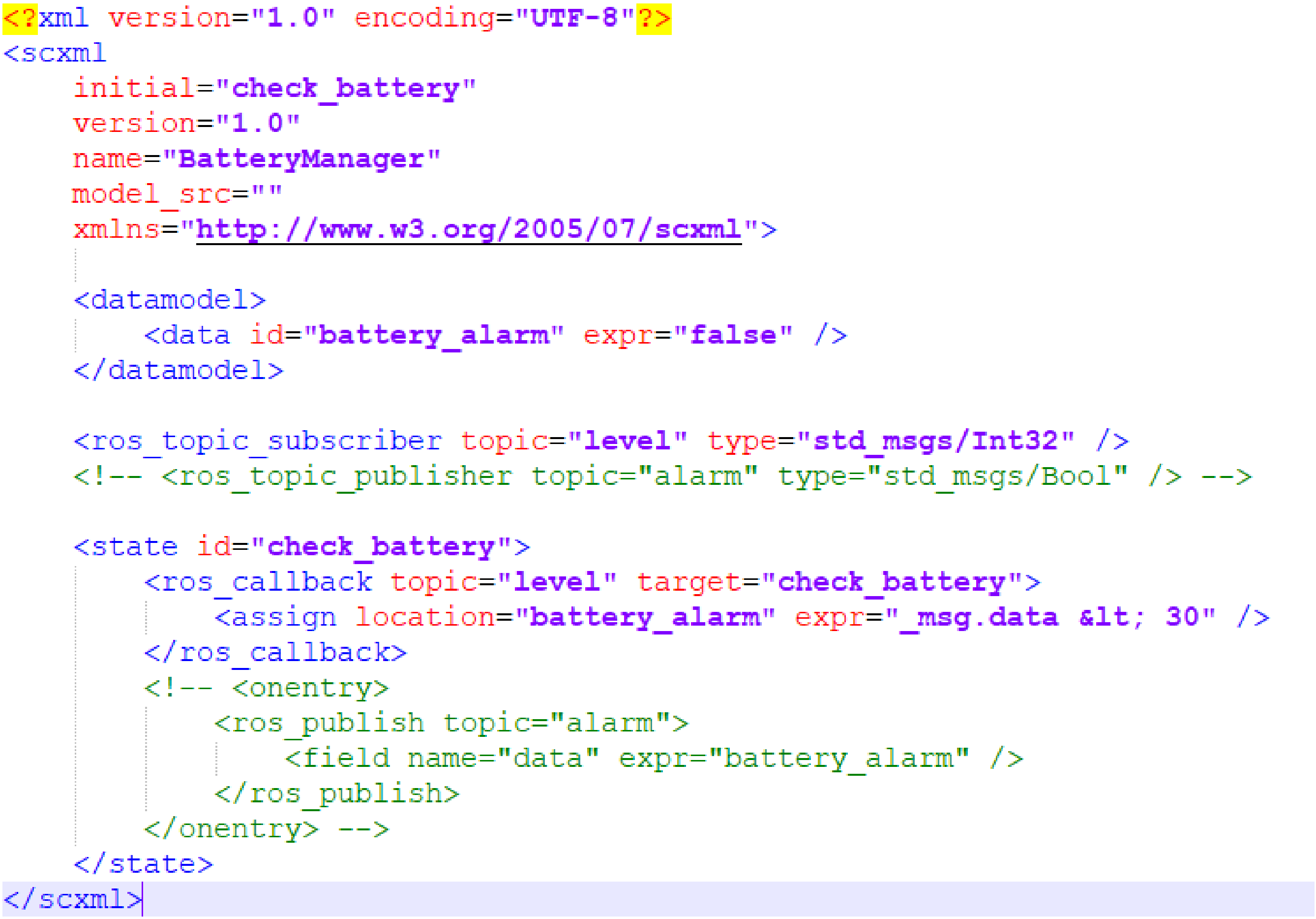
Tutorial: https://alexzhornyak.github.io/SCXML-tutorial/

SCXML
Example



```
<?xml version="1.0" encoding="UTF-8"?>
<scxml
    initial="use_battery"
    version="1.0"
    name="BatteryDrainer"
    model_src=""
    xmlns="http://www.w3.org/2005/07/scxml">

    <datamodel>
        <data id="battery_percent" expr="100" />
    </datamodel>

    <!-- <ros_topic_subscriber topic="charge" type="std_msgs/Empty" /> -->
    <ros_topic_publisher topic="level" type="std_msgs/Int32" />
    <ros_time_rate rate_hz="1" name="my_timer" />

    <state id="use_battery">
        <onentry>
            <ros_publish topic="level">
                <field name="data" expr="battery_percent" />
            </ros_publish>
        </onentry>
        <ros_rate_callback name="my_timer" target="use_battery" cond="battery_percent > 0">
            <assign location="battery_percent" expr="battery_percent - 1" />
        </ros_rate_callback>
        <!-- <ros_callback topic="charge" target="use_battery">
            <assign location="battery_percent" expr="100" />
        </ros_callback> -->
    </state>
</scxml>
```

SCXML
Example



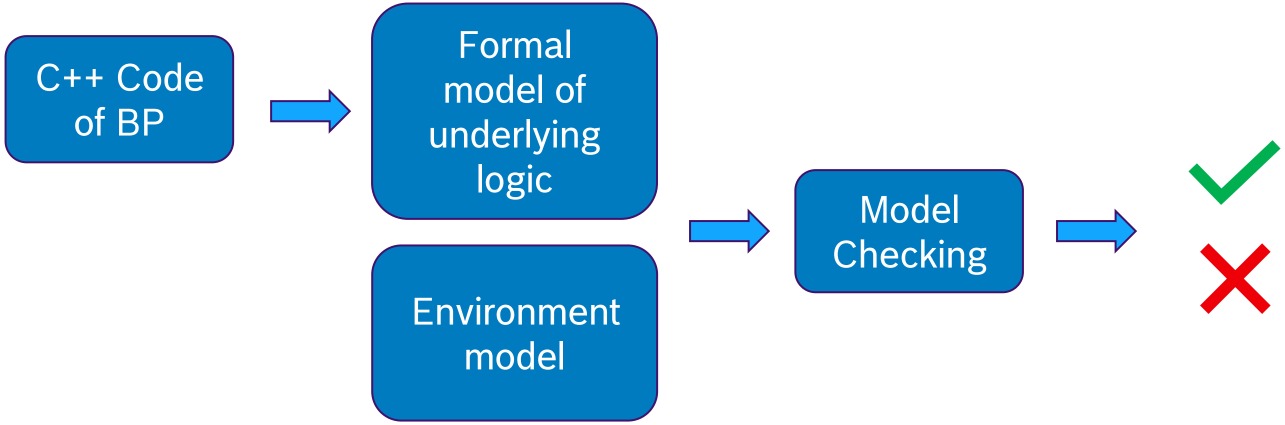
```
<?xml version="1.0" encoding="UTF-8"?>
<scxml
    initial="check_battery"
    version="1.0"
    name="BatteryManager"
    model_src=""
    xmlns="http://www.w3.org/2005/07/scxml">

    <datamodel>
        <data id="battery_alarm" expr="false" />
    </datamodel>

    <ros_topic_subscriber topic="level" type="std_msgs/Int32" />
    <!-- <ros_topic_publisher topic="alarm" type="std_msgs/Bool" /> -->

    <state id="check_battery">
        <ros_callback topic="level" target="check_battery">
            <assign location="battery_alarm" expr="_msg.data &lt; 30" />
        </ros_callback>
        <!-- <onentry>
            <ros_publish topic="alarm">
                <field name="data" expr="battery_alarm" />
            </ros_publish>
        </onentry> -->
    </state>
</scxml>
```

Real World Application Example:

Model Checking in Industry

Towards Safe Autonomus Driving:
Model Checking a Behavior Planner during Development

König L., Heinzemann C., Griggio A., Klauck M., Cimatti A., Henze F., Tonetta S., Küperkoch S., Fassbender D., & Hanselmann M. Towards safe Autonomous Driving: Model Checking a Behavior Planner during Development. In: Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2024.

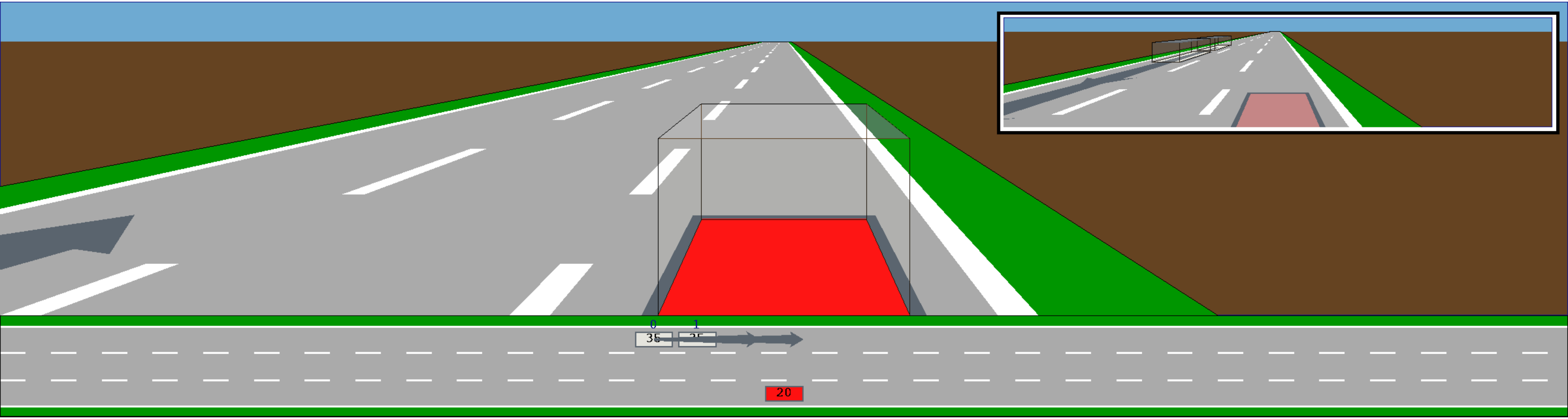


Artifacts Evaluated EAPLS Reusable V1.1

Artifacts Available EAPLS V1.1

Model Checking a Behavior Planner during Development

Counter Examples – Double Merge



Model Checking a Behavior Planner during Development
Results

- Deployed in series development
- Found **relevant issues** in intermediate versions of planner at development time
- Success factors:
  - **automatically extracted formal model** from production code
  - **seamless integration** into development environment

Lessons Learned

Findings from 2 Projects briging FM to Autonomous Systems Engineering

- Formal specifications require **support for engineers:**
  - modeling **languages expressive** enough for industrial settings
  - logics with **efficient tooling** incl. **design & debugging** support
- Support for **model extraction**, generation, writing
  to bridge gap to running industrial systems
- Model **validation**
- Tightly **integrated tooling** in development environments
- Low **inhibition barrier** needed

