ETAPS Poster Book

Collection of posters presented at ETAPS 2025

Hamilton, May 3–8, 2025

FASE 2025 Posters

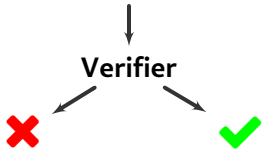
ByteBack: Deductive Verification at the Level of JVM Bytecode

Marco Paganoni · **Carlo A. Furia**

Software Institute, USI Lugano, Switzerland

Deductive Verification

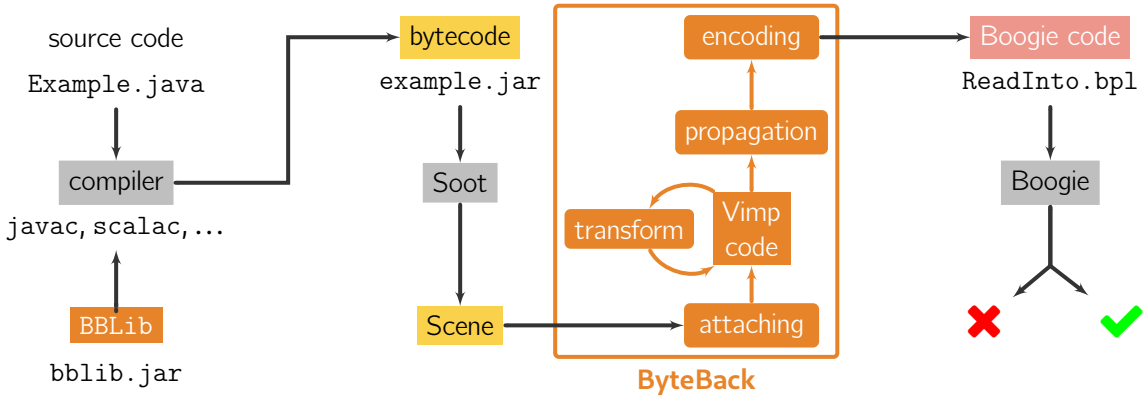
```
@Require(forall j : int • xs[j] ≠ 2)
@Ensure(return ≥ 0)
public int onesMinusTwos(int[] xs) {
    int r = 0;
    for (int i = 0; i < xs.length; ++i) {
        invariant(r ≥ 0)
        switch (xs[i]) {
        case 1: r += 1; break;
        case 2: r -= 1;
        }
    }
    return r;
}
```



Supported Java Features

Feature	Java Version	Support KeY	OpenJML	ByteBack
Generic classes	5	!	✔	✔
Enhanced `for` loop	5	✔	✔	✔
Variable arguments	5	✔	✖	✔
Generic type inference	7	✖	✖	✔
Try-with-resources block	7	✖	✖	✔
Multi-catch block	7	✔	✖	✔
Default methods	8	✔	✔	✔
Local type inference	10	✖	✖	✔
`switch` expressions	12	✖	✖	✔
`yield` in `switch` expressions	13	✖	✖	✔
`instanceof` pattern matching	14	✖	✖	✔
...				

ByteBack's Architecture



https://github.com/atom-sw/byteback

BBlib: Specification in ByteBack

```
@Require("no_twos")     // @require(forall j : int • xs[j] ≠ 2)
@Ensure("nonnegative") // @ensure(return ≥ 0)
public int onesMinusTwos(int... xs) {
  var r = 0;
  for (var x : xs) {
      invariant(gte(r, 0)); // @invariant(r ≥ 0)
      r += switch (x) {
      case 1 -> 1;
      case 2 -> -1;
      default -> 0;
      };
  }
  return r;
}

@Predicate public boolean no_twos(int[] xs)
{ int j = Binding.integer(); return forall(j, neq(xs[j], 2)); }

@Predicate public boolean nonnegative(int[] xs, int r)
{ return gte(r, 0); }
```

Experiments with ByteBack

120 Verified Programs

Language	Count	Size (*LOC*)
Java 8	58	8171
Java 17	23	6489
Scala 2.3	21	2221
Kotlin 1.8	18	2303

Benchmarks

	Size (*LOC*) Source	Boogie	Time (*s*) Encoding	Verification
Total	1984	1031874	66.3	104.7
Average	159	8598	552.7	872.5

ROBOSTAR

Holly Hendry, Ana Cavalcanti,
Cade McCall, Mark Chattington
holly.hendry@york.ac.uk

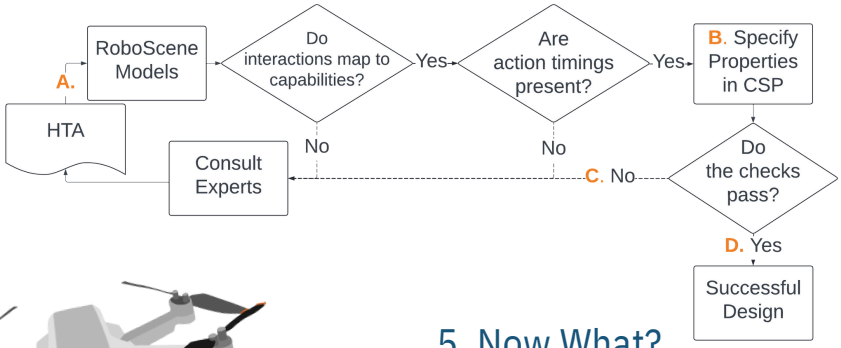
Human-Robot Interaction in the Design and Verification of Robotic Systems

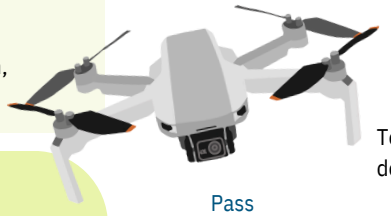
Problem Space

- Modelling and verification of robotic systems lessens the risk of a critical failure
- Desirable properties of a robotic system may depend on assumptions about the behaviour of humans
- Verification of properties of such robotic systems requires a model of expected human behaviour
- Existing techniques for this modelling:
  - need expert knowledge
  - don't cover the whole system: human, software, hardware, scenario
  - don't support formal verification
- Any approach should facilitate cross-discipline communication, without the loss of precision

Approach

The steps of our design and verification approach:





User Needs Analysis

Conducting (n=14) industry interviews on HRI design approaches identified:

- 26 processes
- 20 tools
- 14 standards
- 21 human traits

No common approach or standard

5. Now What?

Testing the system complies with the property we have defined results in one of two outcomes:

Pass

- Automatic generation of artefacts: simulation, tests, proof of properties
- Production of rigorous evidence of system properties, useful in safety cases

Fail

- Indication of an issue within one of the models, whether that be an expectation on the software, hardware, or human
  - Can be used to inform the design of the system through highlighting failure cases
    - Provide communication points between software, hardware and human factors engineers

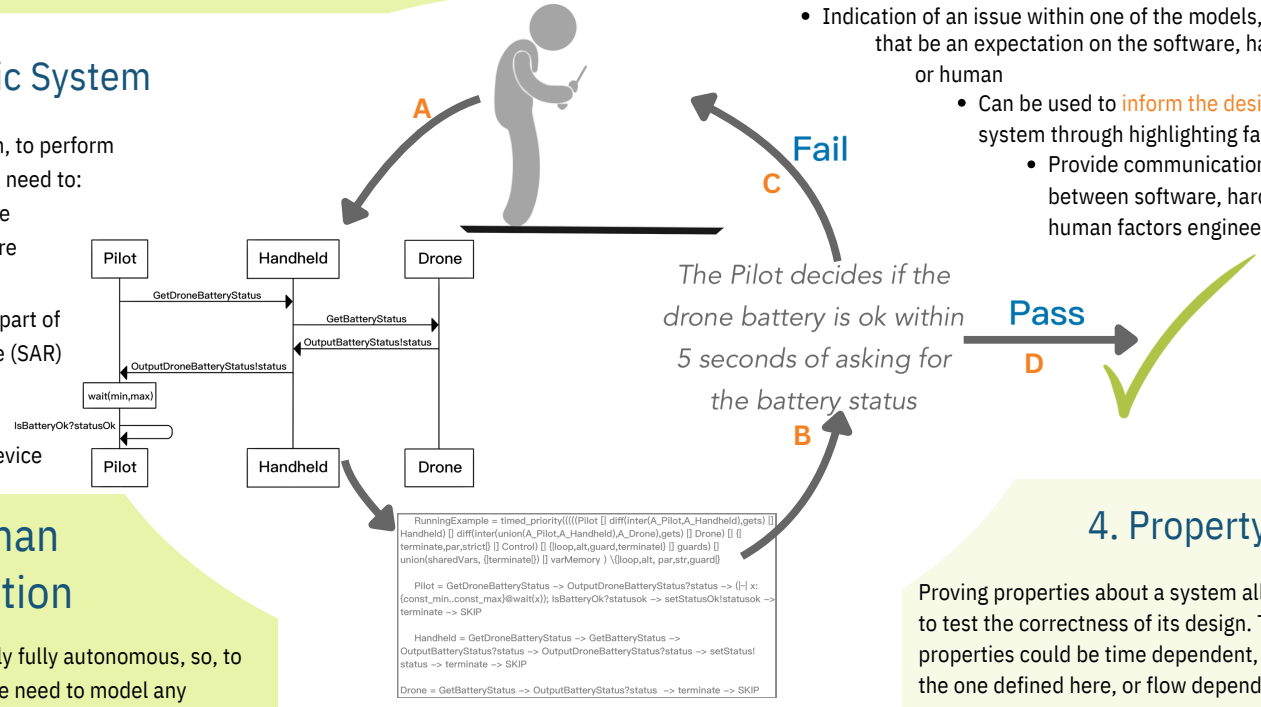
1. Robotic System

For any robotic system, to perform formal verification, we need to:

- Model the software
- Model the hardware

———

We demonstrate here part of our search-and-rescue (SAR) example where a Pilot monitors Drone flight through a Handheld device



2. Human Interaction

No system is currently fully autonomous, so, to model the system, we need to model any expected human interaction.

Our notation, *RoboScene*:

- Applicable to any use case
- Features to capture decision making
- Understandable for various stakeholders in the robotics development lifecycle
- Formal semantics: timed, process algebraic

————————

The Pilot requests the battery status of the drone from the Handheld. The Handheld returns it to the Pilot who then, after the passage of non-deterministic time, decides if the status is ok.

3. Behind the Scenes

Verification through formal methods, and with an automatically generated model of the human, requires:

- Creation of a metamodel to formalise the structure of *RoboScene*
- Definition of formal, mathematical, semantics for *RoboScene*
- Development of a tool to automatically translate from *RoboScene* into a mathematical notation (*CSP*)
- Connection of the human, software and hardware models

4. Property

Proving properties about a system allows us to test the correctness of its design. These properties could be time dependent, such as the one defined here, or flow dependent like:

After the Pilot requests the battery status, no less than 20 seconds passes before successful completion of the request.

Our approach enables you to:

- Prove properties of the system dependent on human interaction
- Mathematically prove these properties
- Receive counterexamples in the case of property failures (FDR)

——————

Properties of the SAR example are derived from human factors models

robostar.cs.york.ac.uk
University of York
THALES
Royal Academy of Engineering
UKRI Engineering and Physical Sciences Research Council

Hybridize Functions: A Tool for Automatically Refactoring Imperative Deep Learning Programs to Graph Execution

Raffi Khatchadourian[1,2] Tatiana Castro Vélez[2] Mehdi Bagherzadeh[3] Nan Jia[2] Anita Raja[1,2]

[1]CUNY Hunter College, USA (ponder@hunter.cuny.edu) [2]CUNY Graduate Center, USA [3]Oakland University, USA

Introduction

- As Deep Learning (DL) datasets grow, efficiency becomes essential to support responsiveness [16].
- Traditionally, DL frameworks embraced *deferred* execution-style DL code for fast execution.
- Hybrid approaches [2, 8, 13] execute *imperative* DL programs quickly.

Hybridization

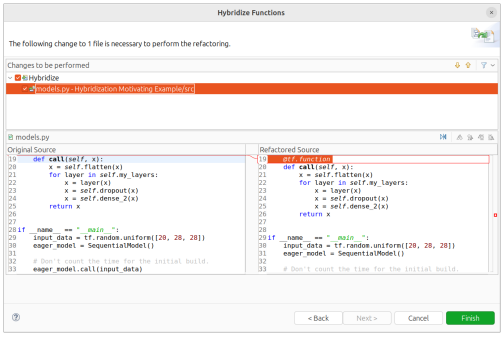


Figure: Screenshot of the HYBRIDIZE FUNCTIONS refactoring preview wizard.

In *TensorFlow* [1], *AutoGraph* [13] can enhance run-time performance by decorating (annotating) appropriate Python function(s) with `@tf.function` (Fig. 1).

Problems with Hybrid Approaches

- Require non-trivial metadata [12].
- Exhibit limitations and known issues with native program constructs [9].
- Are difficult to use correctly and efficiently (e.g., avoiding side-effects) [4].
- Developers *manually* specifying which functions are converted.

Insight

Although imperative DL code typically executes sequentially, hybridization resembles *parallelizing* traditional sequential code.

Automated Tool

We design and implement a fully automated, open-source refactoring tool named HYBRIDIZE FUNCTIONS [11] that transforms otherwise *eagerly*-executed imperative (Python) DL code for enhanced performance.

Contributions

- Refactoring approach for automatically converting imperative DL code to graphs.
- Novel tensor analysis for *imperative* DL.
- Fully automated, open-source tool implemented as a *PyDev* [15] *Eclipse* [7] IDE plug-in that integrates static analyses from *WALA* [14] and *Ariadne* [6].

Architecture & Dependencies

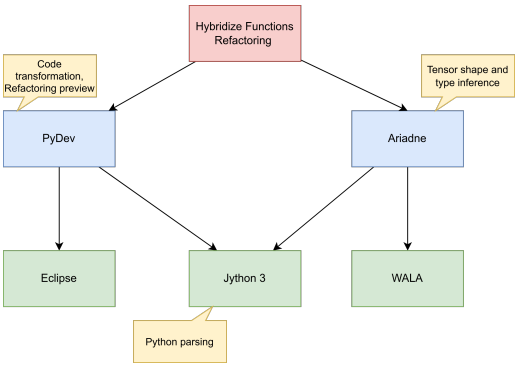


Figure: Overall architecture.

- *Eclipse* is leveraged for its existing, well documented and integrated refactoring framework and test engine [3], including transformation APIs (e.g., `ASTRewrite`), refactoring preview pane (Fig. 1), precondition checking (e.g., `Refactoring.checkInitialConditions()`, `Refactoring.checkFinalPreconditions()`), and refactoring testing (e.g., `RefactoringTest`).
- *PyDev* used for efficient program entity indexing, extensive refactoring support [3], and that it is completely open-source for all Python development.
- *WALA* is used for static analyses, such as ModRef, for which we built our side-effect analysis upon.
- *Ariadne*, which depends on *WALA*, is used for its Python and tensor analysis, including type inference and (*TensorFlow*) library modeling.

Challenges Addressed

- Reworked much of the existing Java (JDT) refactoring tooling to work with Python.
- Integrated *Ariadne* with *PyDev* due to its excellent and long-lived refactoring support for Python, including refactoring preview pane, element GUI selection, and refactoring undo history.
- Augmented *Ariadne* to analyze imperative Deep Learning (Python) code by vastly expanding the XML summaries to support a wide variety of popular *TensorFlow* 2 APIs.
- Added support for Python constructs commonly used in *modern* imperative DL programs.
- Correlated varying intermediate representations (IRs) with the original Python source code.

Modernizing *Ariadne*: New Enhancements

- Python module packages.
- Wild card imports.
- Intra-package references (relative imports; `from .. import X`).
- Package initialization scripts.
- Automatic unit test entry points discovery.
- Non-scalar tensor dataset [10] iteration.
- Modeling of additional libraries.
- Static and class methods analysis.
- Analysis of custom decorators.
- Callable object (functor) analysis (used in *Keras*).

Evaluation Summary

- We applied our approach to 19 open-source Python imperative DL programs of varying size and domain, with thousands of source lines of code ranging from 0.12 to 36.72.
- Our tool considered 766 Python functions, automatically refactoring 42.56% despite being highly conservative.
- During a run-time performance evaluation, we measured an average relative model training speedup of 2.16 (memory consumption measurement pending).
- Differences in model accuracy and loss before and after refactoring were negligible.

Conclusion

- Open-source, automated refactoring *PyDev Eclipse* plug-in, HYBRIDIZE FUNCTIONS, assists developers with writing optimal imperative DL Python code.
- Integrates an Eclipse refactoring with *WALA Ariadne* Python static analyses.

Future Work

- Explore incorporating advanced container-based analyses.
- Automatically split functions.

References

1. Abadi, M. *et al.*: TensorFlow: A System for Large-Scale Machine Learning. In: OSDI (2016)
2. Apache, Hybridize. Apache MXNet documentation. (2021). https://mxnet.apache.org/versions/1.8.0/api/python/docs/tutorials/packages/gluon/blocks/hybridize.html (visited on 04/08/2021)
3. Bäumer, D. *et al.*: "Integrating refactoring support into a Java development tool".
4. Castro Vélez, T. *et al.*: Challenges in Migrating Imperative Deep Learning Programs to Graph Execution: An Empirical Study. In: MSR. MSR '22. ACM (2022). https://doi.org/10.1145/3524842.3528455
5. Chollet, F.: Deep Learning with Python. Manning (2020)
6. Dolby, J. *et al.*: Ariadne. Analysis for Machine Learning Programs. In: MAPL, pp. 1–10. ACM (2018)
7. Eclipse Foundation, Eclipse IDE. (2024). https://eclipseide.org/ (visited on 09/10/2024)
8. Facebook Inc., PyTorch. TorchScript. en. (2019). https://pytorch.org/docs/stable/jit.html
9. Google LLC, Better performance with tf.function. (2021). https://tensorflow.org/guide/function
10. Google LLC, tf.data.Dataset. TensorFlow. Version 2.9.3. (2023). https://www.tensorflow.org/versions/r2.9/api_docs/python/tf/data/Dataset (visited on 12/15/2023)
11. *Hybridize-Functions-Refactoring*. (2024). https://github.com/ponder-lab/Hybridize-Functions-Refactoring (visited on 09/30/2024).
12. Jeong, E. *et al.*: Speculative Symbolic Graph Execution of Imperative Deep Learning Programs. SIGOPS Oper. Syst. Rev. **53**(1), 26–33 (2019). https://doi.org/10.1145/3352020.3352025
13. Moldovan, D. *et al.*: AutoGraph: Imperative-style Coding with Graph-based Performance. (2019). arXiv: 1810.08061 [cs.PL].
14. *T.J. Watson Libraries for Analysis*. (2024). https://github.com/wala/WALA (visited on 09/10/2024). original-date: 2012-04-05T18:57:03Z.
15. Zadrozny, F.: PyDev. (2023). https://www.pydev.org (visited on 05/31/2023)
16. Zhou, W. *et al.*: HARP: Holistic Analysis for Refactoring Python-Based Analytics Programs. In: ICSE (2020). https://doi.org/10.1145/3377811.3380434

Acknowledgments This material is supported in part by the National Science Foundation under awards CCF 2200343, CNS 2213763, and CCF 2343750.

Symbolic State Partitioning for Reinforcement Learning

Mohsen Ghaffari[1], Mahsa Varshosaz[1], Einar Broch Johnsen[2], Andrzej Wąsowski[1]

[1] IT-University of Copenhagen, Copenhagen, Denmark

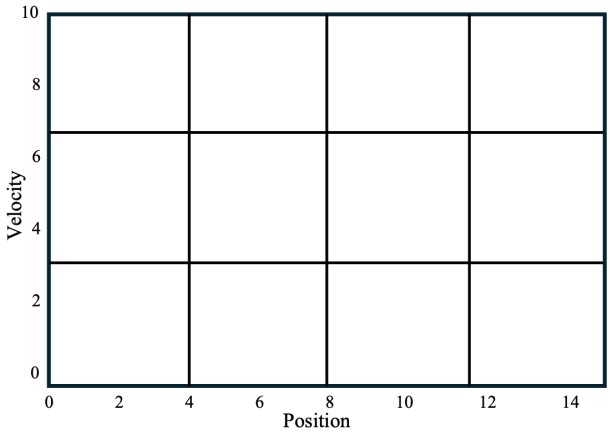
[2] Univesity of Oslo, Oslo, Norway

Talk: FASE, Tuesday, 11:00

Full Paper

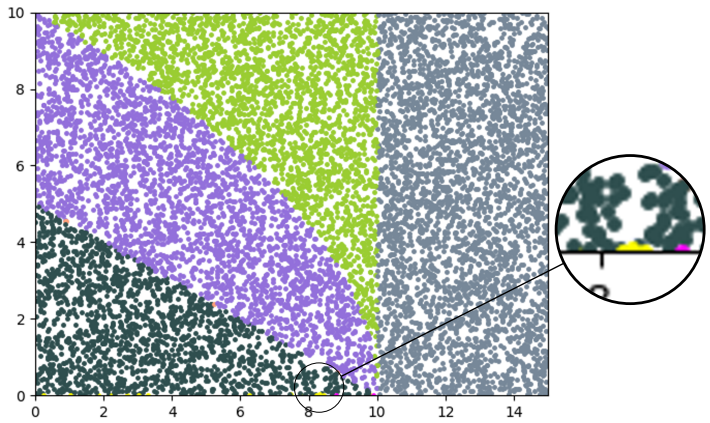
Problem: How to partition the state space for training an agent efficiently using Tabular Reinforcement Learning?

State of the art: Tile coding



The state-of-the-art partitioning methods are not adaptive to the structure of the state space.

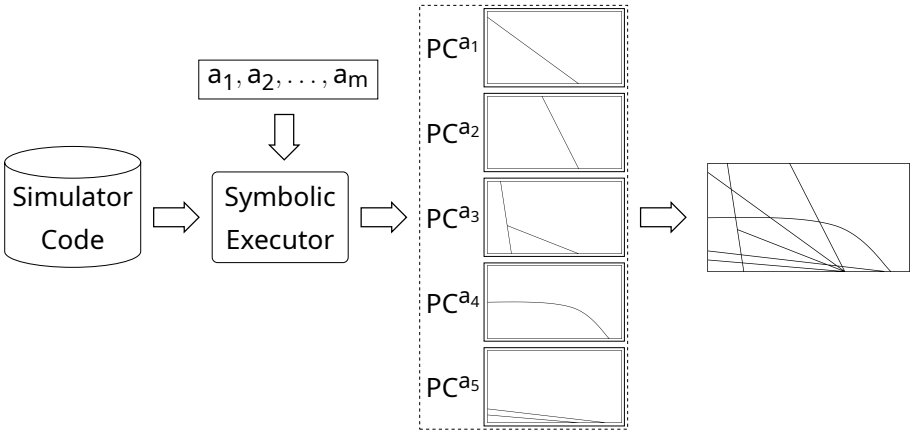
Contribution of this paper: SymPar



Capturing nonlinear dependencies between state components, and **finding narrow parts**.

SymPar

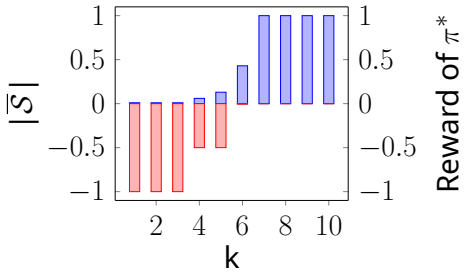
- The environment is modeled as a computer program that implements a single-step transition function, producing the next state and the corresponding reward,
- For each concrete action, the environment simulator is symbolically executed to generate a set of partitions over the state space. SymPar then computes the intersection of these partitions as its final output,
- **Symbolic Execution** extends normal execution by running the operators of a language using symbolic variables and producing symbolic formulas (PC) as output,
- Symbolic execution computes semantic based partition.



Experiments Results

How does granularity of partition affect learning performance?

Increasing the search depth of the symbolic executor leads to finer-grained partitions produced by SymPar. This higher granularity enhances learning performance, as reflected in improved accumulated rewards.



How does SymPar scale with state space sizes?

The number of partitions generated by SymPar remains constant as the state space grows, demonstrating scalability with respect to state space size.

	$\lvert\overline{\mathcal{S}}\rvert$	$\lvert\mathcal{S}\rvert$		$\lvert\overline{\mathcal{S}}\rvert$	$\lvert\mathcal{S}\rvert$		$\lvert\overline{\mathcal{S}}\rvert$	$\lvert\mathcal{S}\rvert$
Simple Maze	10×10	33	**Wumpus World**	64×64	73		10×10	51
	10^2×10^2	33		10^2×10^2	73	**Navigation**	10^2×10^2	51
	10^3×10^3	33		10^3×10^3	73		10^3×10^3	51

FoSSaCS 2025 Posters

Fair Quantitative Games

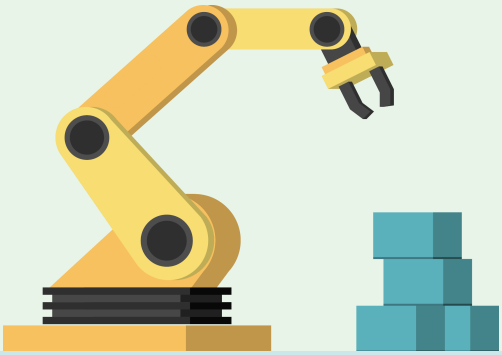
Ashwani Anand, Satya Prakash Nayak, Ritam Raha, Irmak Sağlam, and Anne-Kathrin Schmuck

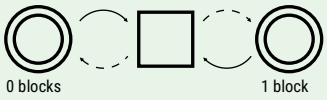
System (Player 1)

Environment (Player 2)

Fairness assumptions eliminate unrealistic scenarios.

Fairness: Whenever the source node of a dashed edge is taken infinitely often, the dashed edge is also taken infinitely often.





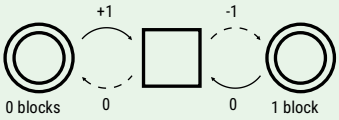
Can we guarantee the robot will put a block at the mark, and remove it, infinitely often?

Qualitative
Safety, Parity, Rabin…

Fairness is easy on qualitative games

Can be solved in **same time complexity** as the original game [1, 2]

Fairness in Synthesis



Can we guarantee the robot will put a block at the mark, and remove it, infinitely often,
without the battery running out,
for some initial value of the battery?

Quantitative
Energy, Mean-payoff

What about fairness on quantitative games?

Fairness on system can be solved in **super-exponential time** using current approaches, whereas there is no known approach for fairness on environment.

Contributions

	Determined?	Complexity (Pseudopolynomial)	Reduction
1-fair MP	Yes	$O(n^3mW)$	To MP on 6n nodes and max absolute weight
2-fair MP	Yes	$O(n^3mW)$	To MP on 6n nodes and max absolute weight
1-fair Energy	Yes	$O(n^4mW)$	To Energy on 8n nodes and max absolute weight
2-fair Energy	No	$O(n^3mW)$	Player 1 winning region reduces to that of an energy on the same graph, Player 2 winning region reduces to that of 2-fair MP game on the same graph

Fairness in Quantitative Games

A play ρ is **fair** iff for every node $v \in inf(\rho)$ that has fair *(dashed)* outgoing edges $E^f(v) \neq \emptyset$, $E^f(v) \subseteq inf(\rho)$.

1-Fair Games: Player 1 nodes have fair outgoing edges.

2-Fair Games: Player 2 nodes have fair outgoing edges.

1-Fair Mean Payoff

Does there exist a strategy σ such that, long run average payoff of every σ-play is **non-negative AND the play is fair?**

1-Fair Energy

Does there exist an initial credit **c** and a strategy σ such that, total energy level along every σ-play stays **non-negative AND the play is fair?**

2-Fair Mean Payoff

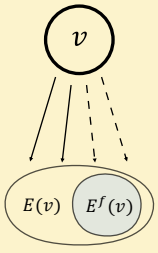
Does there exist a strategy σ such that, long run average payoff of every σ-play is **non-negative OR the play is NOT fair?**

2-Fair Energy

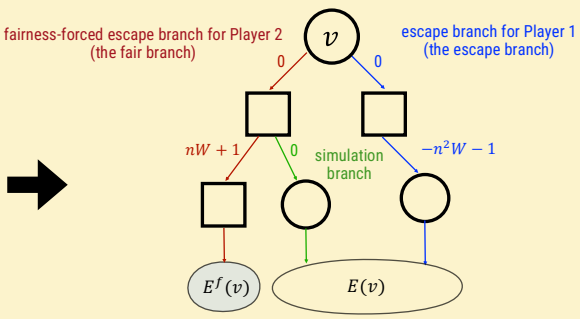
Does there exist an initial credit **c** and a strategy σ such that, total energy level along a play stays **non-negative OR the play is NOT fair?**

Gadgets for Reducing Fair Games to Standard games

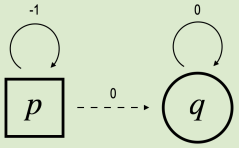
For each fair node v in the 1-fair MP game



replace v with the following v-gadget



2-fair energy games are not determined



A node is won by Player 1 if there exists a Player 1 strategy σ and a credit c s.t. every σ – play is won by Player 1 for credit c.

A node is won by Player 2 if there exists a Player 2 strategy π s.t. every π – play's total payoff goes below $-c$ for every credit c.

[1] Banerjee, T., Majumdar, R., Mallik, K., Schmuck, A., Soudjani, S.: Fast symbolic algorithms for omega-regular games under strong transition fairness. TACAS'22
[2] Hausmann, D., Piterman, N., Saglam, I., Schmuck, A.: Fair ømega-regular games. FoSSaCS'24

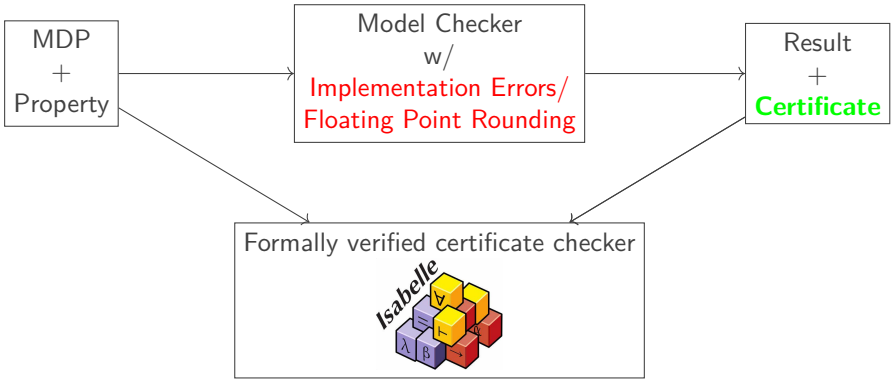
MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

TACAS 2025 Posters

Fixed Point Certificates for Reachability and Expected Rewards in MDPs

Krishnendu Chatterjee, Maximilian Weininger, Tim Quatmann, Tobias Winkler, Maximilian Schäffeler, and Daniel Zilken

What even are Certificates?

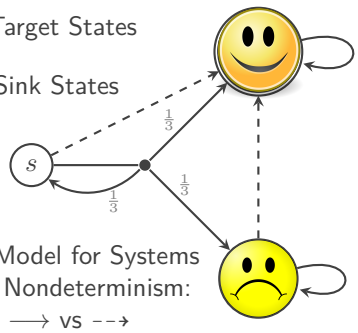


Easy-to-check proofs of the verification results!

MDPs (Markov decision processes)

: Target States

: Sink States



The Model for Systems with Nondeterminism:

$\longrightarrow$ vs $\dashrightarrow$

and Probabilities:

$\xrightarrow{\frac{1}{3}} + \xrightarrow{\frac{1}{3}} + \xrightarrow{\frac{1}{3}} = \xrightarrow{1}$

Let's Play! - The Reachability Game

(2)

$\mathbb{P}^{\min} = 0$

$\mathbb{P}^{\min} > 0$

$\mathbb{P}^{\max} > 0$

$\mathbb{P}^{\max} = 0$

Connect wisely

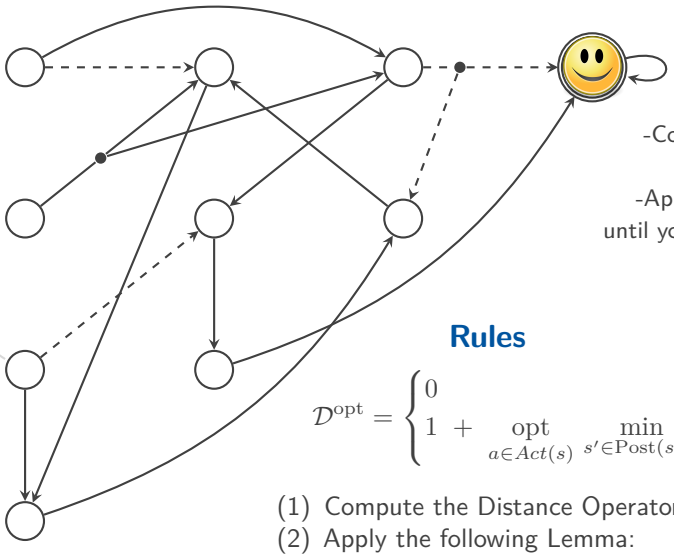
(1)

$\mathcal{D}^{\min} = \infty$

$\mathcal{D}^{\max} = \infty$

$\mathcal{D}^{\max} < \infty$

$\mathcal{D}^{\min} < \infty$



Hints

-Consider $\mathrm{opt} = \min$ and $\mathrm{opt} = \max$ separately

-Apply $\mathcal{D}^{\mathrm{opt}}$ to each state until you reach a fixed point

Rules

$$\mathcal{D}^{\mathrm{opt}} = \begin{cases} 0 & \text{if } s \in \text{☺} \\ 1 + \underset{a \in Act(s)}{\mathrm{opt}} \; \min_{s' \in \mathrm{Post}(s,a)} r(s') & \text{ow.} \end{cases}$$

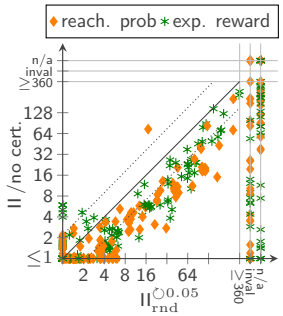
(1) Compute the Distance Operator ($\widehat{\ }$) for each state

(2) Apply the following Lemma:

$$\mathcal{D}^{\mathrm{opt}} = \infty \iff \mathbb{P}^{\overline{\mathrm{opt}}} = 0$$

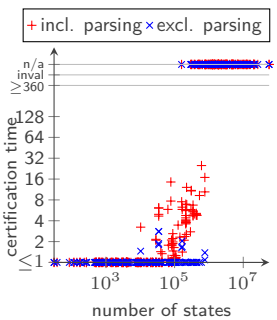
Experimental Evaluation

- Interval Iteration: No Cert vs. Cert



Scalability

- Certification Time vs. #states



Conclusion

- We developed *fixed point certificates* as a new standard for **Certified Model Checking**
- We compute certificates with **rational** value vectors and check them with **exact, arbitrary precision** arithmetic
- Scalable to $\approx 10^6$ **states** with a runtime within ≈ 30 **sec.**
- Soundness formalized in **Isabelle/HOL**
- CT: daniel.zilken@cs.rwth-aachen.de

Institute of Science and Technology Austria

Technische Universität München

RWTH AACHEN UNIVERSITY

GPU accelerated probabilistic model checking

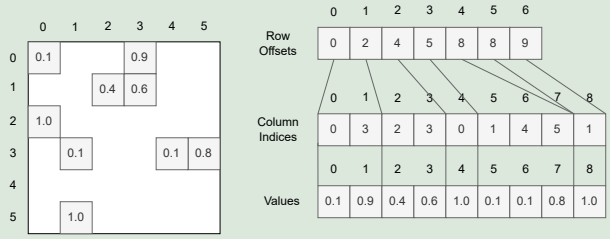
Jan Heemstra (j.h.heemstra@tue.nl), Anton Wijs (a.j.wijs@tue.nl)

Objectives

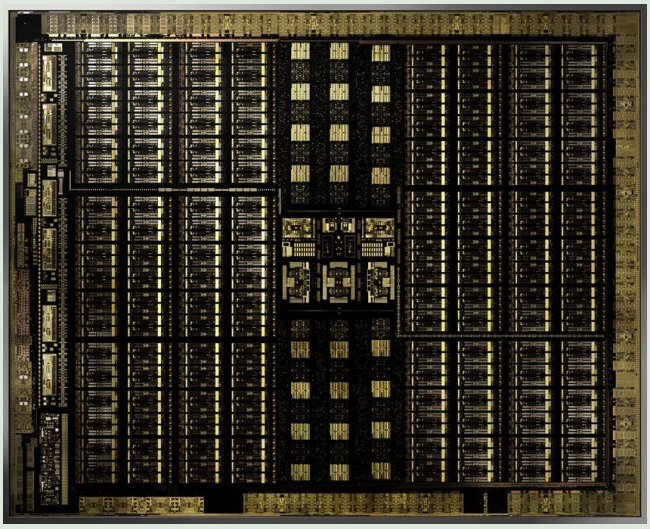
- Perform probabilistic model checking **entirely** on a **GPU**.
- **Previous achievements with the GPUexplore model checker:**
  - Use hardware acceleration to speed process up [4].
  - Verify specification adherence with LTL formulae [3].
- **New:** compute the **probability** of invalid system behavior with PCTL formulae [1, 2].

Matrix construction

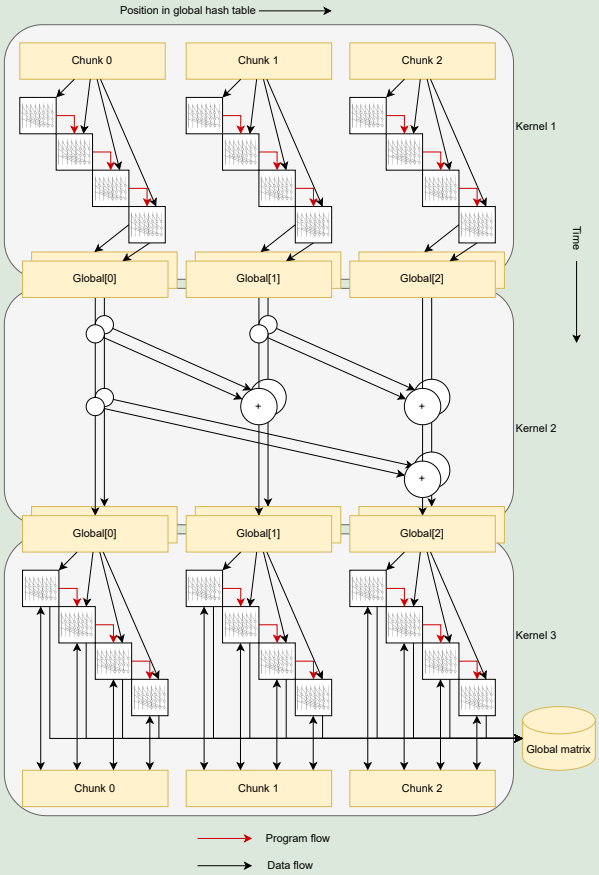
- **Goal:** construct (sparse) transition matrix on a GPU.
- **Input:** explored states are chaotically distributed over hash table.
- Sparse matrix is stored in memory using **CSR format**.



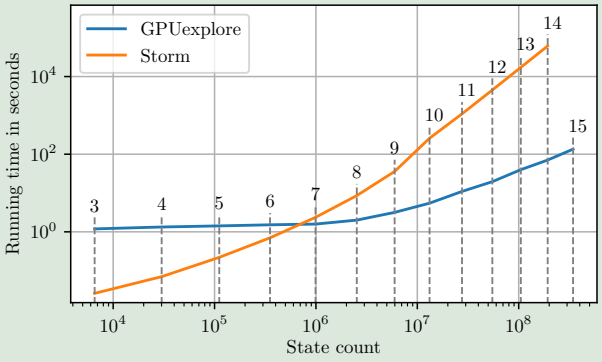
- Memory usage is main constraint.
- On-chip (a.k.a. shared) memory is fast and low latency.



- Two prefix sums are needed: for row offsets and column indices.
- Interweaved prefix sum optimizing both memory access latency and usage.



- After matrix construction, the CUSPARSE library is used to perform matrix-vector multiplications, for the verification of PCTL formulae.
- **Up to 855× faster than Storm, fastest on larger models.**



References

[1] Heemstra, J., Osama, M., Wijs, A.: Towards End-to-End GPU Acceleration of PCTL Model Checking. Springer, Cham (2025)

[2] Heemstra, J., Wijs, A.: GPUexplore-prob: Markov Chain State Space Construction and Verification with GPUs. In: TACAS (2025)

[3] Osama, M., Wijs, A.J.: Hitching a Ride to a Lasso: Massively Parallel On-The-Fly LTL Model Checking. In: TACAS. Springer (2024)

[4] Wijs, A.J., Osama, M.: GPUexplore 3.0: GPU Accelerated State Space Exploration for Concurrent Systems with Data. In: SPIN. Springer (2023)

https://github.com/egolf-cs/tacas25

Accelerating Protocol Synthesis and Detecting Unrealizability with Interpretation Reduction

Derek Egolf, Stavros Tripakis
Northeastern University, Boston
egolf.d@northeastern.edu

Key Contributions

- Synthesize symbolic distributed protocols represented in TLA$^+$ [Lamport].
- Improve state of art in TLA$^+$ synthesis (100x).
- Synthesize a lock protocol "from scratch."
- Halt when no solution: *unrealizability*.
- New search space reduction technique: *Interpretation Reduction*.
- Improved counterexample generalization for pruning.

Sketching [Solar-Lezama]

Given an incomplete sketch with "holes," find a correct completion.

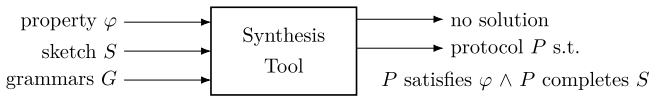
Example sketch:

$Send(src, dst) :=$
$\quad \wedge\ ???_1$
$\quad \wedge\ message' = ???_2$
$\quad \wedge\ has\_lock' = ???_3$
$Receive(src, dst) :=$
$\quad \wedge\ ???_4$
$\quad \wedge\ message' = ???_5$
$\quad \wedge\ has\_lock' = ???_6$

Example completion

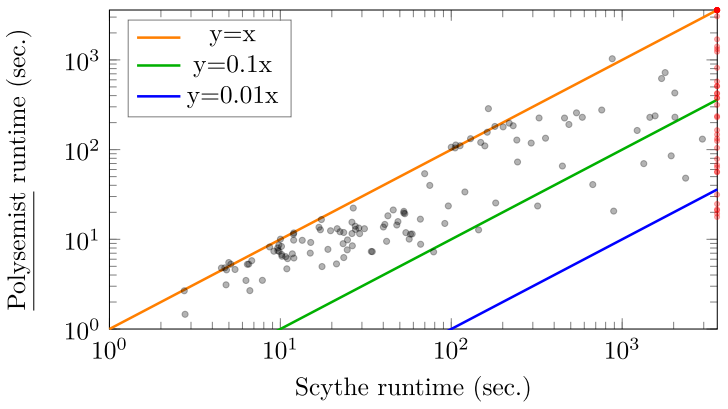
$Send(src, dst) :=$
$\quad \wedge\ has\_lock[src]$
$\quad \wedge\ message' = message \cup \{(src, dst)\}$
$\quad \wedge\ has\_lock' = has\_lock[src \leftarrow false]$
$Receive(src, dst) :=$
$\quad \wedge\ (src, dst) \in message$
$\quad \wedge\ message' = message \setminus \{(src, dst)\}$
$\quad \wedge\ has\_lock' = has\_lock[dst \leftarrow true]$

Problem Statement



Experimental Results

Scythe vs Polysemist Runtime (Realizable Experiments)



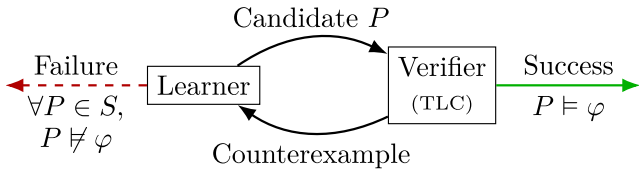
$n = 171$; worse: 11; scythe TO: 47; poly TO 15

Unrealizable Experiments ($n = 123$)

- **Scythe** (old): TO = 107 / HALT = 16
- **Polysemist** (new): TO = 43 / HALT = 80
  - Usually halted in < 60 seconds
  - Did not TO unless Scythe did

Counterexample-Guided Inductive Synthesis (CEGIS)

Our approach uses standard CEGIS technique [Solar-Lezama].



Key Technical Ideas for Learner

- Naive learner: ignore cex, enumerate all protocols
  many expressions; model checking expensive
- Pruning constraints: generalize counterexamples
  discard protocols before model checking
- Equivalence reduction: do not use equiv. sub-expressions
  avoid enumerating protocols in the first place
- Interpretation reduction: coarse, dynamic equivalence relation

Counterexample Generalization

safety cex: $[x \mapsto a] \xrightarrow{A_1} [x \mapsto b]$
Sketch: $(A_1 := ???_1 \wedge\ x' = ???_2)$
Bad Completion: $(A_1 := \text{true} \wedge\ x' = b)$
Bad Completion: $(A_1 := x = a \wedge\ x' = b)$
Good Completion: $(A_1 := x = a \wedge\ x' = a)$
Good Completion: $(A_1 := x \neq a \wedge\ x' = b)$

Pruning Constraint:

$$\pi_{cex} := ???_1([x \mapsto a]) \neq true \vee ???_2([x \mapsto a]) \neq b$$

- In general, many completions violate π_{cex}.
- Checking $P \models \pi_{cex}$ is much cheaper than model checking.
- We generalize deadlock, safety, and liveness violations.
- Prior work [FMCAD'24] uses less exact pruning constraints for deadlock/liveness.

Interpretation Reduction

Absolute Equivalence, e.g: $x + y \equiv y + x$
Interpretation Equivalence, e.g:

$$x + y \equiv_{\mathcal{A}} x + x, \text{ where } \mathcal{A} = \{[x \mapsto 0, y \mapsto 0]\}$$

- $\mathcal{A}$ comes from pruning constraints. E.g.,
  $\Pi = \{\ ???_1([x \mapsto 0, y \mapsto 1]) \neq 1\ \} \rightarrow \mathcal{A} = \{[x \mapsto 0, y \mapsto 1]\}$
- Suppose we've enumerated y; enumerate $x + y$? No: $y \equiv_{\mathcal{A}} x + y$.
- Avoids enumerating all "super-expressions" of $x + y$, e.g., $x + x + y$.
- Coarse eq. relation makes detecting unrealizability faster.
- [FMCAD'24] uses absolute equivalence for reduction.

Theorem: If $e_1 \equiv_{\mathcal{A}} e_2$ and e_1 enumerated, skipping e_2 does not compromise completeness

References

Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley (2002).
Solar-Lezama, A.: Program Sketching. Int. J. Softw. Tools Technol. Transf. (2013).
Egolf, D., Tripakis, S.: Efficient Synthesis of Distributed Protocols by Sketching. FMCAD (2024).

ETH zürich

Department of Computer Science
Information Security Group

Pushing the Limit: Verified Performance-Optimal Causally-Consistent Database Transactions


Shabnam Ghasemirad


Dr. Christoph Sprenger


Dr. Si Liu


Luca Multazzu

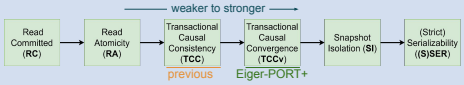

Prof. Dr. David Basin

shabnam.ghasemirad@inf.ethz.ch

Project repository:

1 Motivation

- Distributed databases (key-value stores)
- **Isolation** of concurrent transactions, realized by concurrency control protocols.
- Spectrum of isolation levels:



- Trade-off: isolation vs performance
- TCC previously conjectured to be the strongest achievable isolation level for performance-optimal reads in the presence of transactional writes.
- We **refute** the **conjecture** and push the limit to **TCCv** with our novel protocol **Eiger-PORT+.**
- Concurrency control protocols are highly complex and prone to design errors and isolation bugs. → Deductive **verification**

Contributions

❖ **Eiger-PORT+**
- ➢ *Stronger isolation* guarantee
- ➢ *Superior performance*

❖ **Protocol verification** in Isabelle/HOL
- ➢ Refinement
- ➢ Reduction

2 Abstract Model: Isolation Level

We specify isolation as an abstract model parameterized by an Isolation Level (IL).

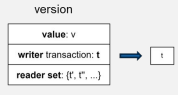
The abstract model's event system:

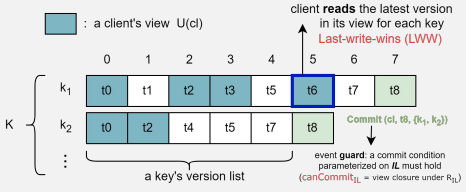
➔ States: (K, U)
- **K**: centralized multi-versioned key-value store
- **U**: client views, capturing the distributed aspect

➔ Events:

1) Atomic Commit
$(K, U) \xrightarrow{\text{commit}}_{IL} (K', U')$

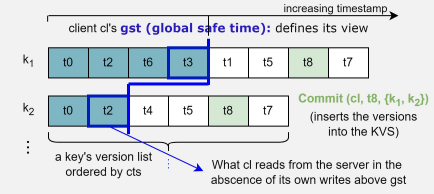
2) View Extension
$(K, U) \xrightarrow{\text{view ext}}_{IL} (K, U')$



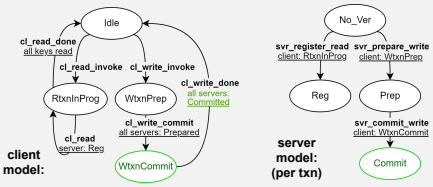


3 Concrete Model: Eiger-PORT+

Eiger-PORT+ protocol:
Timestamp-based (uses Lamport clocks)



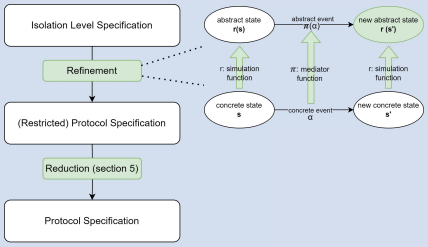
- read-only transactions
  - ➔ own write above gst
  - ➔ or last write below gst
- write-only transactions: two-phase commit



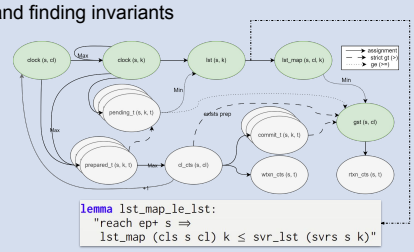
4 Correctness Proof & Invariants

Proof guarantee:
refinement (reach (protocol)) ⊆ reach (IL)

- refinement mapping:
  r : K and U reconstructed as shown above
  π: client_write_commit and client_read_done mapped to *Atomic commit*
- proof obligations:
  $canCommit_{IL}$: needs invariants (below)
  LWW: needs reduction



- The relation of different timestamps in the model and finding invariants



5 Inverted Commits & Reduction

Inverted commits: pairs of client commits in protocol executions not ordered by **commit timestamps**.

Can occur for causally independent concurrent transactions.

Problem: Inverted commits would require ***inserting*** rather than ***appending*** a transaction's version to the version list.

Can not be simulated by the abstract model.

Hence, refinement alone is not enough for verifying the protocol.

Solution:
- We introduce a restricted protocol model that doesn't produce inverted commits.

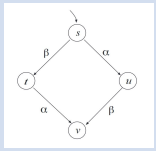
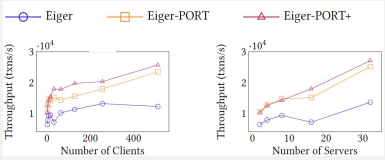


Image source: C. Baier, J. Katoen. 2008. Principles of model checking, p. 595, Figure 8.1.

- We use **reduction** to transform any protocol execution into one of the restricted model such that:
  reach (protocol) = reach (restricted protocol).
- This is achieved by commuting independent concurrent events to eliminate inverted commits. (see arrows on the execution above)

6 Conclusions and Discussion

- Our Eiger-PORT+ protocol provides TCCv, thus refuting an open conjecture.
- Eiger-PORT+ outperforms state-of-the-art



- Refinement is not always enough
- We deductively verify that Eiger-PORT+ satisfies TCCv, using a combination of refinement and reduction.

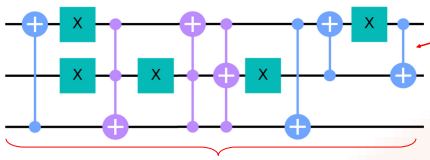
SliQSim: A Quantum Circuit Simulator and Solver for Probability and Statistics Queries

Tian-Fu Chen and Jie-Hong R. Jiang, National Taiwan University
{d11k42001, jhjiang} @ntu.edu.tw

1. Quantum system of n qubits

- Quantum state: complex vector $|\psi\rangle$ with size 2^n

Quantum state evolves: apply U on $|\psi\rangle$



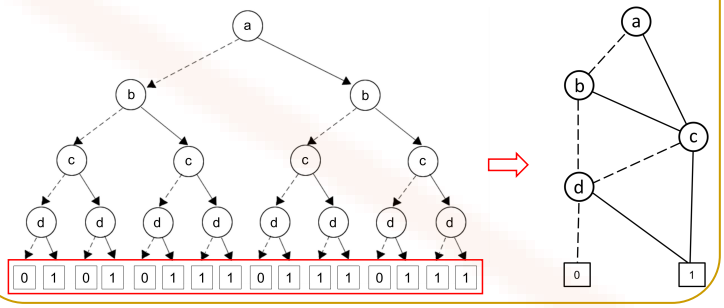
Quantum gates: complex matrix U with size $2^n \times 2^n$

A line is a qubit

Quantum circuit: a sequence of quantum gates

2. Binary Decision Diagram (BDD)

- A compact representation for bit vector/matrices
- Boolean operations can be done efficiently
- 2^n-dimensional vector → n Boolean variable
- Example: $[0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,1]$



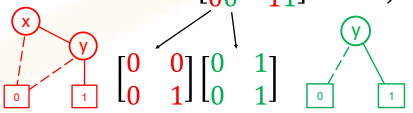
3. Algebraic Representation & Bit-slicing

- Transform quantum state/gate into several bit vectors/matrices
  - $\alpha = \frac{1}{\sqrt{2^k}}(a\omega^3 + b\omega^2 + c\omega + d)$, $a, b, c, d, k \in \mathbb{Z}$ and $\omega = e^{i\pi/4}$
- a, b, c, d are decomposed into binary numbers and further represented by BDDs

Compact, Efficient, Exact (no precision loss), and Easy for matrix/vector entries manipulations!

Example:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & e^{i\pi/4}\\ 1 & -e^{i\pi/4}\end{bmatrix} = \frac{1}{\sqrt{2}}\left(\omega^3[\dots] + \omega^2[\dots] + \omega\begin{bmatrix}00 & 01\\ 00 & 11\end{bmatrix} + [\dots]\right)$$



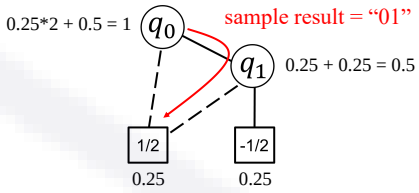
4. Simulation & Probability and Statistics Queries

- Start from the initial quantum state

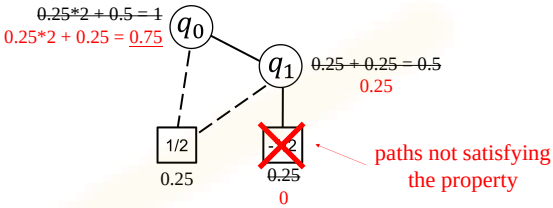
$$F = \overline{x_0}\,\overline{x_1}\dots\overline{x_{n-1}} \qquad v = \begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix}\begin{matrix}\overline{x_0}\,\overline{x_1}\dots\overline{x_{n-1}}\\ \overline{x_0}\,\overline{x_1}\dots x_{n-1}\\ \\ x_0\,x_1\dots x_{n-1}\end{matrix}$$

and apply gates sequentially

- Conduct measurement for the final state
  1. Combine all BDDs to form a multi-terminal BDD (e.g., $|\psi_m\rangle = [1/2, 1/2, 1/2, -1/2]^T$)
  2. Decide the probability of each node by dynamic programming
  3. Sample according to the probability



- Check the probability of certain property
  1. Parse the condition as a Boolean formula (e.g., $F_M = \overline{q_0} \vee \overline{q_1}$)
  2. Update $F \leftarrow F \wedge F_M$ for all BDDs → *equivalently mask out states not satisfying the condition*
  3. Decide the probability of the root node by dynamic programming
  4. Probability obtained!



5. Supported Statistics

- Probability of satisfying a Boolean formula
- Probability of Hamming weight being in a range
- Expectation value of a Pauli-string
- Weighted sum of multiple properties
- Whether the value of a property is in a range
- Probability amplitude of a basis state
- Exact spectrum of the probability distribution
- ...

Previous Publications & Open Sourced Tools

1. Y.-H. Tsai, J.-H. R. Jiang, and C.-S. Jhang, "Bit-slicing the Hilbert space: Scaling up accurate quantum circuit simulation," in *Proc. DAC*, 2021, pp. 439–444.
2. C.-Y. Wei, Y.-H. Tsai, C. -S. Jhang, and J.-H. R. Jiang, "Accurate BDD-based Unitary Operator Manipulation for Scalable and Robust Quantum Circuit Verification," in *Proc. DAC*, 2022, pp. 523-528.
3. T.-F. Chen, J.-H. R. Jiang, and M.-H. Hsieh, "Partial Equivalence Checking of Quantum Circuits," in *Proc. QCE*, 2022.

a) Simulator – SliQSim https://github.com/NTU-ALComLab/SliQSim b) Equivalence Checker – SliQEC https://github.com/NTU-ALComLab/SliQEC

Synthesis with Guided Environments

Orna Kupferman and Ofer Leshkowitz
Hebrew University of Jerusalem, Jerusalem, Israel

Synthesis of Reactive Systems

Reactive System –

I = Finite set of **input** signals.
O = Finite set of **output** signals

- Environment generates assignments to input signals $i_1, i_2, i_3, \ldots \in 2^I$
- System generates assignments to output signals $o_1, o_2, o_3, \ldots \in 2^O$
- Outputs generated online: o_k is determined by $i_1, i_2, \ldots, i_k$.

Synthesis –

φ = LTL formula

- Given a specification φ of "good computations", automatically construct a reactive system that realizes φ.

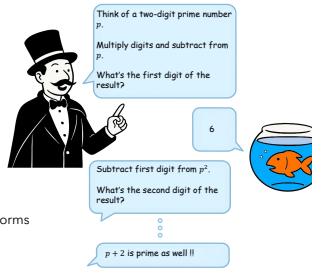
Dealing with partial visibility

- The assignment to $H \subseteq I$ is hidden from the system.
- System should satisfy φ for all assignments to H.
- Simple specification become unrealizable.

Example:
$\varphi = i \leftrightarrow o$
$I = \{i\}, O = \{o\}$ and $H = \{i\}$.

What's going on?

- Magician –
  1. Gains partial information.
  2. Guides fish step-by-step.
- Fish –
  1. Picks p adversarially.
  2. Follows instructions – Remembers values & Performs calculations.

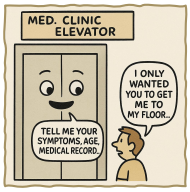


Dealing with partial visibility

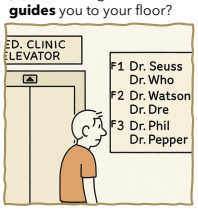
(Back to Reactive Systems…)

When visiting your doctor do you –

(A) Share all medical info with the elevator?



(B) Follow a sign that **guides** you to your floor?



Transducer with a Guided Environment

A TGE sends ***"programs"*** used by Environment to assign guided outputs.

Example: **Delay output:** $G(i \leftrightarrow X o)$
Specification: $\varphi = G(i \leftrightarrow o)$.
Hidden: $H = I = \{i\}$.
Guided: $G = O = \{o\}$.

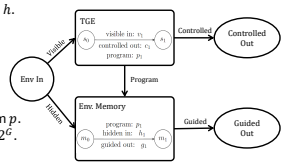
"copy previous(i) to o"
program = "copy i to o"

Environment needs ***memory***.

Transducer with a Guided Environment

A round of interaction with a TGE:
($I = V \cup H, O = C \cup G$)

- Env generates:
  - Visible input v & Hidden input h.
- TGE reads visible input v:
  - Updates state to s'.
  - Generates – Controlled output c & Program p. $c \in 2^C$ and p: $M \times 2^H \to M \times 2^G$.
- Env evaluates $p(m, h) = (m', g)$.
  - Updates memory to m'.
  - Sets guided out to g.



Memory-State Trade-offs

Env' does all the work

TGE
Realizing φ
no visibility ($V = \emptyset$)
no control ($C = \emptyset$)
1 state
m memories

↔

Standard transducer
(Full visibility and control)
Realizing φ
m states

TGE
Realizing φ
Partial Visibility\Control
n states
m memories

Product :TGE×Env →

Standard transducer
(Full visibility and control)
Realizing φ
$n \cdot m$ states

Memory Monotonicity

More visibility \ guidance ↘ Less memory.

1. Increasing visibility (viewing also $h \in H$) can only reduce needed memory.
   Assignment to h can be hard-coded into programs
2. Increasing guidance (guiding also $c \in C$) can only reduce needed memory.
   Assignment to c can be hard-coded into programs

If the system sees more or guides more, the environment can remember less.

Solving SGE – Synthesis with Guided Environments

The problem: Synthesize a TGE that realizes φ with H hidden, G guided and at most k environment memories or determine that there is no such TGE.

Theorem: (1) A tree $\mathcal{T}$ is accepted by $A^{\varphi}_{M,H,G}$ iff it realizes φ.
(2) $A^{\varphi}_{M,H,G}$ is of size $|M| \cdot \exp(|\varphi|)$.

Reduce synthesis to the **non-emptiness** of $A^{\varphi}_{M,H,G}$:

1. Search for a finite witness $\mathcal{T}$ for non-emptiness of $A^{\varphi}_{M,H,G}$ in EXPTIME.
2. If empty – Not realizable.
3. Otherwise – $\mathcal{T}$ describes a TGE realizing φ with H hidden and G guided.

Corollary: SGE is 2EXPTIME in $|\varphi|$ and EXPTIME in $|M|$.

Non-Zero-Sum Games with Multiple Weighted Objectives

PRESENTER:
Yoav Feinstein

We introduce and study non-zero-sum multi-player games with weighted multiple objectives.
In these games, the objective of each player consists of a set α of underlying objectives and a weight function w: $2^{\alpha} \to \mathbb{Z}$ that maps each subset X of α to the utility of the player when exactly all the objectives in X are satisfied.
The weight functions lift the setting of non-zero-sum multi-player games to the general quantitative case, allowing a rich reference to the underlying objectives.

Settings:
- Several interacting agents (robots, users)
- each having an objective
- each objective is composed of weighted sub-objectives

k-player weighted multiple objectives

Set of objectives $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ $\alpha_i \subseteq V$ (α_i is a Büchi objective)
Set of weight functions $\{w_1, w_2, \ldots, w_k\}$ $w_i: 2^{\alpha} \to \mathbb{Z}$ (one for each player in the game)

For a play ρ:
- $sat(\rho,\alpha) \subseteq \alpha$ The set of objectives that ρ satisfies
- For every $i \in [k]$, $value(\rho,\alpha,w_i) = w_i(sat(\rho,\alpha))$
The utility of Player i

Stable profiles in weighted k-player games:

Zero-sum games: No overlap among objectives, find the winning players
Non-zero-sum games: Overlap among objectives, find stable profiles

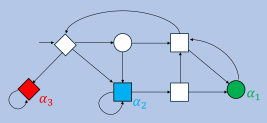
A predicate describes desired utilities of the play.
For example: $u_1 > 2 \wedge (u_2 < 3 \vee u_3 \geq 1)$

For a set $S \subseteq [k]$ of system players, and a predicate P.
The Partially-Fixed NE with Desired Utilities (DNE) problem is to return an S-fixed NE π that satisfies P.

There is a warehouse, P1 and P2 are robots who wants to access the warehouse to move inventory. P3 is a guard making sure all is in order.

α_1 P1 gets her goal
α_2 P2 gets her goal
α_3 The warehouse is closed

P1: Robot 1 (system)
P2: Robot 2
P3: Warehouse Guard



Goal: NE in which P1 achieves her objective (get a utility higher than zero)

w_1	w_2	w_3
$w_1(\alpha_1) = 3$	$w_2(\alpha_1) = 0$	$w_3(\alpha_1) = 2$
$w_1(\alpha_2) = 0$	$w_2(\alpha_2) = 3$	$w_3(\alpha_2) = 2$
$w_1(\alpha_3) = 0$	$w_2(\alpha_3) = 0$	$w_3(\alpha_3) = 3$

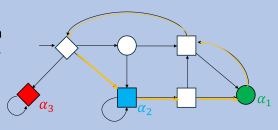
Additive weight function:
$w_i: \alpha \to \mathbb{N}$, and $w_i(S) = \sum_{\alpha_i \in S} w_i(S)$

P3 can move to red which gives her \$3

Thus, every NE which satisfies α_1 must also give at least \$3 to P3 (else she would deviate)

Our profile needs to satisfy both α_1 and α_2 so that P3 gets \$4 and will not benefit from deviating to satisfy α_3.

In this profile, P1 and P2 each gets \$3, and P3 gets \$4



Goal: NE in which P1 achieves her objective (get a utility higher than zero)

Take a picture to download the full paper

Our contribution:

Limitation on the weights	Zero-sum games	Non-zero-sum DNE
No limitation	PSPACE-complete [KS 24]	PSPACE-complete (already for k=2)
Non-decreasing	co-NP-complete [KS 24]	NP-complete (already for k=2)
Additive	PSPACE-complete	PSPACE-complete (already for k=2)
Almost-positive additive	co-NP-complete	NP-complete (already for k=2)

Applications of k-player weighted multiple objectives

We study the extension of the game by payments, with which players can incentivize each other to follow strategies that are beneficial for the paying player. We show how such payments can be used in order to repair systems.

Games with Payments

$\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ $\alpha_i \subseteq V$
Set of Buchi objectives, one for each player

$\{R_1, R_2, \ldots, R_k\}$ $R_i \in \mathbb{N}$
R_i the reward Player i receives when objective α_i is satisfied

A payment function $p: [k] \times [k] \to \mathbb{N}$
$p(i,j)$ the amount Player i commits to pay Player j when α_i is satisfied

For a play ρ, $W(\rho)$ is the set of objectives satisfied by ρ

The utility of Player i is:

The reward for Player i
The payment **to** other players
The payment **from** other players

$$utility_i(\rho) = \begin{cases} R_i - \sum_{j \in [k]} p(i,j) + \sum_{j \in W(\rho)} p(j,i) & i \in W(\rho) \\ \sum_{j \in W(\rho)} p(j,i) & i \notin W(\rho) \end{cases}$$

The payment **from** other players

The monetary-based system-repair problem is given a set of system players S, and a predicate P to decide if there exists a payment function $p: S \times [k] \to \mathbb{N}$ and a profile π such that π is a solution for the DNE problem over the payment game with the payment function p.

Yoav Feinstein, Orna Kupferman, Noam Shenwald

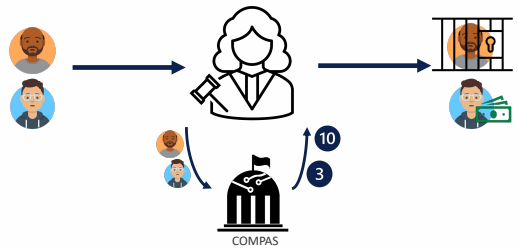
yoav.feinstein@mail.huji.ac.il,
orna@cs.huji.ac.il,
noam.shenwald@mail.huji.ac.il

THE HEBREW UNIVERSITY OF JERUSALEM

STREAM-BASED MONITORING OF ALGORITHMIC FAIRNESS

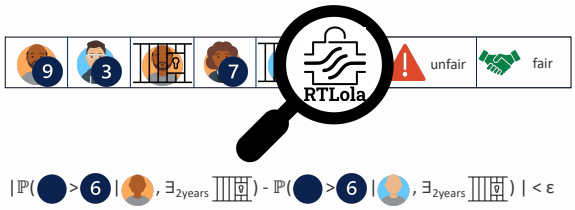
Jan Baumeister, Bernd Finkbeiner, Frederik Scheerer, Julian Siber, Tobias Wagenpfeil

UNFAIR AI SYSTEMS



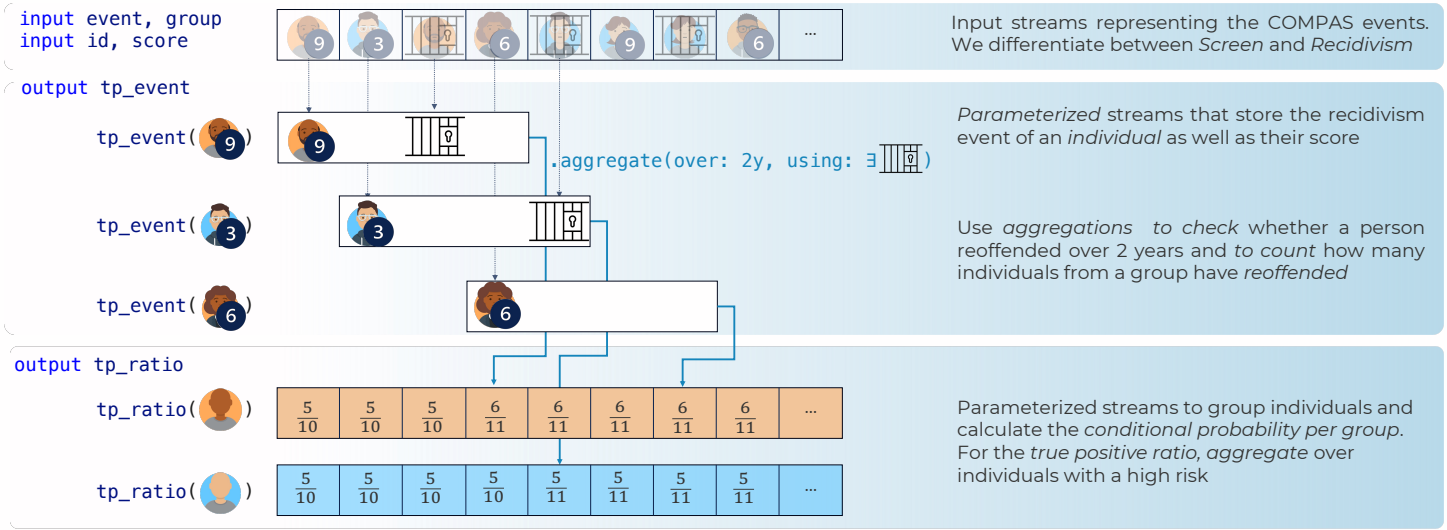
*AI is used for critical decisions. COMPAS is an AI tool to predict recidivism. ProPublica revealed that **COMPAS scores are biased** against Afro-Americans.*

MONITORING FAIRNESS PROPERTIES



*We propose to use the **RTLola monitoring framework** for **detecting unfairness** during the deployment of black-box AI systems through estimating conditional probabilities.*

SPECIFYING EQUALIZED ODDS WITH RTLOLA



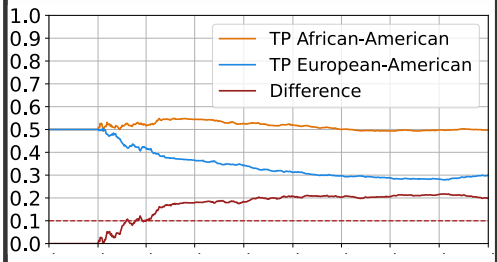
Input streams representing the COMPAS events. We differentiate between *Screen* and *Recidivism*

Parameterized streams that store the recidivism event of an *individual* as well as their score

Use *aggregations* to *check* whether a person reoffended over 2 years and *to count* how many individuals from a group have *reoffended*

Parameterized streams to group individuals and calculate the *conditional probability per group*. For the *true positive ratio*, *aggregate* over individuals with a high risk

MONITORING COMPAS

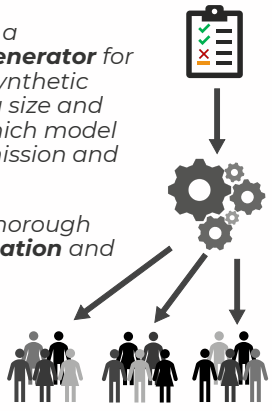


*The monitor detects that the COMPAS system violates equalized odds **around 3 months after the first outcomes** are observed.*

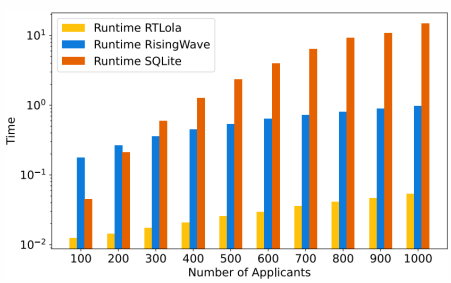
SYNTHETIC DATA

*We contribute a **benchmark generator** for constructing synthetic data of scaling size and complexity, which model university admission and job hiring.*

*This allows a thorough **runtime evaluation** and comparison.*



RUNTIME COMPARISON



*Monitoring with **RTLola is faster** than implementations based on SQLite and the streaming database RisingWave on the synthetic data.*

CISPA HELMHOLTZ-ZENTRUM FÜR INFORMATIONSSICHERHEIT
RTLola www.rtlola.org
31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems **(TACAS 2025)**
Funded by the European Union
erc European Research Council
CPEC CENTER FOR PERSPICUOUS COMPUTING

CERTIFYING PARETO OPTIMALITY IN MULTI-OBJECTIVE MAXSAT

Christoph Jabs[1], Jeremias Berg[1], Bart Bogaerts[2], Matti Järvisalo[1]
[1]University of Helsinki, [2]KU Leuven & Vrije Universiteit Brussel

@ TACAS'25
Presentation Wednesday 11:30

MULTI-OBJECTIVE OPTIMIZATION

- Many real-world problems have multiple conflicting objectives.
- Aim: **Pareto-optimal** solutions
- Linear combination of objectives not sufficient

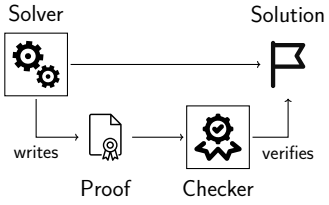
MAXIMUM SATISFIABILITY

- Constraints: propositional formula, objective(s): linear function over variables
- Efficient for real-world optimization problems

CONTRIBUTIONS

- VeriPB proofs for multi-objective MaxSAT
- No modifications to VeriPB proof system
- Open-source implementation for three algorithms
- Showing low overhead of proof logging

PROOF LOGGING



- Write log of reasoning steps while solving
- Simple or formally verified checker
- Checking proof verifies correctness of result

VERIPB [BOGAERTS ET AL. JAIR'23]

- Pseudo-boolean (cutting planes) proof system
- Derive constraints by linear combination
- Redundant constraints (RAT generalization [Järvisalo et al. IJCAR'12])
- Native single-objective support
- Preorder (originally for symmetry breaking), witness for redundant constraints must be smaller in preorder

MO-PROOFS

Certificate that at least one representative solution for each non-dominated point was discovered.

PROOF SETUP

Unmodified VeriPB proof system

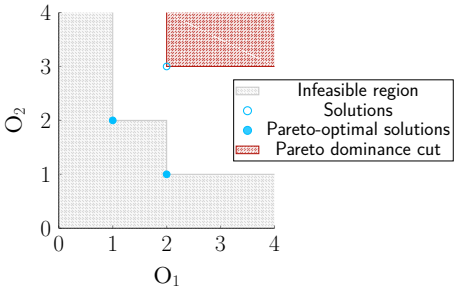
1. Encode Pareto dominance as preorder in proof
2. Now Pareto-optimal solutions can only be *explicitly* excluded in the proof
3. It the proof terminates as unsatisfiable, all Pareto-optimal solutions must appear in the proof

Syntactic restrictions

- First step in proof must load the Pareto order
- Order must never be changed

PARETO DOMINANCE CUTS

- Blocking all solutions that are worse than the current one
- Similar to *solution-improving constraint*



Certifying a Pareto dominance cut for solution α

1. Map each weakly dominated solution to α; Redundant with α as witness
2. Exclude α itself
3. Derive cut from steps 1 and 2

MO-MAXSAT ALGORITHMS

P-MINIMAL [SOH ET AL. CP'17]

1. Find a solution SAT solver
2. Introduce PD cut
3. Find a dominating solution
   - If yes, goto 2
   - If no, previous is optimal, goto 1

Proof needs to cover

- SAT solver reasoning
- CNF Objective encodings
- PD cuts (see above)

LOWER-BOUNDING [CORTES ET AL. TACAS'23]

- Execute *P*-Minimal within upper-bounds on objectives
- Once done, loosen upper-bounds

Upper-bounds on objectives can be ignored in proof
→ same as *P*-Minimal

BIOPTSAT [JABS ET AL. JAIR'24]

- Bi-objective
- Optimize first one objective, then the other
- When minimizing the second objective, PD cuts can be strengthened to unit clauses

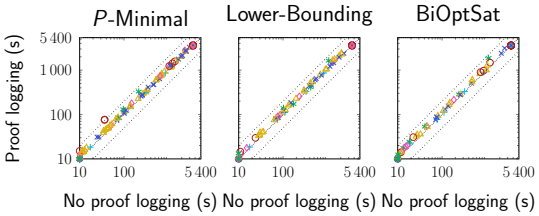
In proof

- Derive lower-bound on first objective
- Certify PD cut
- Strengthen PD cut based on known lower-bound

EXPERIMENTAL EVALUATION

RESULTS

PER-INSTANCE PROOF LOGGING OVERHEAD



AVERAGE OVERHEAD AND CHECKING TIME

Algorithm	w/logging / w/o logging	checking / w/logging
P-Minimal	1.233	47.52
Lower-Bounding	1.220	21.81
BiOptSat	1.247	29.70

REFERENCES

Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström: *Certified Dominance and Symmetry Breaking for Combinatorial Optimisation*, JAIR, 2023.

João Cortes, Inês Lynce, and Vasco M. Manquinho: *New Core-Guided and Hitting Set Algorithms for Multi-Objective Combinatorial Optimization*, TACAS, 2023.

Christoph Jabs, Jeremias Berg, Andreas Niskanen, and Matti Järvisalo: *From Single-Objective to Bi-Objective Maximum Satisfiability Solving*, JAIR, 2024.

Matti Järvisalo, Marijn Heule, and Armin Biere: *Inprocessing Rules*, IJCAR, 2012.

Takehide Soh, Mutsunori Banbara, Naoyuki Tamura, and Daniel Le Berre: *Solving Multiobjective Discrete Optimization Problems with Propositional Minimal Model Generation*, CP, 2017.

christophjabs.info/tacas25

HIIT

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

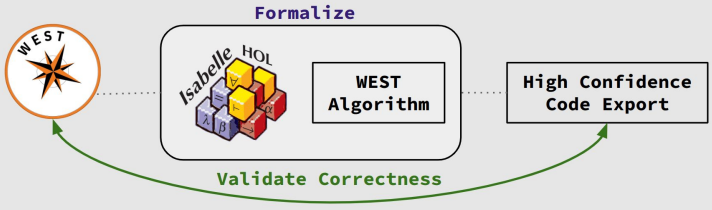
MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
MATEMATISK-NATURVETENSKAPLIGA FAKULTETEN
FACULTY OF SCIENCE

Formally Verifying a Transformation from MLTL Formulas to Regular Expressions

Zili Wang*, Katherine Kosaian+, Kristin Yvonne Rozier*
*Iowa State University, +University of Iowa
[ziliw1, kyrozier]@iastate.edu, katherine-kosaian@uiowa.edu

Overview

The results of formal verification are only as trustworthy as their input specifications. As such, the WEST tool was created to facilitate writing specifications in Mission-time Linear Temporal Logic (MLTL) by visualizing MLTL formulas as regular expressions (regexes). To certify the correctness of the WEST tool, we formally verify the correctness of the WEST algorithm in the interactive theorem prover Isabelle/HOL. We then generate a code export from our verified development in Haskell and use this to experimentally validate the existing WEST tool.



Mission-time LTL (MLTL)

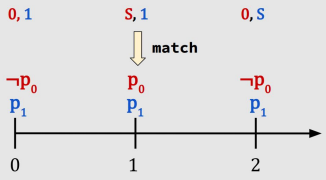
MLTL is a finite version of Linear Temporal Logic with discrete integer time bounds on the temporal operators. Here are some examples of traces (assignments of variables through time) that satisfy various MLTL formulas:

Operator	Syntax	0	1	2	3	4	5	6	7
Globally	$G_{[2,5]}p$			p	p	p	p		
in the Future	$F_{[0,4]}p$					p			
Until	$p\,U_{[1,6]}q$	p	p	p	p	q			
Release	$p\,R_{[2,7]}q$			q	q	q	q	q	p,q

MLTL → Regular Expressions

Regular expressions (regexes) describe which variables must be true or false at certain timesteps. We say that a regex **matches** a trace. For example:

- Fix a variable ordering $\{p_0, p_1\}$
- 1 = true, 0 = false, S = any value



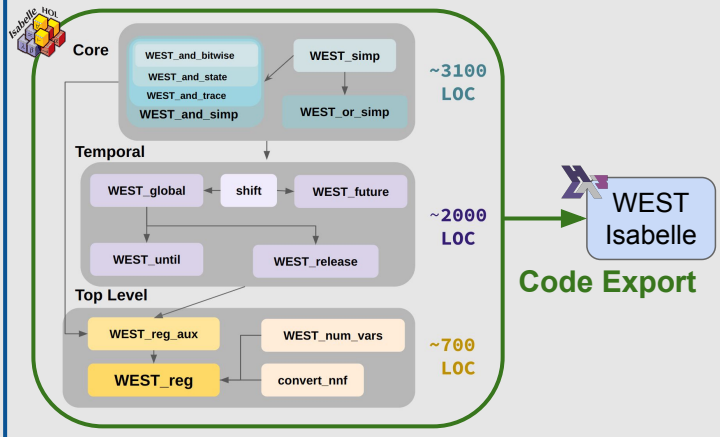
The WEST algorithm:
Transforms MLTL → Regex
Given an MLTL formula, computes the regular expressions which captures *all* the satisfying traces of the formula.
For example on $(p_0 U_{[1,6]} p_1)$, WEST produces the following:

```
ss,s1,ss,ss,ss,ss,ss
ss,1s,s1,ss,ss,ss,ss
ss,1s,1s,s1,ss,ss,ss
ss,1s,1s,1s,s1,ss,ss
ss,1s,1s,1s,1s,s1,ss
ss,1s,1s,1s,1s,1s,s1
```

Check out all of our work here →
west.temporallogic.org

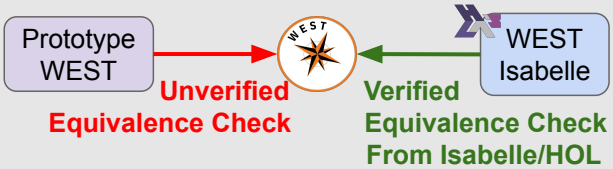
Formalization Contributions

- Available on the Archive of Formal Proofs under the entry "Mission-time Linear Temporal Logic to Regular Expressions"
- We formalize each operation of the WEST algorithm in and prove overall correctness of the algorithm.
- We use Isabelle/HOL's code generator to obtain an implementation of the WEST algorithm in Haskell



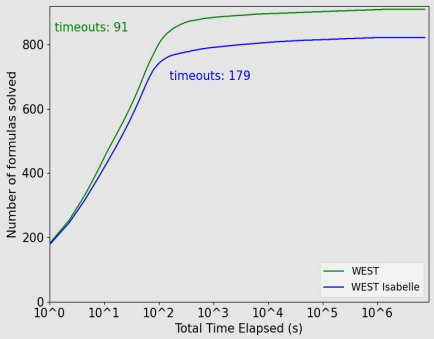

```
theorem WEST_correct:
  fixes φ::"(nat) mltl" and π::"trace"
  assumes "intervals_welldef φ"
  assumes π_long_enough: "length π ≥ complen_mltl φ"
  shows  "match π (WEST_reg φ) ⟷ semantics_mltl π φ"
```

Tool Validation



Test suite: 1662 formulas that targets every combination of operator nesting
Verified check: holds on **1658** formulas, times out on **4**
Example - $(\neg p_3\ U[0,2]\ \neg p_2)\ R[0,2]\ (p_0\ U[0,2]\ \neg p_1)$
Unverified check: Succeeded on all **1662**

Speed Comparison



Speed was not the focus of this work, but surprisingly fast!
1000 random formulas
- Operator nesting depth = 10
- Maximum interval bounds = 5
- Number of variables = 10

31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), May 5-8 2025, Hamilton, Canada

Acknowledgments: This work is supported by the NSF-GRFP 2024364991, NSF CAREER Award CNS-1552934, and NSF CCRI-2016592. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors(s) and do not necessarily reflect the views of the National Science Foundation.

INCREMENTAL SAT-BASED ENUMERATION OF SOLUTIONS TO THE YANG-BAXTER EQUATION

Daimy Van Caudenberg[1], Bart Bogaerts[1,2],
Leandro Vendramin[3]
[1]Declarative Languages and Artificial Intelligence section, KULeuven, