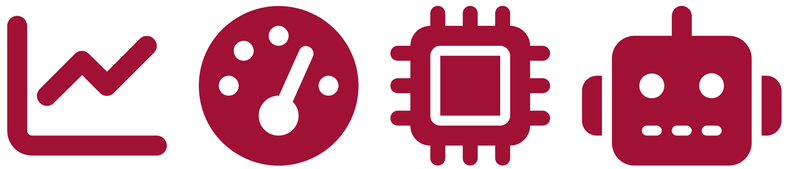
Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Seminar Trends in Model Checking

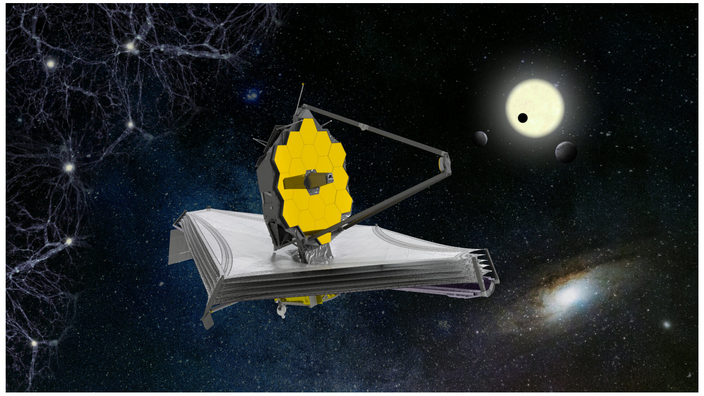
SS24 — Introduction



Tim Quatmann

https://moves.rwth-aachen.de/teaching/ss-24/seminar-trends-in-model-checking/

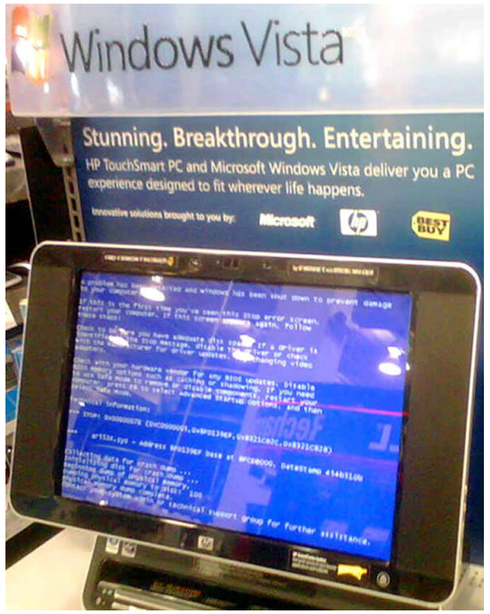
Software Systems are everywhere!

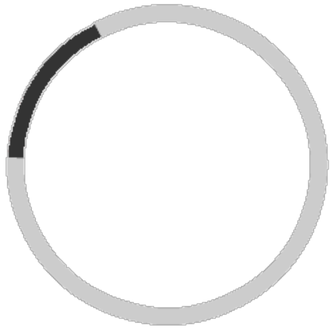


Software Systems are everywhere!

Software Failures are everywhere!

Software Failures are everywhere!



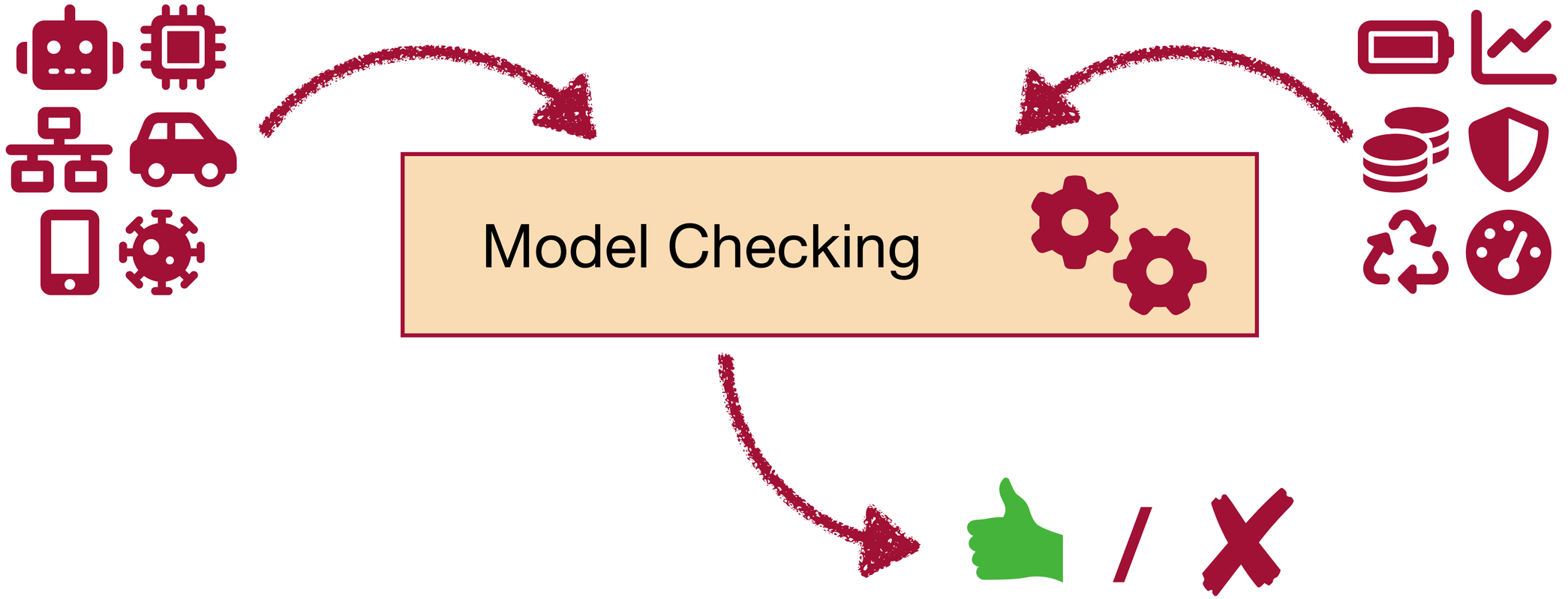


Loading..

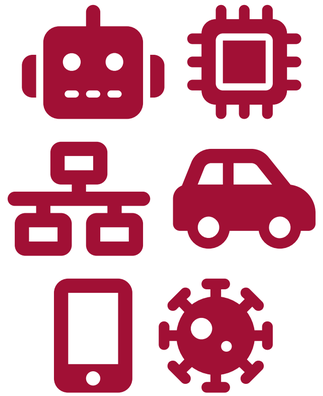
How to prevent such failures?

System Model

System Property

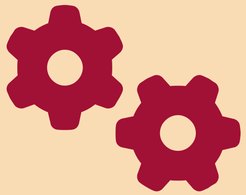


System Model



System Property

Model Checking

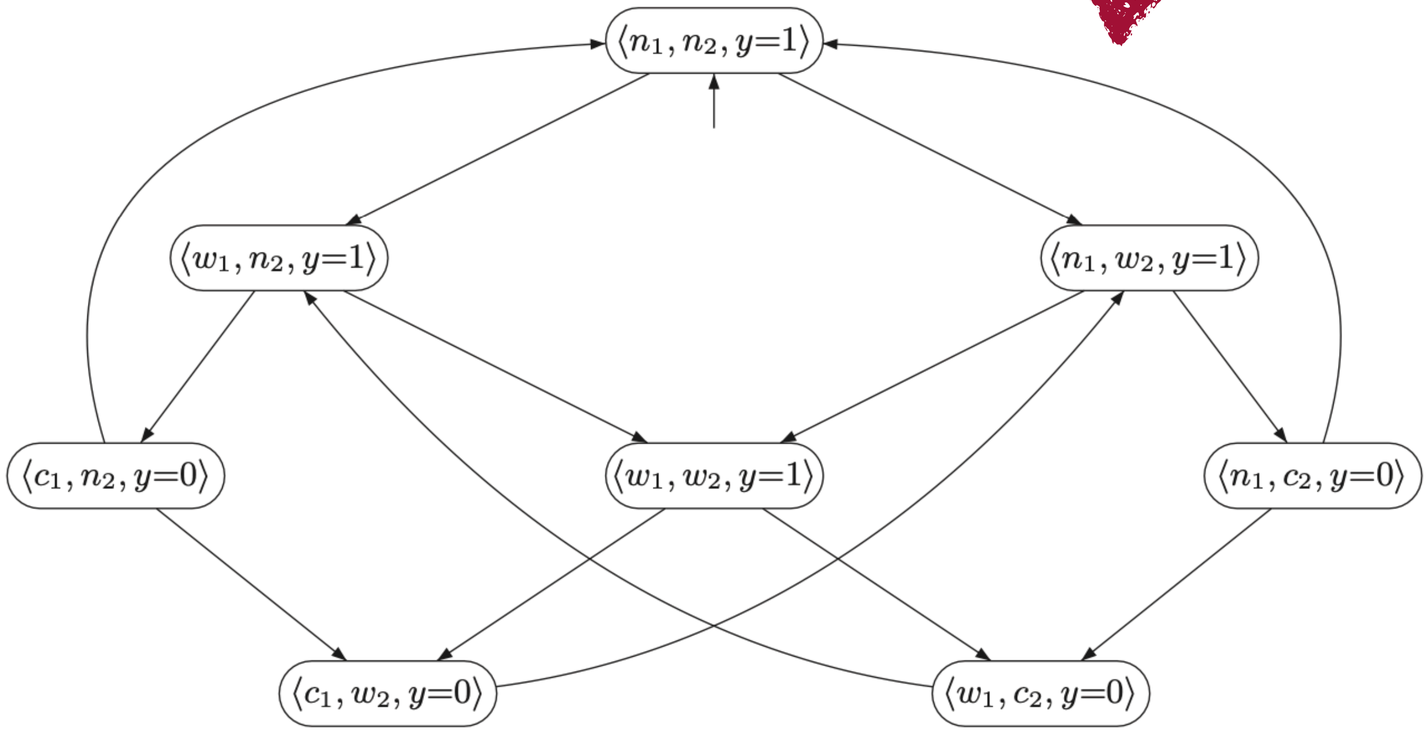


Scalable Algorithms

- Graph Theory,
- Automata Theory,
- Logics,
- ...

Model Checking - Example

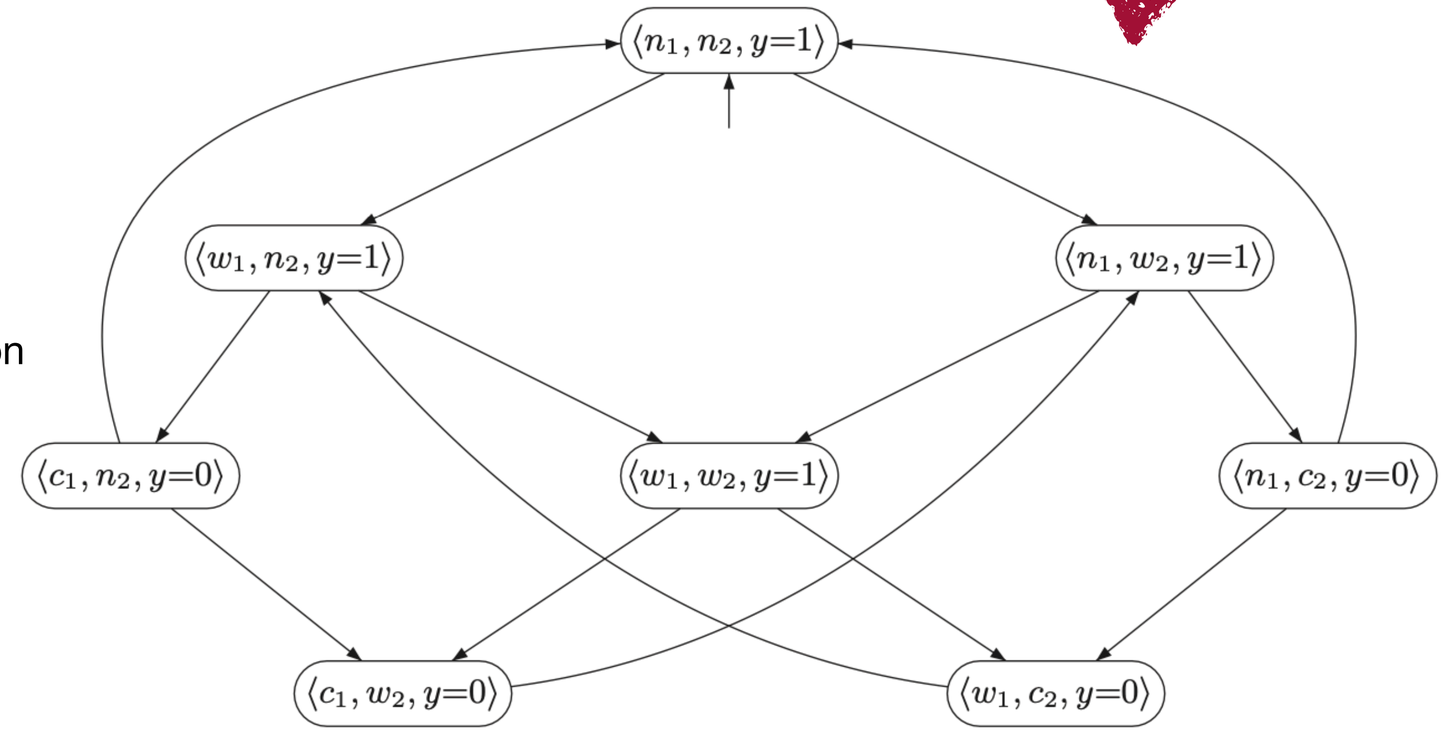
- **Model:** Two processes with a shared semaphore y
  - Each process i performs either noncritical actions n_i, waits w_i, or performs critical actions c_i



Model Checking - Example

- **Model:** Two processes with a shared semaphore y
  - Each process i performs either noncritical actions n_i, waits w_i, or performs critical actions c_i

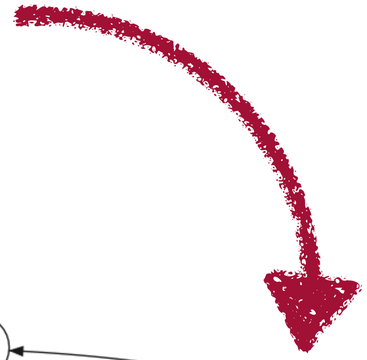
- **Properties**:
  - $\neg\Diamond(c_1 \wedge c_2)$
    "do not each a state in which both processes are in their critical section



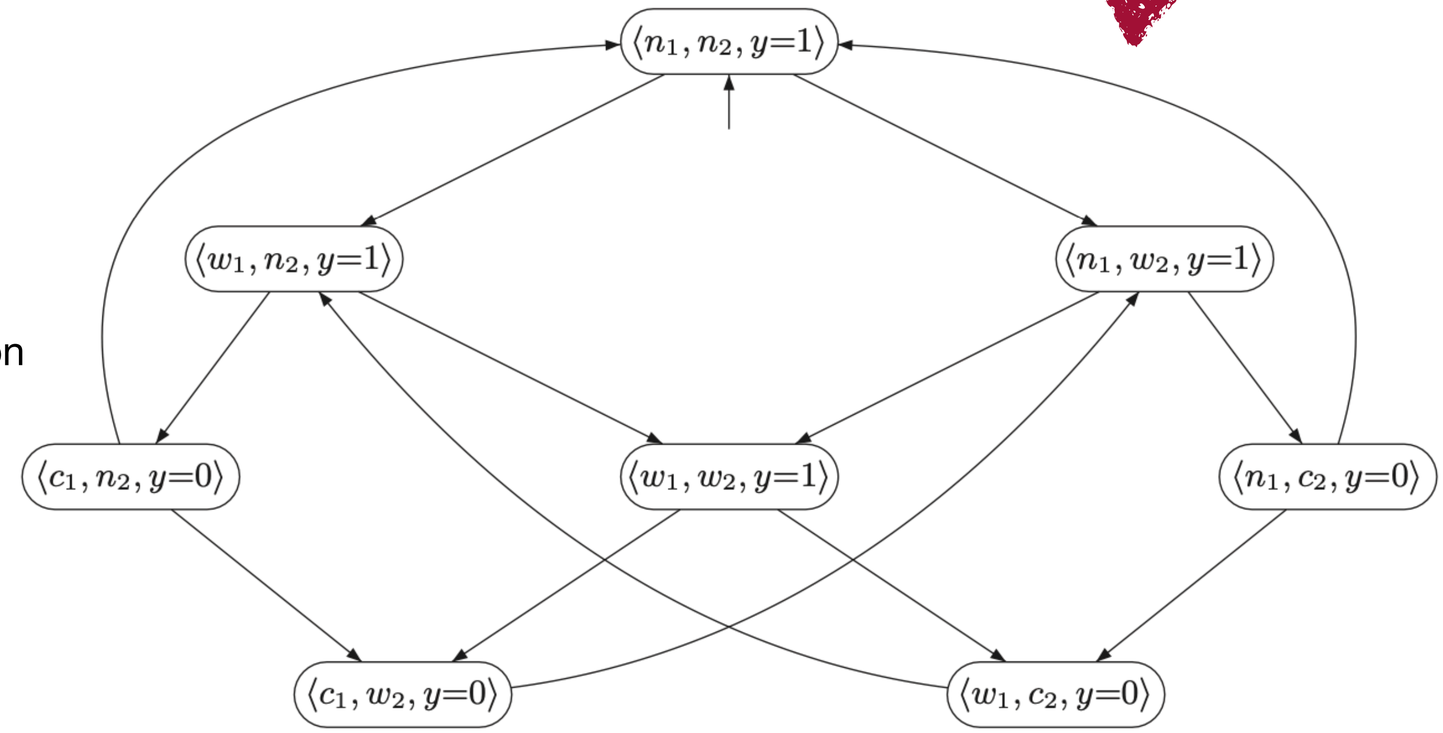
Model Checking - Example

- **Model:** Two processes with a shared semaphore y
  - Each process i performs either noncritical actions n_i, waits w_i, or performs critical actions c_i

- **Properties**:
  - $\neg\Diamond(c_1 \wedge c_2)$

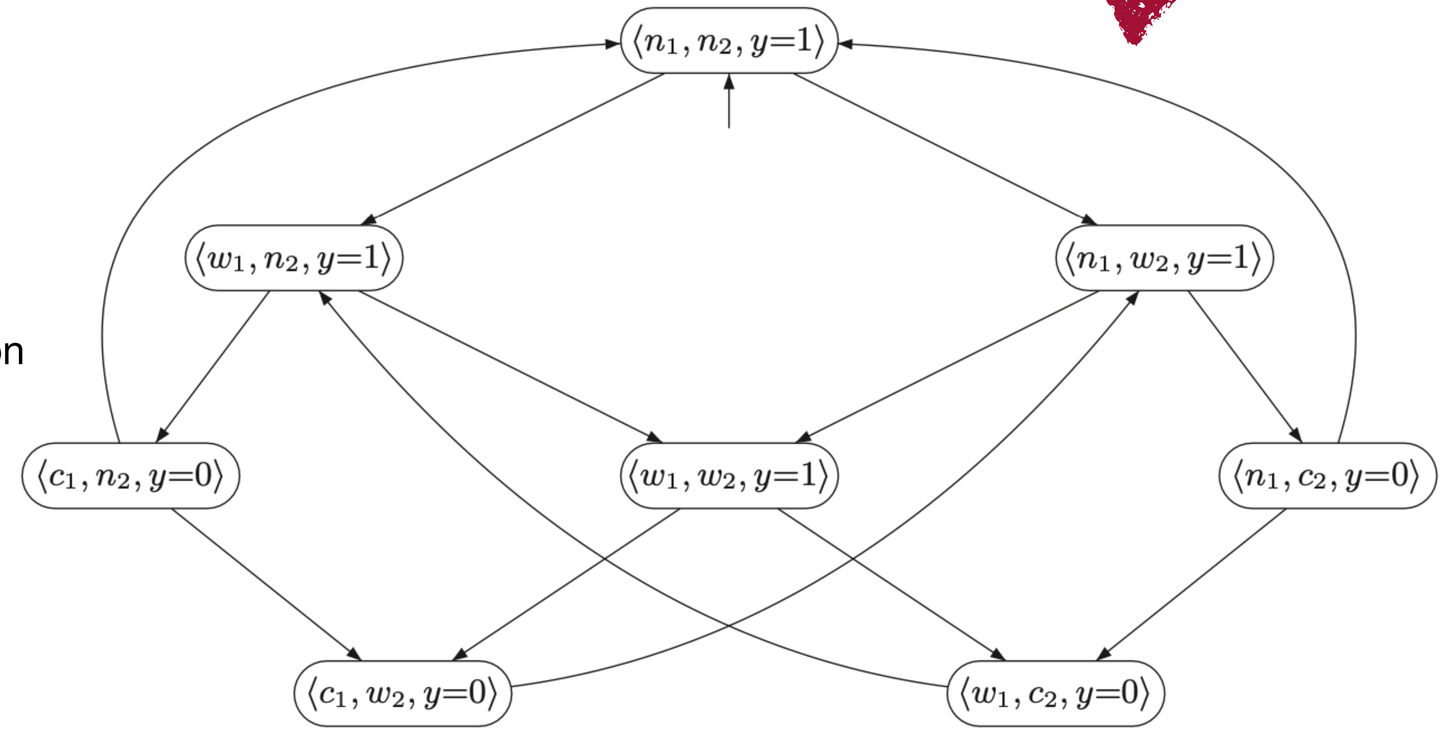


"do not each a state in which both processes are in their critical section



Model Checking - Example

- **Model:** Two processes with a shared semaphore y
  - Each process i performs either noncritical actions n_i, waits w_i, or performs critical actions c_i



- **Properties**:
  - $\neg\Diamond(c_1 \wedge c_2)$

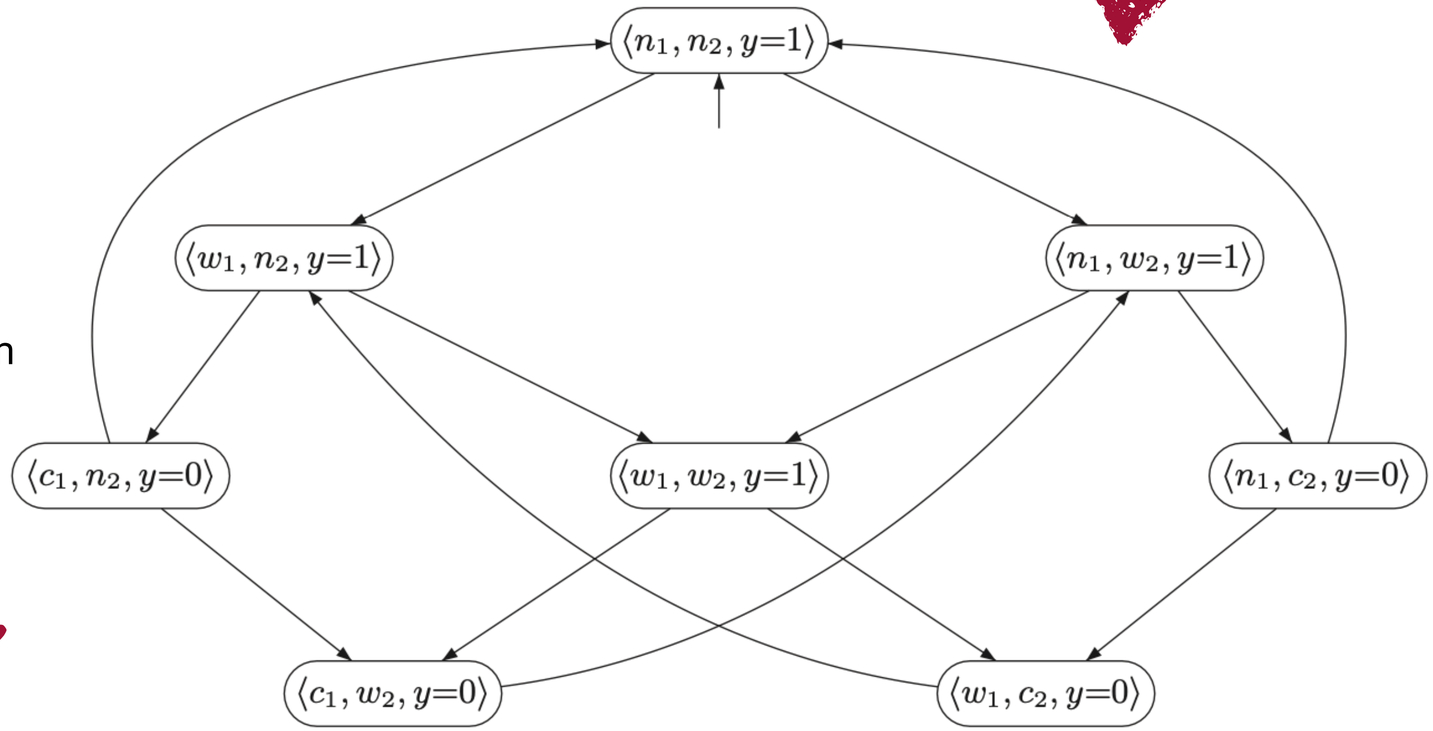
    "do not each a state in which both processes are in their critical section

  - $\Box\Diamond w_i \implies \Box\Diamond c_i$

    "if process i waits infinitely often, it infinitely often enters its critical section"

Model Checking - Example

- **Model:** Two processes with a shared semaphore y
  - Each process i performs either noncritical actions n_i, waits w_i, or performs critical actions c_i



- **Properties**:
  - $\neg \Diamond (c_1 \wedge c_2)$

"do not each a state in which both processes are in their critical section

  - $\Box \Diamond w_i \implies \Box \Diamond c_i$

"if process i waits infinitely often, it infinitely often enters its critical section"

Objectives of this Seminar

- **Independent understanding** of a scientific research article
- Describe the **problem(s)** considered in the article and the necessary **background**
- Explain relevant **research results** using an **adequate level of detail**
- Acquire, read, and understand **related literature**

Write a **report** and give an oral **presentation** covering the above points

- Assume fellow students as target audience

Requirements on Report

- Independent writing of a report of **12 – 15 pages**
  - Font size: **12pt** with "standard" page layout (LaTeX template on website)
  - Do not stretch the content, e.g., with overly sized figures

- First milestone: detailed **outline** + **one page** of content
  - Provide overview of structure (section headers, main definitions/theorems)
  - Be specific — "*1. Introduction / 2. Main part/ 3. Conclusions*" is not enough!
  - Also write one page of actual content (in a main part of the report)

- **Complete** and **correctly cited** set of references to all consulted literature
  - **Plagiarism**: taking text blocks (from literature or web) without source indication causes **immediate exclusion from this seminar**

- Correct **spelling** and **grammar** is required; use german or english
  - More than 10 errors per page lead to abortion of correction

Requirements on Talk

- **Total duration**: 30 minutes (25 minutes presentation time + 5 minutes for Q&A)
  - Finish in time — Overtime is bad

- Focus your talk on the **audience**, abstract away from details as necessary

- Descriptive slides (LaTeX template on website, can also use other software)
  - ≤ 15 lines of text per slide,
  - use (base) colours in a useful manner
  - number your slides

- Correct spelling! (German or English)

- Prepare for expected questions, e.g., with backup slides

Soft Requirements and Hints

- Get an understanding of the **practical** and **theoretical(!)** aspects of the article
  - **Reminder:** this is a seminar in theoretical computer science

- Find the right **level of detail**
  - Your report and talk should be **self-contained** and **understandable** by people not familiar with the original article
  - Focus on **core ideas**, omit too specific details (e.g., related work or optimisations)
  - Provide more extensive explanations, **examples**

- Discuss contents / ideas / problems with your **supervisor**
  - Contact them on time, prepare the meetings

- **Take your time**
  - Seminar yields 4 credit points
  - Officially, this translates to around 4 * 30 = 120 hours of work
  - Do not expect to write the report / prepare the talk in a single day ...

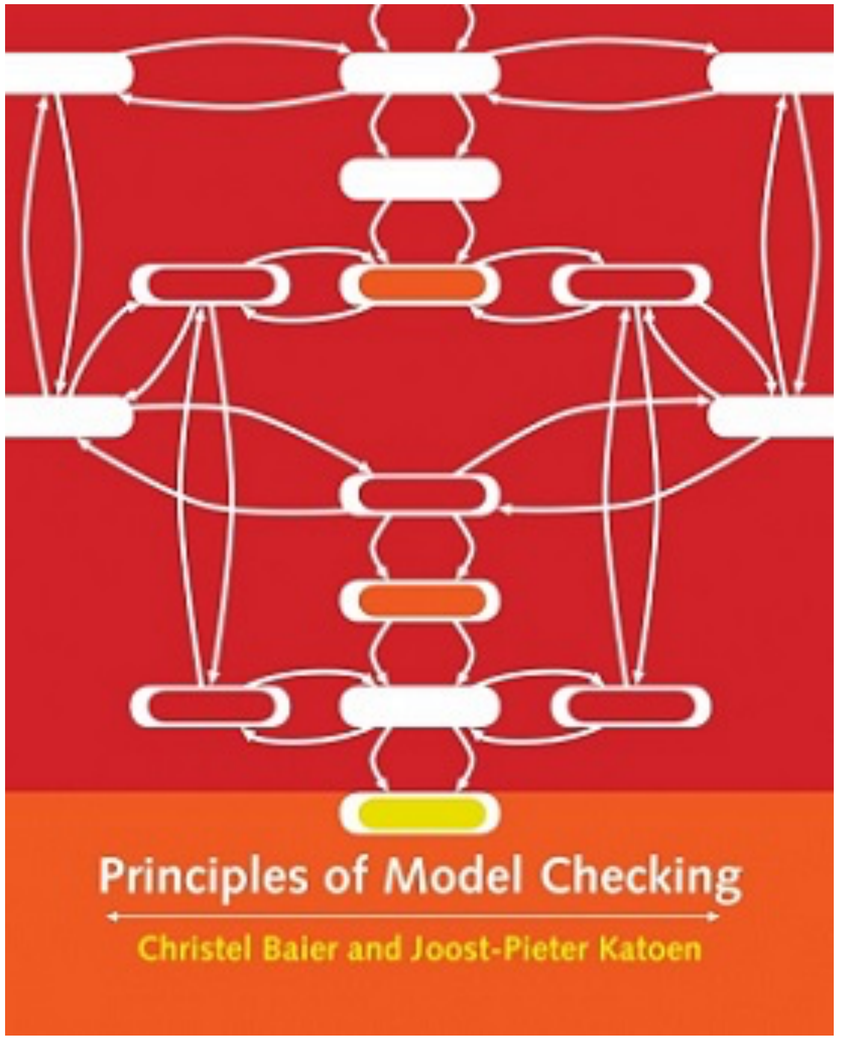
Literature

- See references in research article

- For general background:

  - Baier & Katoen:
    Principles of Model Checking

  - Model Checking Lecture
    (approach me if you want Moodle access)



Important Dates

15.04.2024, 10:30: Topic preferences due (**Monday**); see last slide

06.05.2024: Detailed outline and one page of content due

10.06.2024: Full report due

08.07.2024: Presentation slides due

15.07. to 19.07.2024: Seminar talks (precise date will be announced soon)

Withdrawal

- You have **up to three weeks** to refrain from participating in this seminar

- Later cancellation (by you or by us) causes a not passed for this seminar and reduces your (three) tries by one.

Missing a deadline causes immediate exclusion from the seminar
Please notify us if you decide to quit

Topics

Model Checking Strategies from Synthesis over Finite Traces

Bansal, S., Li, Y., Tabajara, L.M., Vardi, M.Y., Wells, A. (2023).

Abstract. The innovations in reactive synthesis from *Linear Temporal Logics over finite traces* (LTLf) will be amplified by the ability to verify the correctness of the strategies generated by LTLf synthesis tools. This motivates our work on LTLf *model checking*. LTLf model checking, however, is not straightforward. The strategies generated by LTLf synthesis may be represented using *terminating* transducers or *non-terminating* transducers where executions are of finite-but-unbounded length or infinite length, respectively. For synthesis, there is no evidence that one type of transducer is better than the other since they both demonstrate the same complexity and similar algorithms.

In this work, we show that for model checking, the two types of transducers are fundamentally different. Our central result is that LTLf model checking of non-terminating transducers is *exponentially harder* than that of terminating transducers. We show that the problems are EXPSPACE-complete and PSPACE-complete, respectively. Hence, considering the feasibility of verification, LTLf synthesis tools should synthesize terminating transducers. This is, to the best of our knowledge, the *first* evidence to use one transducer over the other in LTLf synthesis.

- Considers LTLf: LTL over finite traces

$$\varphi := \mathsf{true} \mid \mathsf{false} \mid a \in \mathsf{Prop} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathrm{X}\varphi \mid \varphi_1 \mathrm{U} \varphi_2$$

- Automata & Complexity Theory

	LTL	LTLf
Non-deterministic Automata	(NBA) Exponential	(NFA) Exponential
Satisfiability	PSPACE-complete [28]	PSPACE-complete [14]
Synthesis	2EXPTIME-complete [25]	2EXPTIME-complete [13]
Model Checking (NT)	PSPACE-complete [33]	EXPSPACE-complete (New!)
Model Checking (T)	Undefined	PSPACE-complete (New!)

Algorithms for Model Checking HyperLTL and HyperCTL

Finkbeiner, B., Rabe, M.N., Sánchez, C. (2015).

Abstract. We present an automata-based algorithm for checking finite state systems for hyperproperties specified in HyperLTL and HyperCTL*. For the alternation-free fragments of HyperLTL and HyperCTL* the automaton construction allows us to leverage existing model checking technology. Along several case studies, we demonstrate that the approach enables the verification of real hardware designs for properties that could not be checked before. We study information flow properties of an I2C bus master, the symmetric access to a shared resource in a mutual exclusion protocol, and the functional correctness of encoders and decoders for error resistant codes.

HyperLTL and HyperCTL* extend the standard temporal logics LTL and CTL* by *quantification over path variables*. Their formulas are generated by the following grammar, where $a \in \mathsf{AP}$ and π ranges over path variables:

$$\begin{aligned}\varphi \;::=\; & \mathsf{true} \mid a_\pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi \mid \varphi \,\mathcal{R}\, \varphi \mid \exists \pi.\, \varphi \mid \forall \pi.\, \varphi\end{aligned}$$

- HyperLTL/HyperCTL allow to specify relations over executions of the system

- Example: If paths π and π' only differ in h, they have the same output o at all times

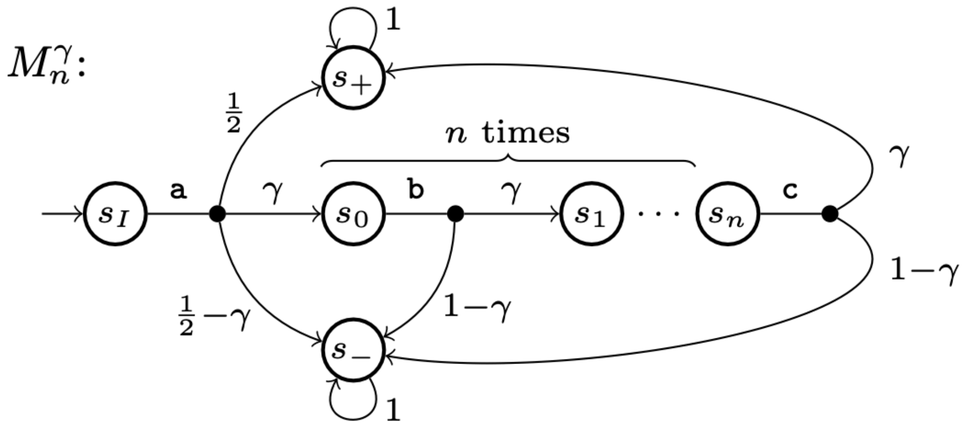
$$\forall \pi . \forall \pi' .\ \square \Big(\bigwedge_{i \in I \setminus h} i_\pi = i_{\pi'} \Big) \ \Rightarrow\ \square \, (o_\pi = o_{\pi'})$$

Correct Probabilistic Model Checking with Floating-Point Arithmetic

Hartmanns, A. (2022)

Abstract. Probabilistic model checking computes probabilities and expected values related to designated behaviours of interest in Markov models. As a formal verification approach, it is applied to critical systems; thus we trust that probabilistic model checkers deliver correct results. To achieve scalability and performance, however, these tools use finite-precision floating-point numbers to represent and calculate probabilities and other values. As a consequence, their results are affected by rounding errors that may accumulate and interact in hard-to-predict ways. In this paper, we show how to implement fast *and* correct probabilistic model checking by exploiting the ability of current hardware to control the direction of rounding in floating-point calculations. We outline the complications in achieving correct rounding from higher-level programming languages, describe our implementation as part of the MODEST TOOLSET's mcsta model checker, and exemplify the trade-offs between performance and correctness in an extensive experimental evaluation across different operating systems and CPU architectures.

- Compute reachability probabilities



- Fixed point theory
- Numerics

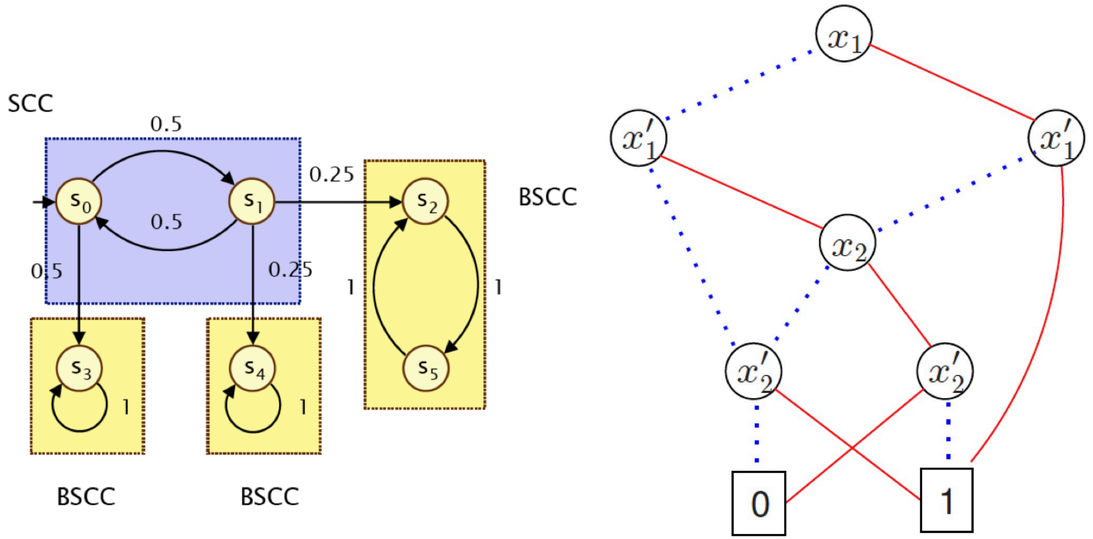
Fast Symbolic Computation of Bottom SCCs

Jakobsen, A.B., Jørgensen, R.S.M., van de Pol, J., Pavlogiannis, A. (2024)

Abstract. The computation of bottom strongly connected components (BSCCs) is a fundamental task in model checking, as well as in characterizing the attractors of dynamical systems. As such, symbolic algorithms for BSCCs have received special attention, and are based on the idea that the computation of an SCC can be stopped early, as soon as it is deemed to be non-bottom.

In this paper we introduce PENDANT, a new symbolic algorithm for computing BSCCs which runs in linear symbolic time. In contrast to the standard approach of *escaping* non-bottom SCCs, PENDANT aims to *start* the computation from nodes that are likely to belong to BSCCs, and thus is more effective in sidestepping SCCs that are non-bottom. Moreover, we employ a simple yet powerful *deadlock-detection* technique, that quickly identifies singleton BSCCs before the main algorithm is run. Our experimental evaluation on three diverse datasets of 553 models demonstrates the efficacy of our two methods: PENDANT is decisively faster than the standard existing algorithm for BSCC computation, while deadlock detection improves the performance of each algorithm significantly.

- Compute Bottom Strongly Connected Components



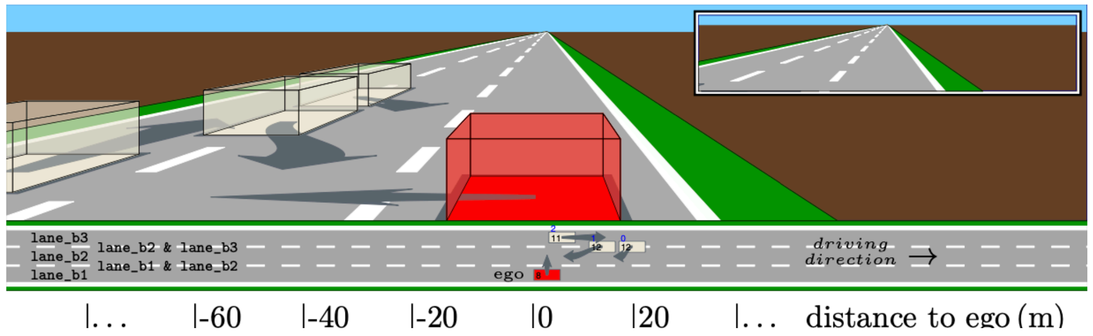
- Graph theory
- Symbolic (BDD) based computations

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

König, L. et al. (2024)

Abstract. Automated driving functions are among the most critical software components to develop. Before deployment in series vehicles, it has to be shown that the functions drive safely and in compliance with traffic rules. Despite the coverage that can be reached with very large amounts of test drives, corner cases remain possible. Furthermore, the development is subject to time-to-delivery constraints due to the highly competitive market, and potential logical errors must be found as early as possible. We describe an approach to improve the development of an actual industrial behavior planner for the *Automated Driving Alliance* between Bosch and Cariad. The original process landscape for verification and validation is extended with model checking techniques. The idea is to integrate automated extraction mechanisms that, starting from the C++ code of the planner, generate a higher-level model of the underlying logic. This model, composed in closed loop with expressive environment descriptions, can be exhaustively analyzed with model checking. This results, in case of violations, in traces that can be re-executed in system simulators to guide the search for errors. The approach was exemplarily deployed in series development, and successfully found relevant issues in intermediate versions of the planner at development time.

- Practical application of Model Checking
- Uses state-of-the-art tools



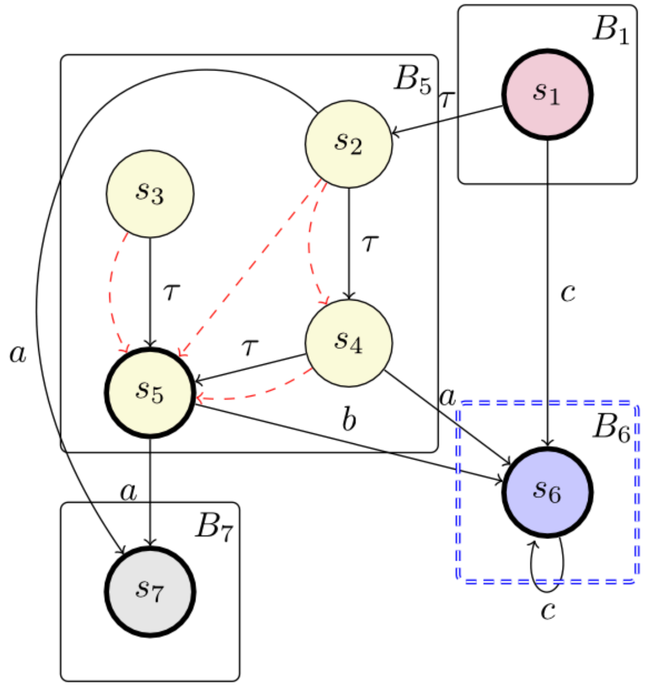
Linear parallel algorithms to compute strong and branching bisimilarity

Martens, J. et al. (2023)

Abstract

We present the first parallel algorithms that decide strong and branching bisimilarity in linear time. More precisely, if a transition system has n states, m transitions and $|Act|$ action labels, we introduce an algorithm that decides strong bisimilarity in $\mathcal{O}(n + |Act|)$ time on $\max(n, m)$ processors and an algorithm that decides branching bisimilarity in $\mathcal{O}(n + |Act|)$ time using up to $\max(n^2, m, |Act|n)$ processors.

- Identify "equivalent" states
- Graph theory, parallel computing

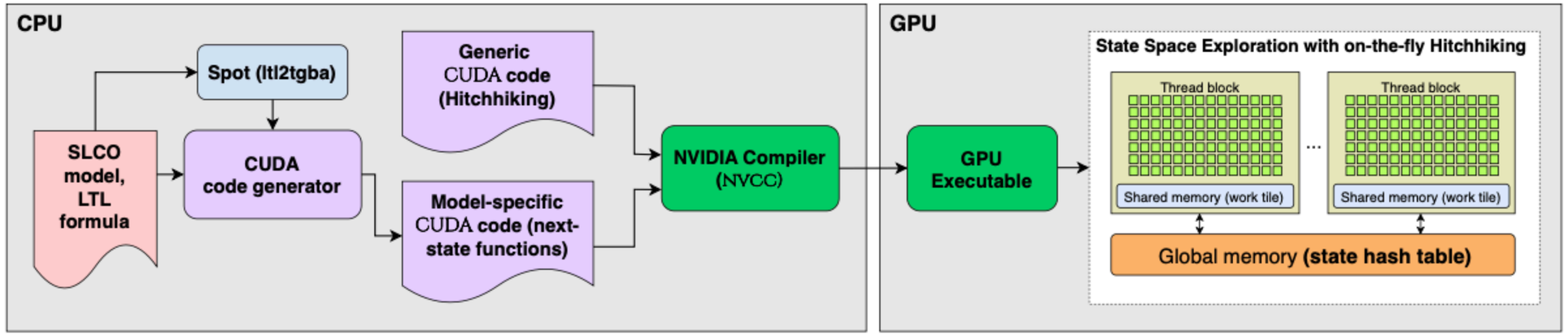


Hitching a Ride to a Lasso: Massively Parallel On-The-Fly LTL Model Checking

Osama, M., Wijs, A. (2024)

Abstract. The need for massively parallel algorithms, suitable to exploit the computational power of hardware such as graphics processing units, is ever increasing. In this paper, we propose a new algorithm for the on-the-fly verification of Linear-Time Temporal Logic (LTL) formulae [45] that is aimed at running on such devices. We prove its correctness and termination guarantee, and experimentally compare a GPU implementation with state-of-the-art LTL model checkers. Our new GPU LTL-checking algorithm is up to 150× faster on proving the correctness of a system than LTSmin running on a 32-core high-end CPU, and is more economic in using the available memory.

- How to implement LTL Model Checking w/ GPUs

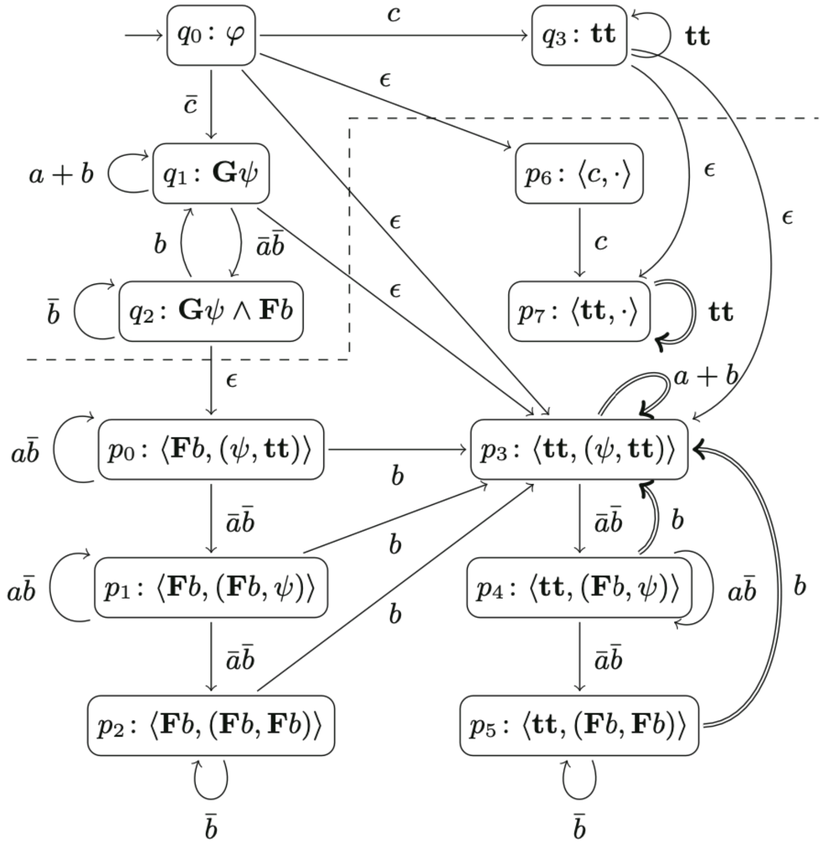


Limit-Deterministic Büchi Automata for Linear Temporal Logic

Sickert, S., Esparza, J., Jaax, S., Křetínský, J. (2016).

Abstract. Limit-deterministic Büchi automata can replace deterministic Rabin automata in probabilistic model checking algorithms, and can be significantly smaller. We present a direct construction from an LTL formula φ to a limit-deterministic Büchi automaton. The automaton is the combination of a non-deterministic component, guessing the set of eventually true $\mathbf{G}$-subformulas of φ, and a deterministic component verifying this guess and using this information to decide on acceptance. Contrary to the indirect approach of constructing a non-deterministic automaton for φ and then applying a semi-determinisation algorithm, our translation is compositional and has a clear logical structure. Moreover, due to its special structure, the resulting automaton can be used not only for qualitative, but also for quantitative verification of MDPs, using *the same* model checking algorithm as for deterministic automata. This allows one to reuse existing efficient implementations of this algorithm without any modification. Our construction yields much smaller automata for formulas with deep nesting of modal operators and performs at least as well as the existing approaches on general formulas.

- Get Büchi Automata from LTL formulae that are "almost" deterministic

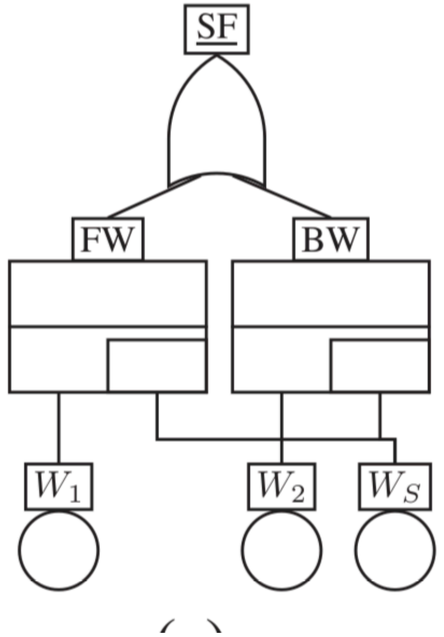


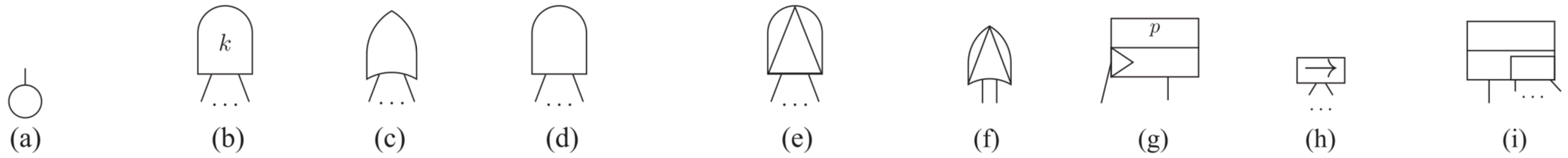
Fast Dynamic Fault Tree Analysis by Model Checking Techniques

Volk, M., Junges, S., Katoen, J.-P. (2017).

***Abstract*—This paper presents a new state-space generation approach for dynamic fault trees (DFTs) that exploits several successful reduction techniques from the field of model checking. The key idea is to aggressively exploit the DFT structure—detecting symmetries, spurious nondeterminism, and don't cares. Benchmarks show a gain of more than two orders of magnitude in terms of state-space generation and analysis time. This fast, scalable approach is complemented by an approximative technique that determines bounds on DFT measures by a partial state-space generation. This is shown to yield another order of magnitude gain while guaranteeing tight error bounds.**

- Create (small) models for dynamic fault trees





Selecting your Topic

- **Enter the poll** in the link you received via email
  - https://terminplaner4.dfn.de/.....
  - Do this until **Monday 15.04.2024, 10:30**

- We do our best to find a *good* topic-student assignment
  - It helps when you indicate multiple topics

- Topic assignment will be announced on **Monday**

We wish you success and look forward to an enjoyable and high-quality seminar!