CONVINCE

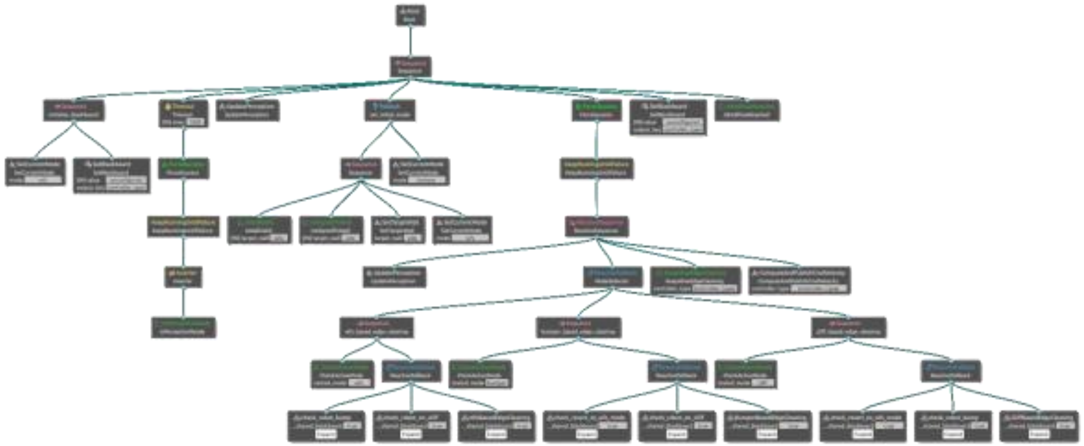
CONtext-aware Verifiable and adaptIve dyNamiC deEliberation

BOSCH

Use Cases

Dynamic Deliberation with Behavior Trees

Excerpt of edge cleaning BT

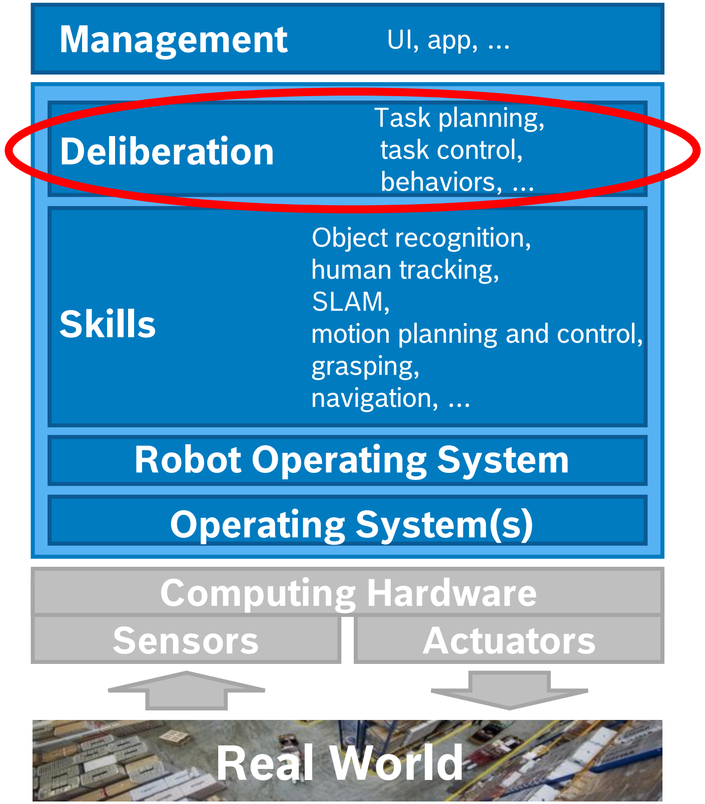


(blurred for confidentiality reasons)

Vision
Robust Autonomous Robots

1. Robustness in system architecture & environment interaction

2. Adaptive, **cognitive deliberation systems**
   - detecting & coping with unexpected situations
   - providing **contingency plans**



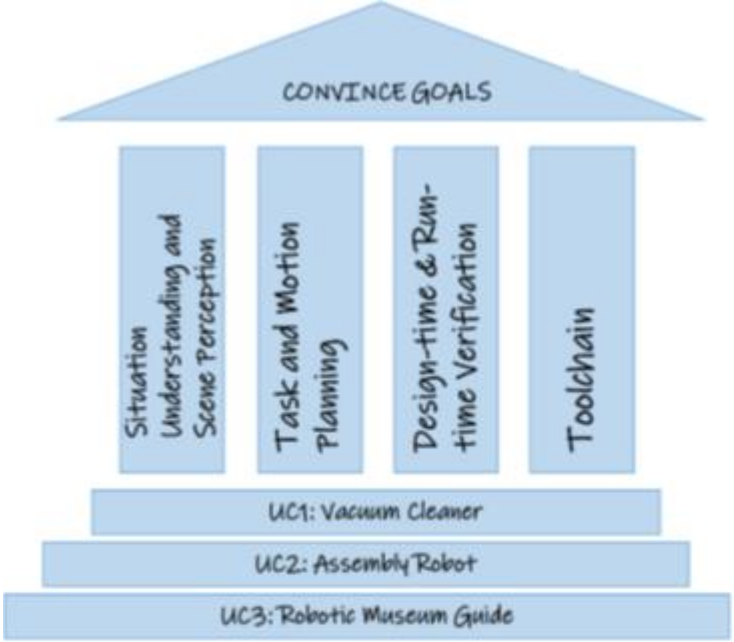
Vision
Robust Autonomous Robots

1. Robustness in system architecture & environment interaction

2. Adaptive, **cognitive deliberation systems**
   - detecting & coping with unexpected situations
   - providing **contingency plans**

3. **Formal methods** to ensure:
   - robust & correct execution of behaviors
   - at **design & run time**
   - **Statistical Model Checking** on entire system

4. **Open-source software toolchain** for behavior developers
   - Bring together **existing model checking & robotic** tools
   - Reduce software development and maintenance efforts

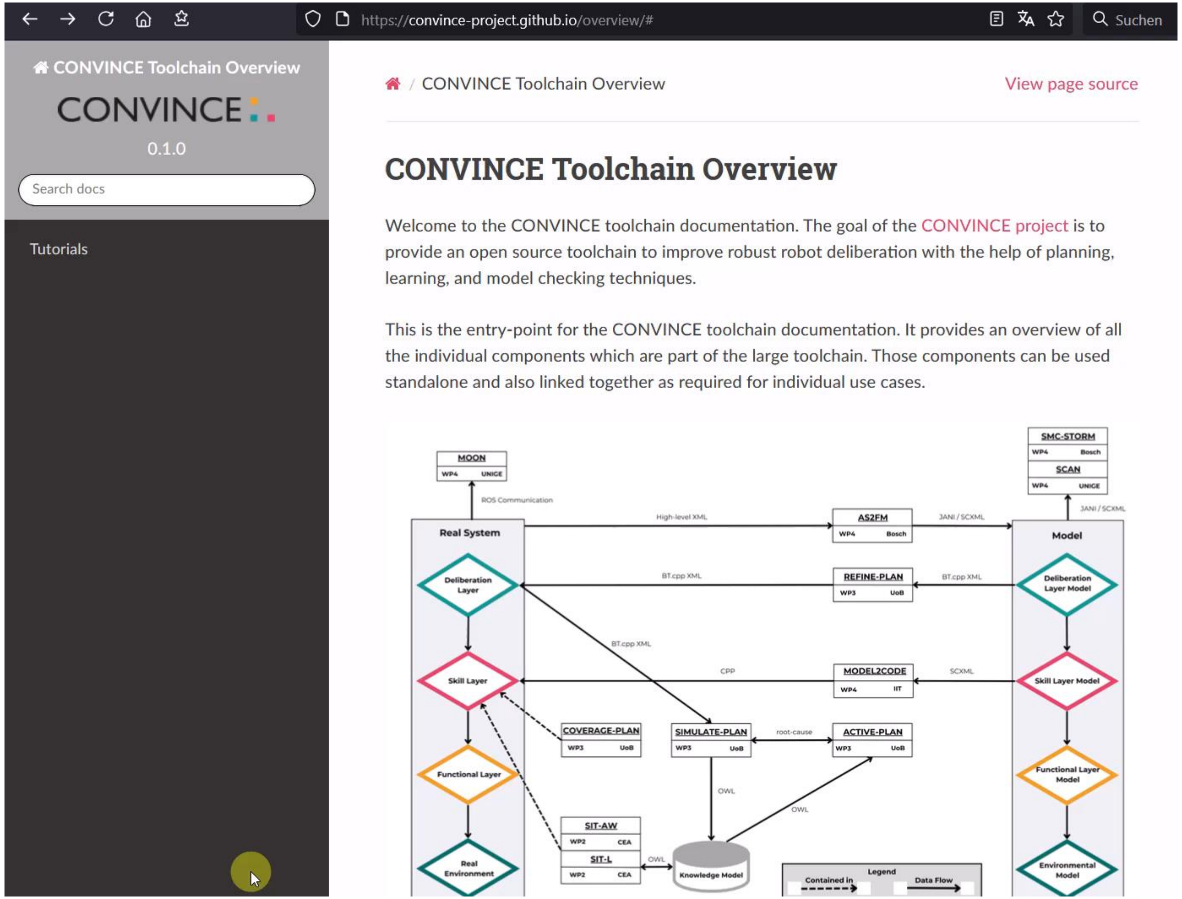

CONVINCE GOALS
Situation Understanding and Scene Perception
Task and Motion Planning
Design-time & Run-time Verification
Toolchain
UC1: Vacuum Cleaner
UC2: Assembly Robot
UC3: Robotic Museum Guide

Toolbox Components

Verification, Monitoring, Planning, Situation Understanding

Street, Warsame, Mansouri, Klauck, Henkel, Lampacrescia, Palmas, Lange Ghiorzi, Tacchella, Azrou, Lallement, Morelli, Chen, Wallis, Bernagozzi, Rosa, Randazzo, Faraci, Natale.
Towards a Verifiable Toolchain for Robotics.
Proceedings of the AAAI Symposium Series 2024.

Best Paper Award



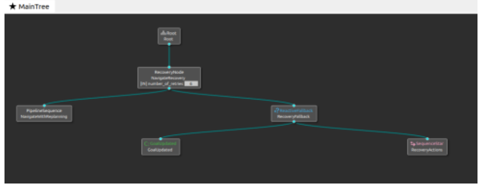
CONVINCE Toolchain Overview

Welcome to the CONVINCE toolchain documentation. The goal of the CONVINCE project is to provide an open source toolchain to improve robust robot deliberation with the help of planning, learning, and model checking techniques.

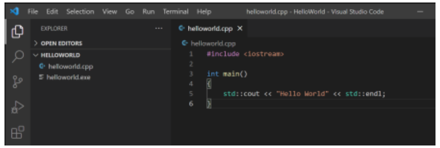
This is the entry-point for the CONVINCE toolchain documentation. It provides an overview of all the individual components which are part of the large toolchain. Those components can be used standalone and also linked together as required for individual use cases.

https://convince-project.github.io/overview/

Model Checking Tooling
Overview



BT-Model


File Edit Selection View Go Run Terminal Help
helloworld.cpp - HelloWorld - Visual Studio Code
EXPLORER
OPEN EDITORS
HELLOWORLD
helloworld.cpp
helloworld.exe
helloworld.cpp

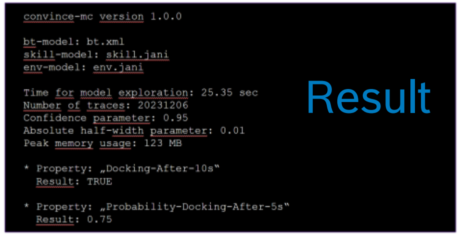
```
#include <iostream>

int main()
{
    std::cout << "Hello World" << std::endl;
}
```

Skill Model

Environment Model

Properties

Model Checker



```
convince-mc version 1.0.0

bt-model: bt.xml
skill-model: skill.jani
env-model: env.jani

Time for model exploration: 25.35 sec
Number of traces: 20231206
Confidence parameter: 0.95
Absolute half-width parameter: 0.01
Peak memory usage: 123 MB

* Property: „Docking-After-10s"
  Result: TRUE

* Property: „Probability-Docking-After-5s"
  Result: 0.75
```

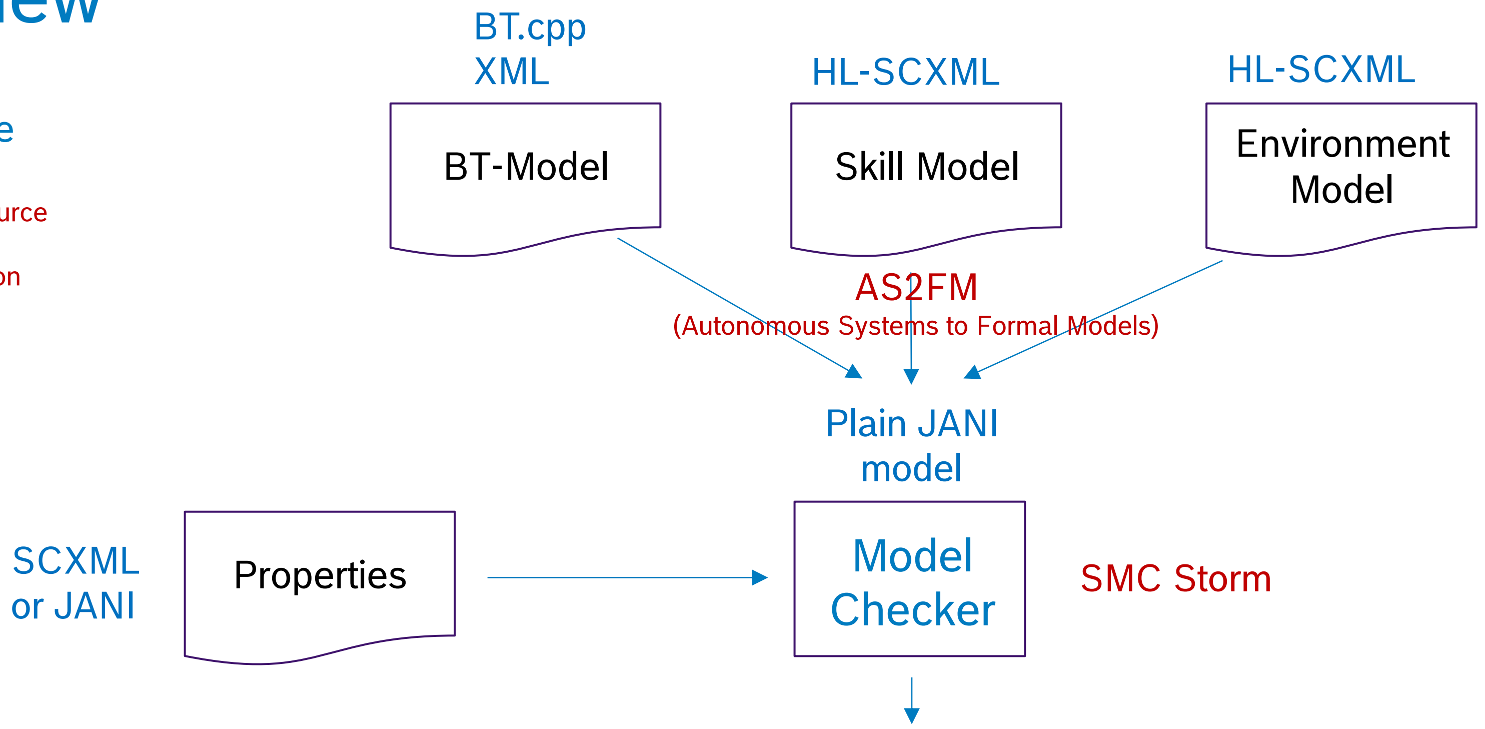
Result

Model Checking Tooling
Overview

Format Name

Tool Name

Available open source in the CONVINCE GitHub organization



BT.cpp
XML

BT-Model

HL-SCXML

Skill Model

HL-SCXML

Environment Model

AS2FM
(Autonomous Systems to Formal Models)

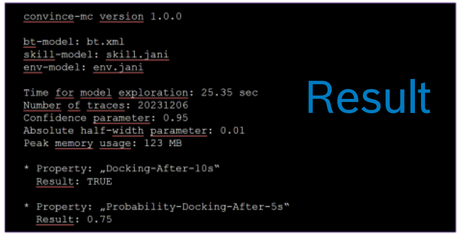
Plain JANI model

SCXML or JANI

Properties

Model Checker

SMC Storm

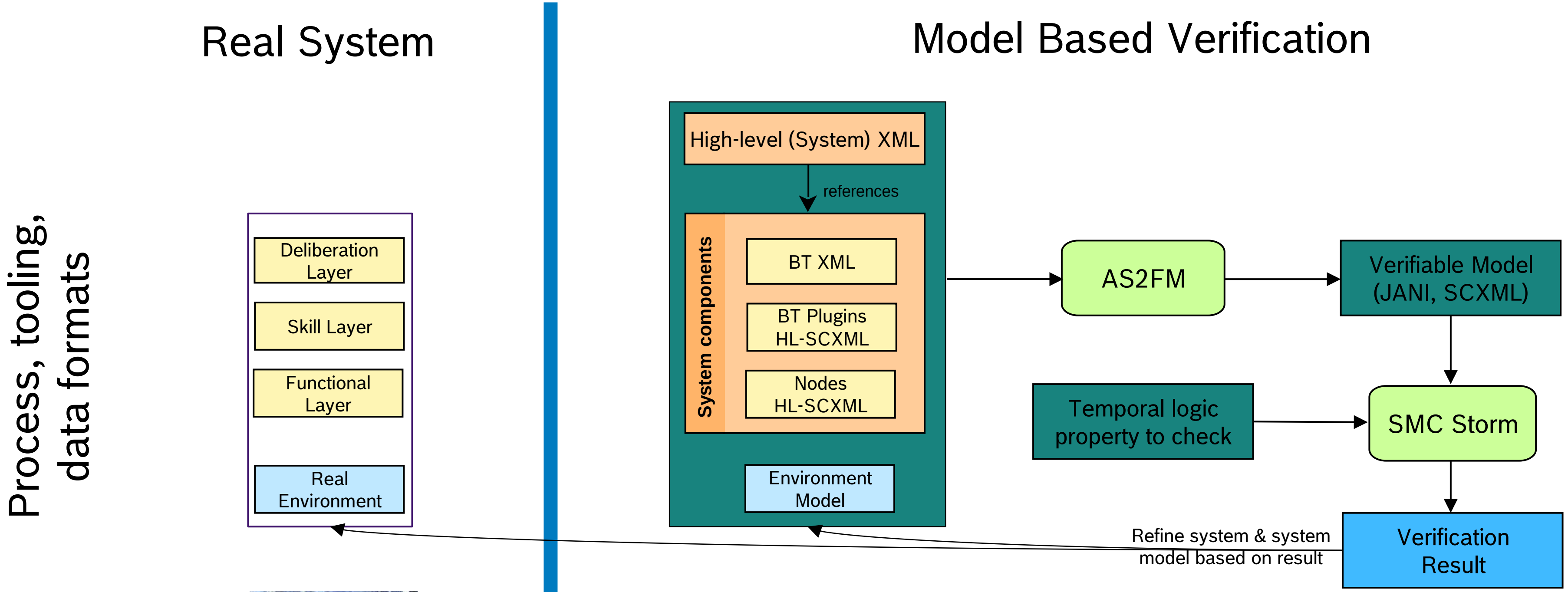


```
convince-mc version 1.0.0

bt-model: bt.xml
skill-model: skill.jani
env-model: env.jani

Time for model exploration: 25.35 sec
Number of traces: 20231206
Confidence parameter: 0.95
Absolute half-width parameter: 0.01
Peak memory usage: 123 MB

* Property: „Docking-After-10s"
  Result: TRUE

* Property: „Probability-Docking-After-5s"
  Result: 0.75
```

Result

Model Checking Tooling

Overview



Example

- BT in BT.cpp XML
- BT Plugins
- ROS 2 Nodes
- Skills
- Environment XML

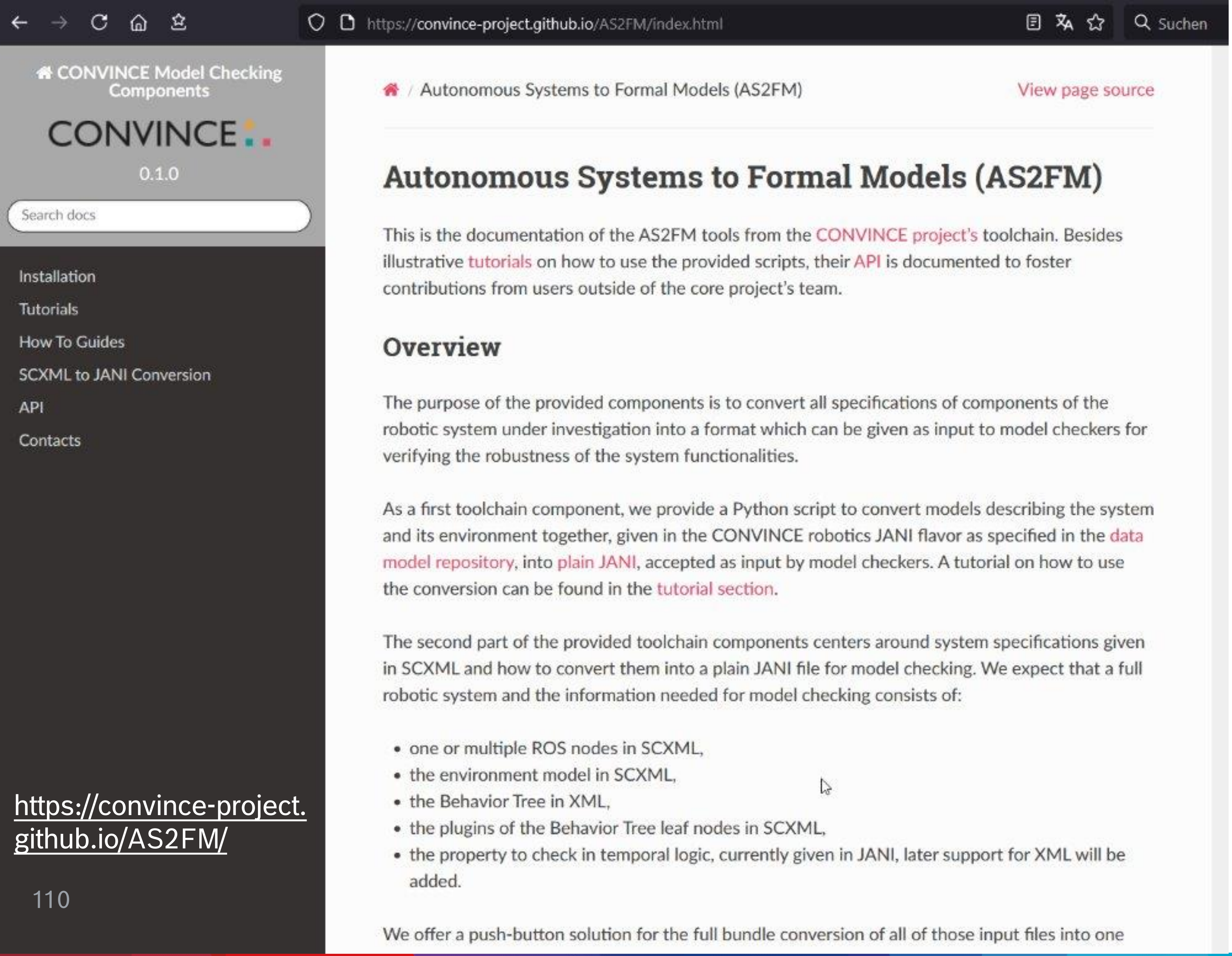
- Destination always reached within time limit?
- Battery never depleted
- Docking always successfull?

- No + counter example execution trace
- Yes
- No, only in 98.5% + counter examples

AS2FM

Autonomous Systems to Formal Models

BOSCH


https://convince-project.github.io/AS2FM/index.html
Suchen
CONVINCE Model Checking Components
CONVINCE
0.1.0
Search docs
Installation
Tutorials
How To Guides
SCXML to JANI Conversion
API
Contacts
Autonomous Systems to Formal Models (AS2FM)
View page source

Autonomous Systems to Formal Models (AS2FM)

This is the documentation of the AS2FM tools from the CONVINCE project's toolchain. Besides illustrative tutorials on how to use the provided scripts, their API is documented to foster contributions from users outside of the core project's team.

Overview

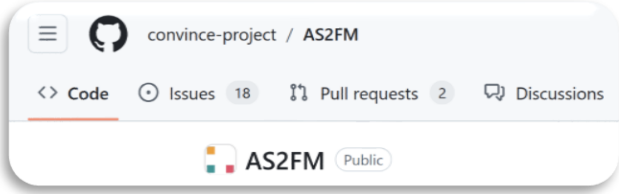
The purpose of the provided components is to convert all specifications of components of the robotic system under investigation into a format which can be given as input to model checkers for verifying the robustness of the system functionalities.

As a first toolchain component, we provide a Python script to convert models describing the system and its environment together, given in the CONVINCE robotics JANI flavor as specified in the data model repository, into plain JANI, accepted as input by model checkers. A tutorial on how to use the conversion can be found in the tutorial section.

The second part of the provided toolchain components centers around system specifications given in SCXML and how to convert them into a plain JANI file for model checking. We expect that a full robotic system and the information needed for model checking consists of:

- one or multiple ROS nodes in SCXML,
- the environment model in SCXML,
- the Behavior Tree in XML,
- the plugins of the Behavior Tree leaf nodes in SCXML,
- the property to check in temporal logic, currently given in JANI, later support for XML will be added.

We offer a push-button solution for the full bundle conversion of all of those input files into one



https://convince-project.github.io/AS2FM/

AS2FM
Docking Example: Intro

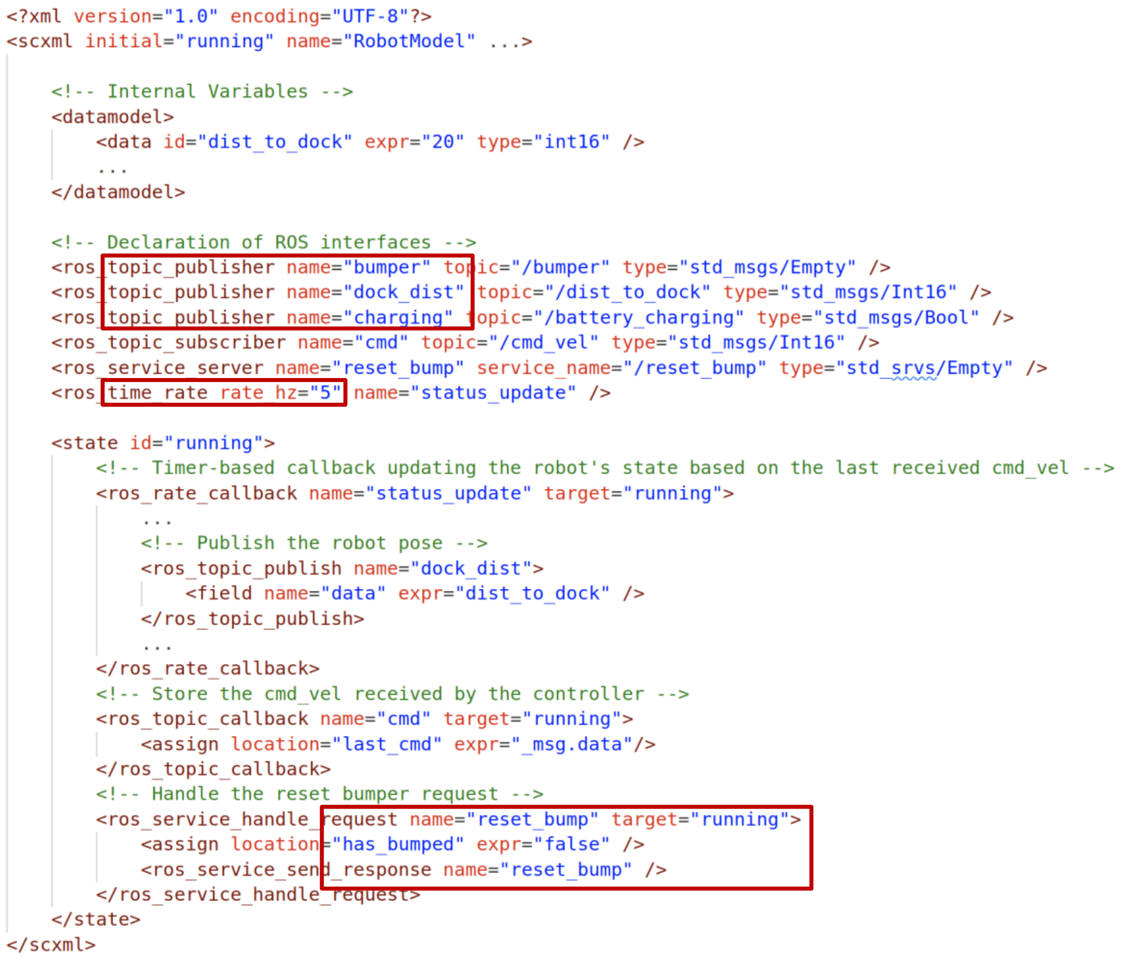
Goal: Verify that docking procedure works

Ingredients:

- Complete model's components:
  - Behavior Tree definition (XML)
  - Behavior Tree plugins’ definition (HL-SCXML)
  - ROS 2 Communication (HL-SCXML)
  - Model of the robot’s environment (HL-SCXML)
- Property: Check if tree_success message is published (i.e., robot starts charging)

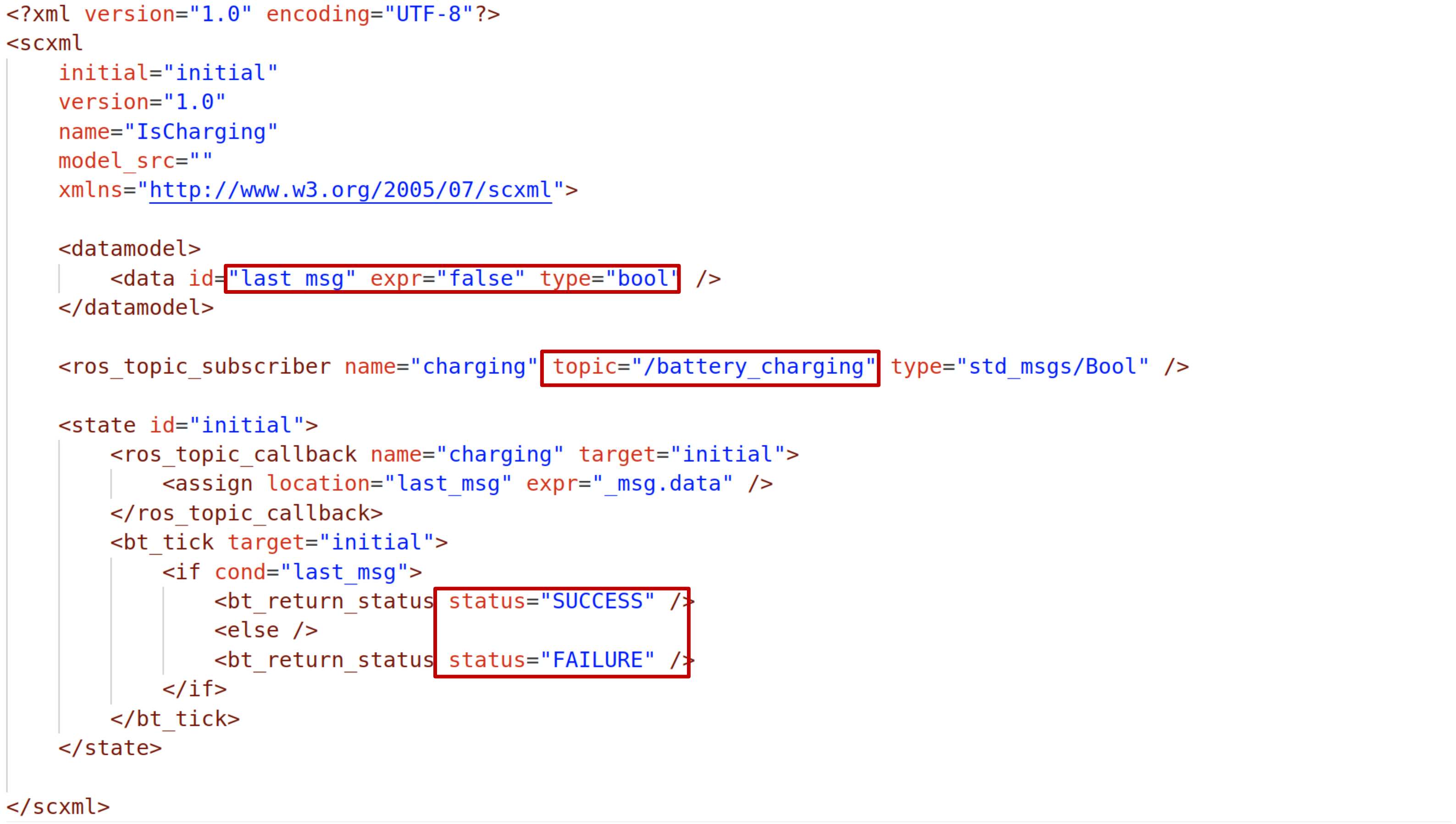
AS2FM

Docking Example: Robot High-Level SCXML Model



```xml
<?xml version="1.0" encoding="UTF-8"?>
<scxml initial="running" name="RobotModel" ...>

    <!-- Internal Variables -->
    <datamodel>
        <data id="dist_to_dock" expr="20" type="int16" />
        ...
    </datamodel>

    <!-- Declaration of ROS interfaces -->
    <ros_topic_publisher name="bumper" topic="/bumper" type="std_msgs/Empty" />
    <ros_topic_publisher name="dock_dist" topic="/dist_to_dock" type="std_msgs/Int16" />
    <ros_topic_publisher name="charging" topic="/battery_charging" type="std_msgs/Bool" />
    <ros_topic_subscriber name="cmd" topic="/cmd_vel" type="std_msgs/Int16" />
    <ros_service_server name="reset_bump" service_name="/reset_bump" type="std_srvs/Empty" />
    <ros_time_rate rate_hz="5" name="status_update" />

    <state id="running">
        <!-- Timer-based callback updating the robot's state based on the last received cmd_vel -->
        <ros_rate_callback name="status_update" target="running">
            ...
            <!-- Publish the robot pose -->
            <ros_topic_publish name="dock_dist">
                <field name="data" expr="dist_to_dock" />
            </ros_topic_publish>
            ...
        </ros_rate_callback>
        <!-- Store the cmd_vel received by the controller -->
        <ros_topic_callback name="cmd" target="running">
            <assign location="last_cmd" expr="_msg.data"/>
        </ros_topic_callback>
        <!-- Handle the reset bumper request -->
        <ros_service_handle_request name="reset_bump" target="running">
            <assign location="has_bumped" expr="false" />
            <ros_service_send_response name="reset_bump" />
        </ros_service_handle_request>
    </state>
</scxml>
```

AS2FM
Docking Example: Behavior Tree Plugins



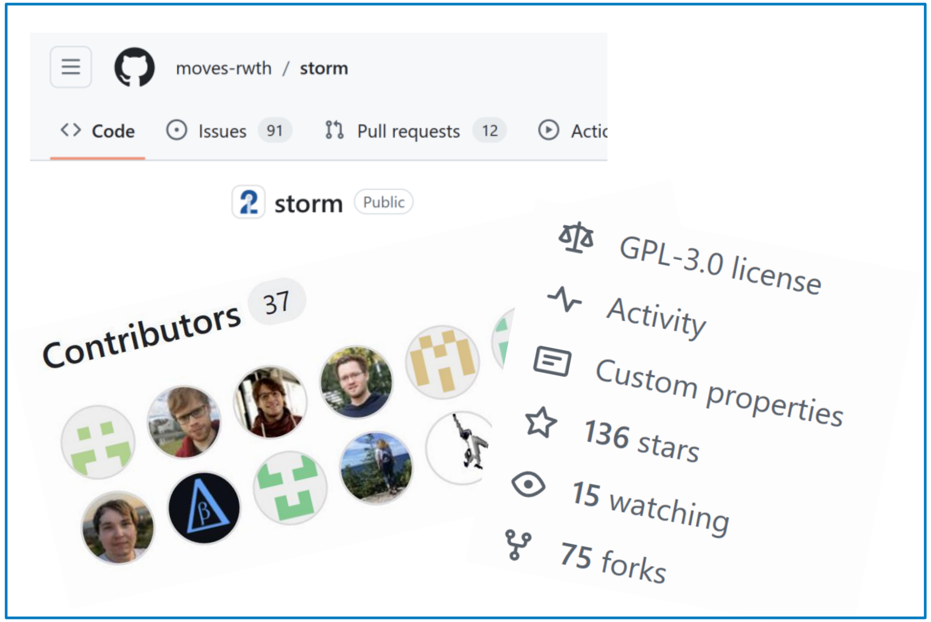
```
<?xml version="1.0" encoding="UTF-8"?>
<scxml
    initial="initial"
    version="1.0"
    name="IsCharging"
    model_src=""
    xmlns="http://www.w3.org/2005/07/scxml">

    <datamodel>
        <data id="last_msg" expr="false" type="bool" />
    </datamodel>

    <ros_topic_subscriber name="charging" topic="/battery_charging" type="std_msgs/Bool" />

    <state id="initial">
        <ros_topic_callback name="charging" target="initial">
            <assign location="last_msg" expr="_msg.data" />
        </ros_topic_callback>
        <bt_tick target="initial">
            <if cond="last_msg">
                <bt_return_status status="SUCCESS" />
                <else />
                <bt_return_status status="FAILURE" />
            </if>
        </bt_tick>
    </state>

</scxml>
```

SMC Storm

SMC Storm
Overview

Why settling on Storm ?:

- Mature open-source project
- Built-in support for JANI models
- Implemented in C++
- Very good performance in QComp



moves-rwth / storm

<> Code · Issues 91 · Pull requests 12 · Actic

storm Public

Contributors 37

GPL-3.0 license

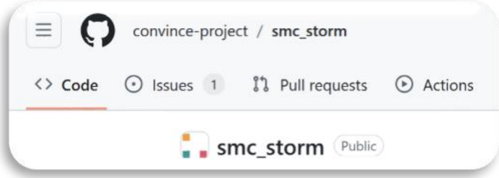
Activity

Custom properties

136 stars

15 watching

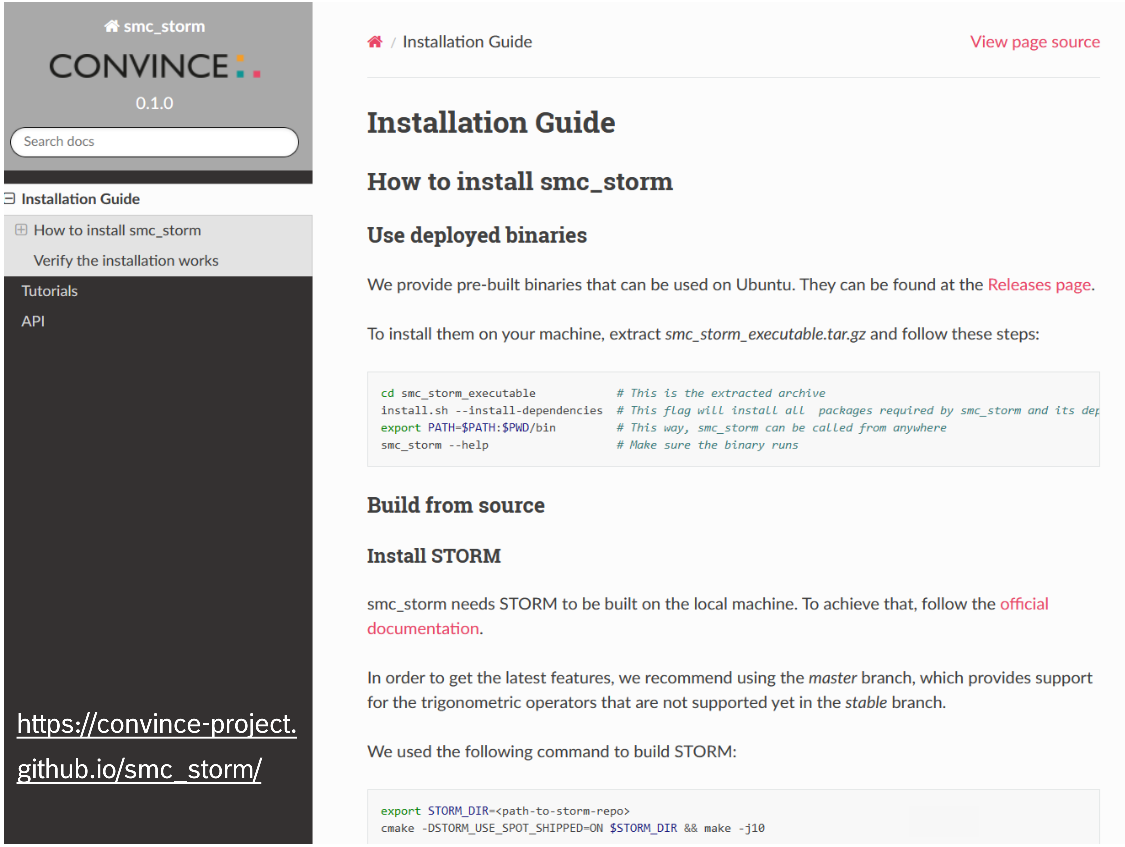
75 forks



convince-project / smc_storm

<> Code · Issues 1 · Pull requests · Actions

smc_storm Public



smc_storm

CONVINCE

0.1.0

Search docs

Installation Guide

How to install smc_storm

Verify the installation works

Tutorials

API

https://convince-project.github.io/smc_storm/

Installation Guide · View page source

Installation Guide

How to install smc_storm

Use deployed binaries

We provide pre-built binaries that can be used on Ubuntu. They can be found at the Releases page.

To install them on your machine, extract *smc_storm_executable.tar.gz* and follow these steps:

```
cd smc_storm_executable            # This is the extracted archive
install.sh --install-dependencies  # This flag will install all  packages required by smc_storm and its dep
export PATH=$PATH:$PWD/bin         # This way, smc_storm can be called from anywhere
smc_storm --help                   # Make sure the binary runs
```

Build from source

Install STORM

smc_storm needs STORM to be built on the local machine. To achieve that, follow the official documentation.

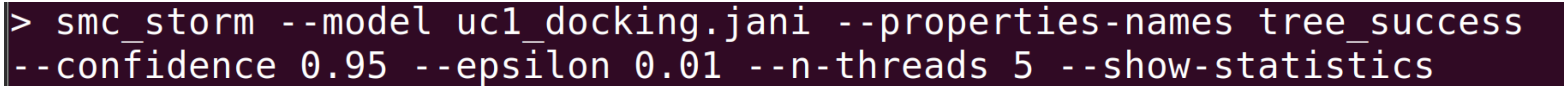
In order to get the latest features, we recommend using the *master* branch, which provides support for the trigonometric operators that are not supported yet in the *stable* branch.

We used the following command to build STORM:

```
export STORM_DIR=<path-to-storm-repo>
cmake -DSTORM_USE_SPOT_SHIPPED=ON $STORM_DIR && make -j10
```

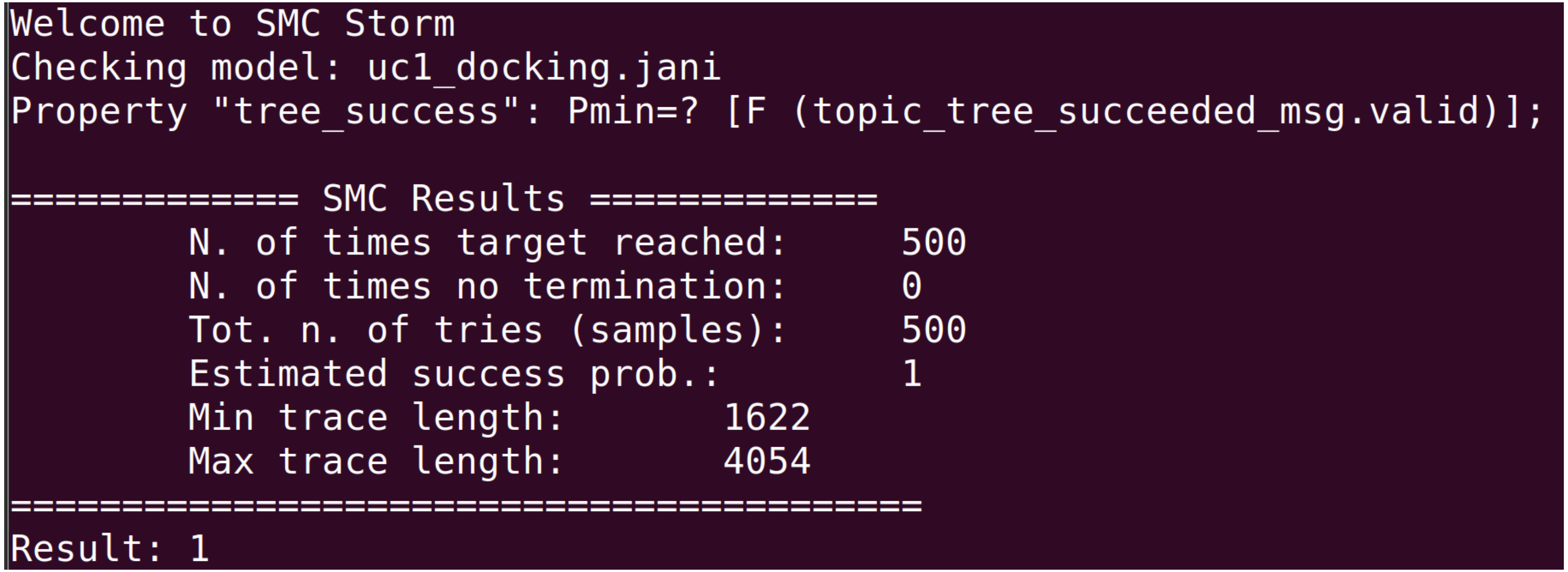
SMC Storm
Typical usage

Input:



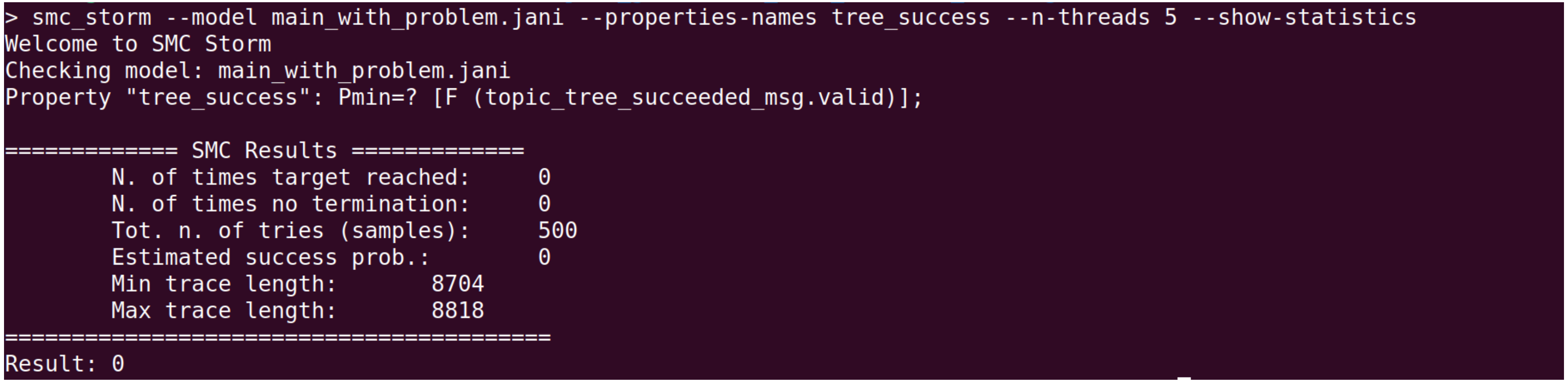
```
> smc_storm --model uc1_docking.jani --properties-names tree_success
--confidence 0.95 --epsilon 0.01 --n-threads 5 --show-statistics
```

Output:

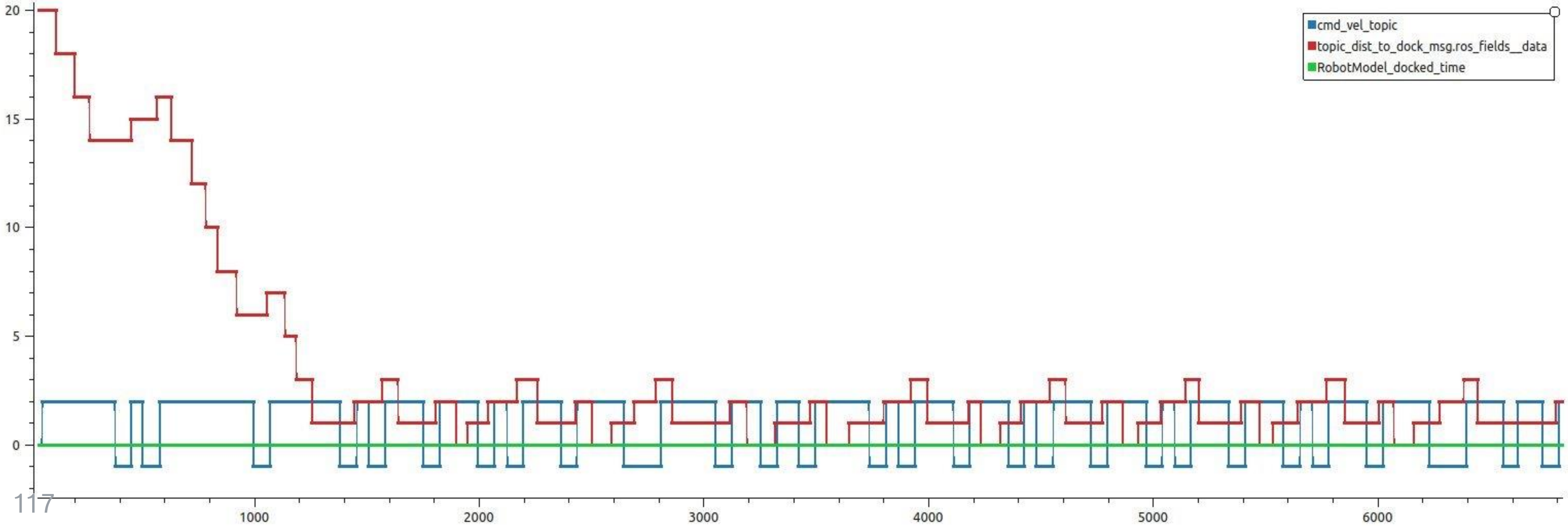


```
Welcome to SMC Storm
Checking model: uc1_docking.jani
Property "tree_success": Pmin=? [F (topic_tree_succeeded_msg.valid)];

============= SMC Results =============
        N. of times target reached:     500
        N. of times no termination:     0
        Tot. n. of tries (samples):     500
        Estimated success prob.:        1
        Min trace length:       1622
        Max trace length:       4054
=========================================
Result: 1
```

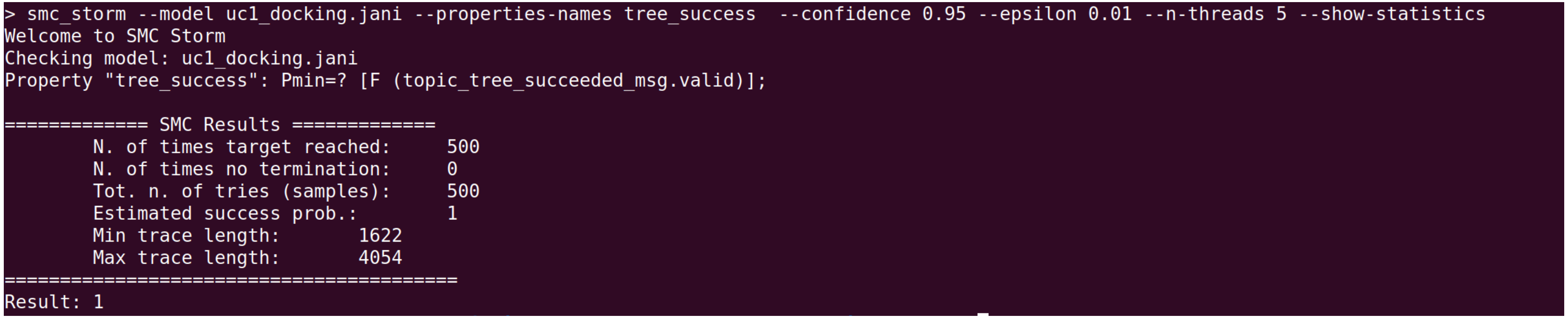
SMC Storm on Docking Example



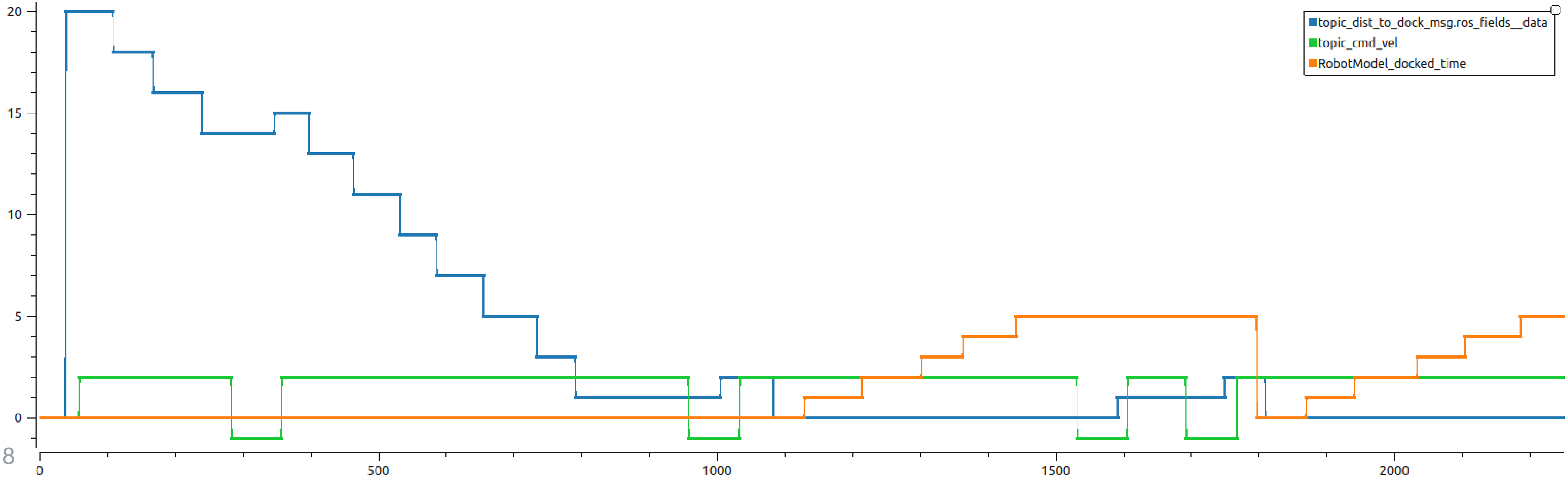
```
> smc_storm --model main_with_problem.jani --properties-names tree_success --n-threads 5 --show-statistics
Welcome to SMC Storm
Checking model: main_with_problem.jani
Property "tree_success": Pmin=? [F (topic_tree_succeeded_msg.valid)];

============= SMC Results =============
        N. of times target reached:     0
        N. of times no termination:     0
        Tot. n. of tries (samples):     500
        Estimated success prob.:        0
        Min trace length:       8704
        Max trace length:       8818
=========================================
Result: 0
```



SMC Storm on Docking Example

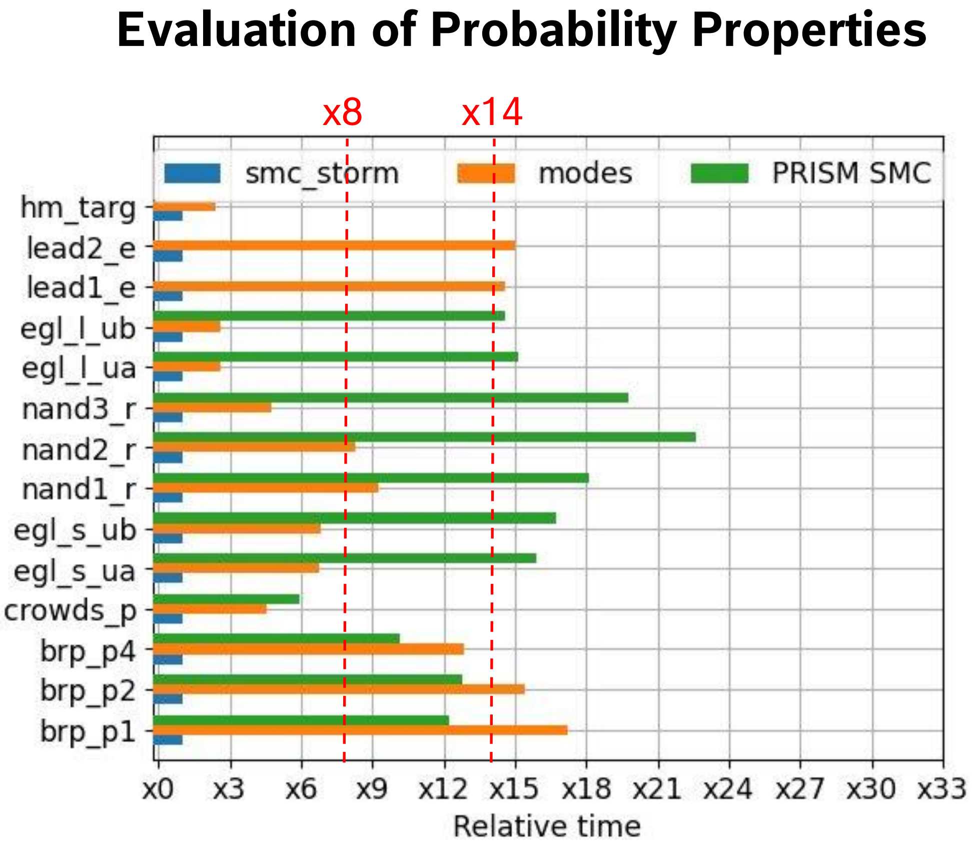


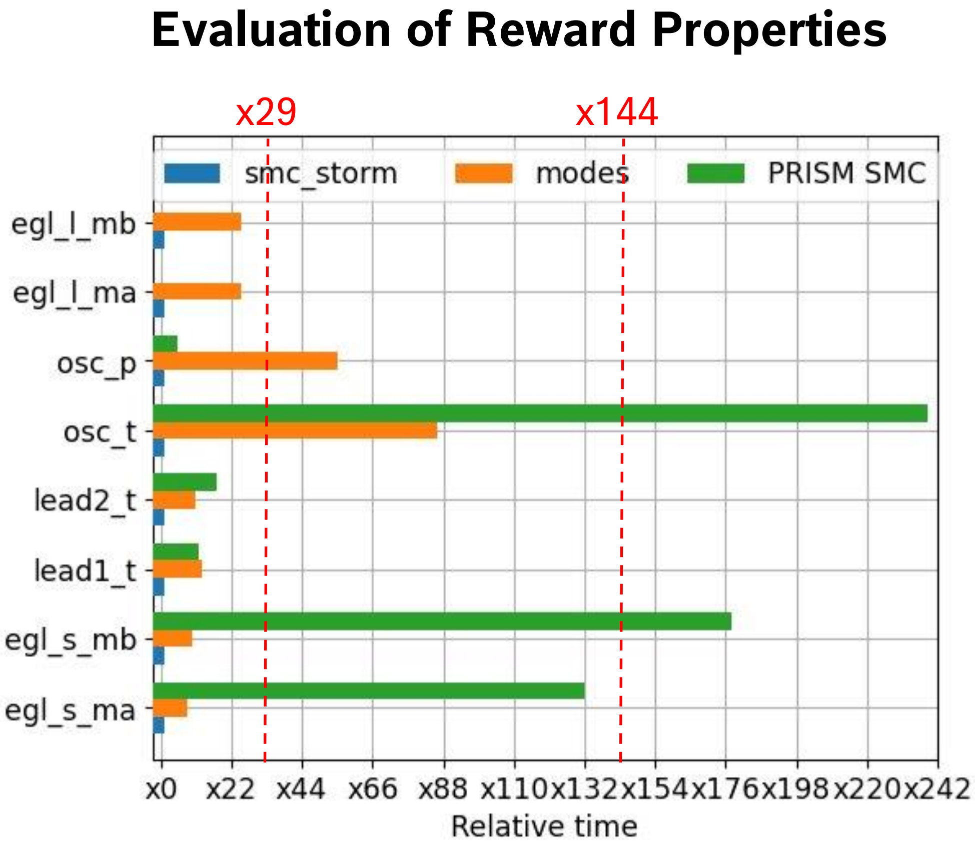
```
> smc_storm --model uc1_docking.jani --properties-names tree_success  --confidence 0.95 --epsilon 0.01 --n-threads 5 --show-statistics
Welcome to SMC Storm
Checking model: uc1_docking.jani
Property "tree_success": Pmin=? [F (topic_tree_succeeded_msg.valid)];

============= SMC Results =============
        N. of times target reached:     500
        N. of times no termination:     0
        Tot. n. of tries (samples):     500
        Estimated success prob.:        1
        Min trace length:       1622
        Max trace length:       4054
=========================================
Result: 1
```



SMC Storm
Performance Evaluation

Lampacrescia M., Klauck M., & Palmas M.
Towards Verifying Robotic Systems using Statistical Model Checking in STORM.
In: Bridging the Gap Between AI and Reality.
AISoLA 2024.





Model Checker Tool Demo

The Modest Toolset,

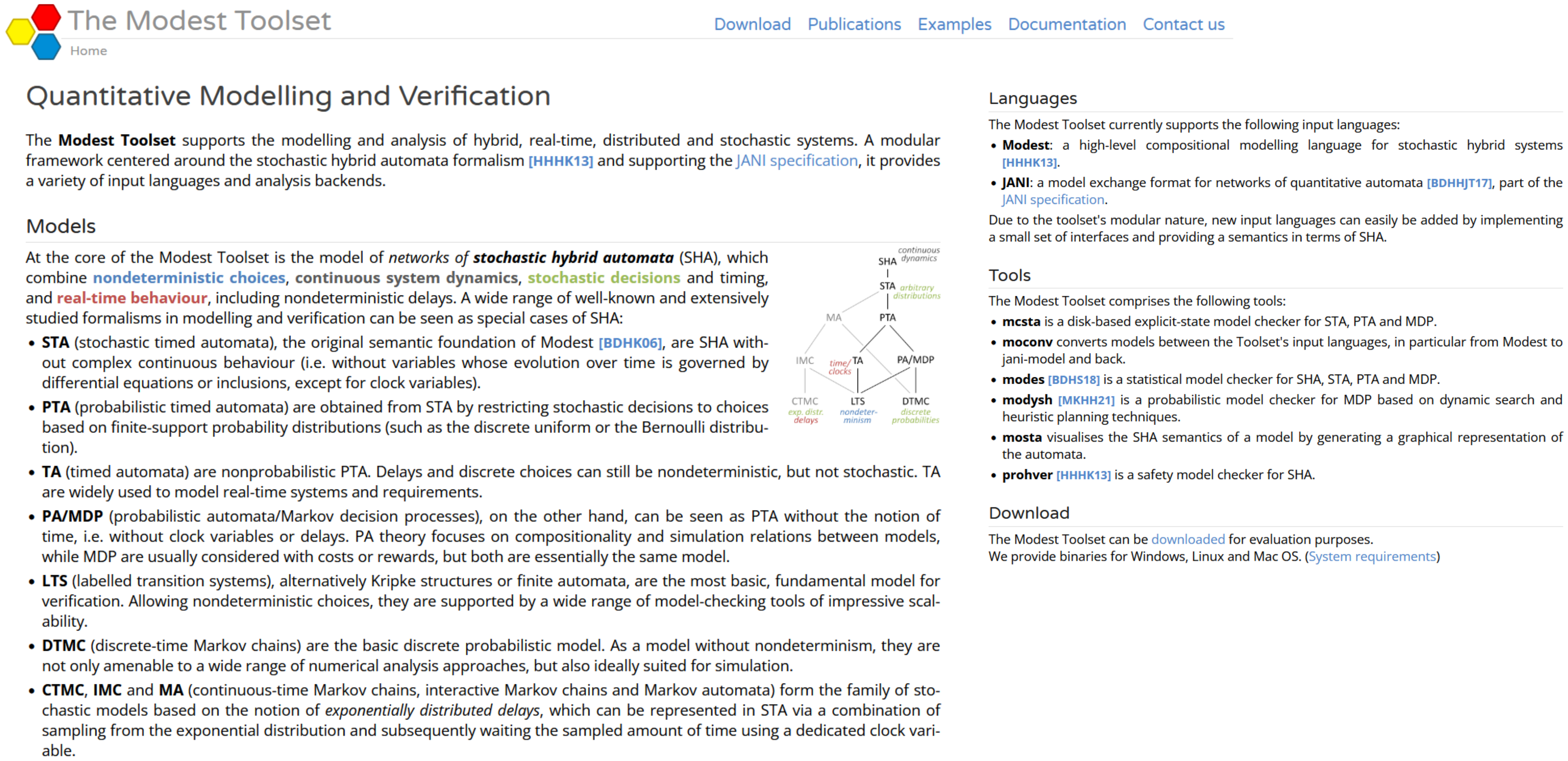
Storm,

SMC Storm

The Modest Toolset

https://www.modestchecker.net/

The Most Versatile Probab. Model Checker wrt. Analysis Backends & Supported Model Types



The Modest Toolset

Download Publications Examples Documentation Contact us

Home

Quantitative Modelling and Verification

The **Modest Toolset** supports the modelling and analysis of hybrid, real-time, distributed and stochastic systems. A modular framework centered around the stochastic hybrid automata formalism **[HHHK13]** and supporting the JANI specification, it provides a variety of input languages and analysis backends.

Models

At the core of the Modest Toolset is the model of *networks of* ***stochastic hybrid automata*** (SHA), which combine **nondeterministic choices**, **continuous system dynamics**, **stochastic decisions** and timing, and **real-time behaviour**, including nondeterministic delays. A wide range of well-known and extensively studied formalisms in modelling and verification can be seen as special cases of SHA:

- **STA** (stochastic timed automata), the original semantic foundation of Modest **[BDHK06]**, are SHA without complex continuous behaviour (i.e. without variables whose evolution over time is governed by differential equations or inclusions, except for clock variables).
- **PTA** (probabilistic timed automata) are obtained from STA by restricting stochastic decisions to choices based on finite-support probability distributions (such as the discrete uniform or the Bernoulli distribution).
- **TA** (timed automata) are nonprobabilistic PTA. Delays and discrete choices can still be nondeterministic, but not stochastic. TA are widely used to model real-time systems and requirements.
- **PA/MDP** (probabilistic automata/Markov decision processes), on the other hand, can be seen as PTA without the notion of time, i.e. without clock variables or delays. PA theory focuses on compositionality and simulation relations between models, while MDP are usually considered with costs or rewards, but both are essentially the same model.
- **LTS** (labelled transition systems), alternatively Kripke structures or finite automata, are the most basic, fundamental model for verification. Allowing nondeterministic choices, they are supported by a wide range of model-checking tools of impressive scalability.
- **DTMC** (discrete-time Markov chains) are the basic discrete probabilistic model. As a model without nondeterminism, they are not only amenable to a wide range of numerical analysis approaches, but also ideally suited for simulation.
- **CTMC**, **IMC** and **MA** (continuous-time Markov chains, interactive Markov chains and Markov automata) form the family of stochastic models based on the notion of *exponentially distributed delays*, which can be represented in STA via a combination of sampling from the exponential distribution and subsequently waiting the sampled amount of time using a dedicated clock variable.

Languages

The Modest Toolset currently supports the following input languages:

- **Modest**: a high-level compositional modelling language for stochastic hybrid systems **[HHHK13]**.
- **JANI**: a model exchange format for networks of quantitative automata **[BDHHJT17]**, part of the JANI specification.

Due to the toolset's modular nature, new input languages can easily be added by implementing a small set of interfaces and providing a semantics in terms of SHA.

Tools

The Modest Toolset comprises the following tools:

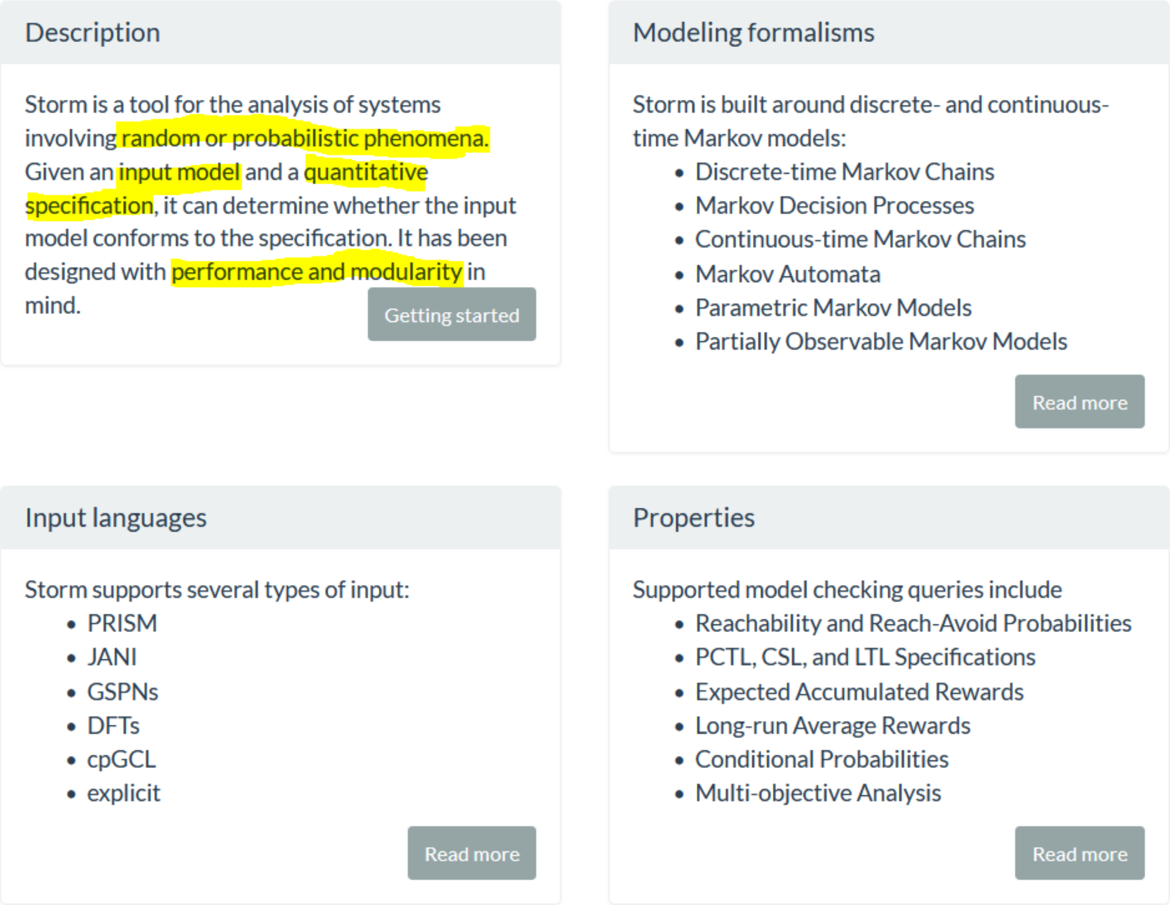
- **mcsta** is a disk-based explicit-state model checker for STA, PTA and MDP.
- **moconv** converts models between the Toolset's input languages, in particular from Modest to jani-model and back.
- **modes** **[BDHS18]** is a statistical model checker for SHA, STA, PTA and MDP.
- **modysh** **[MKHH21]** is a probabilistic model checker for MDP based on dynamic search and heuristic planning techniques.
- **mosta** visualises the SHA semantics of a model by generating a graphical representation of the automata.
- **prohver** **[HHHK13]** is a safety model checker for SHA.

Download

The Modest Toolset can be downloaded for evaluation purposes.
We provide binaries for Windows, Linux and Mac OS. (System requirements)

Storm https://www.stormchecker.org

The Leading State-of-the-Art Probabilistic Model Checker



Description

Storm is a tool for the analysis of systems involving random or probabilistic phenomena. Given an input model and a quantitative specification, it can determine whether the input model conforms to the specification. It has been designed with performance and modularity in mind.

Modeling formalisms

Storm is built around discrete- and continuous-time Markov models:

- Discrete-time Markov Chains
- Markov Decision Processes
- Continuous-time Markov Chains
- Markov Automata
- Parametric Markov Models
- Partially Observable Markov Models

Input languages

Storm supports several types of input:

- PRISM
- JANI
- GSPNs
- DFTs
- cpGCL
- explicit

Properties

Supported model checking queries include

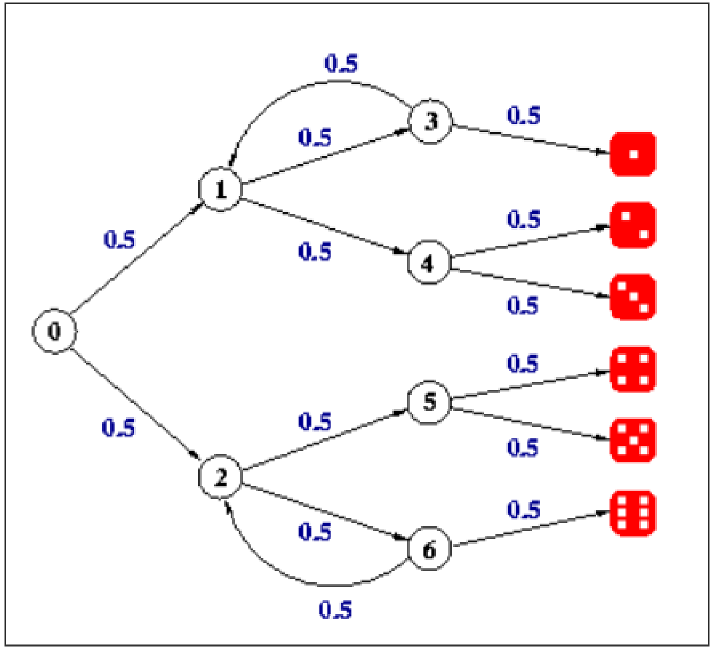
- Reachability and Reach-Avoid Probabilities
- PCTL, CSL, and LTL Specifications
- Expected Accumulated Rewards
- Long-run Average Rewards
- Conditional Probabilities
- Multi-objective Analysis

Engines:

- Sparse (explicit state space representation):
  - Main engine
  - Builds representation on explicit data structures (bit vectors, sparse matrices)
  - Fast numerical computations
- Decision Diagrams (symbolic):
  - BDDs (state sets) + MTBDDs (matrices, vectors)
  - Fast + memory efficient model building
- Hybrid
  - DDs for qualitative representations
  - Explicit for numerical computations
- Exploration:
  - Use ML to explore only parts contributing most to result
- Abstraction-Refinement:
  - Infinite or very large state spaces

Model Checker Tool Demo

Simulating a Dice with a Fair Coin



Hands-on:
Modeling & Model Checking of Robot Behavior

BOSCH

AS2FM & SMC Storm Tutorial

Verifying the Behavior of a Robot in a Fetch & Carry Task

- https://convince-project.github.io/AS2FM/tutorials.html
- Use AS2FM to convert an autonomous robotic system into a formal MDP model compatible with existing model checker tools (SCXML to JANI translation of the model)
- Use SMC Storm to model check a linear temporal logic (LTL) property expressing that the robot is reliably fetching an item and carrying it to the expected place
- Task:
  - Robot should drive to the pantry where food is stored,
  - Pick up snacks,
  - Drive to the table and place the snacks there.
- Extend with probabilistic behavior
- Adapt BT to handle probabilistic failures

AS2FM & SMC Storm Tutorial

Feedback Questionnaire

https://forms.gle/wtcQUPFdijFPhqcH7

Summer School Robot Software Engineering

Questions?

Thank you very much for participating in this great summer school!

Contact: Dr. Michaela Klauck,
michaela.klauck@de.bosch.com

BOSCH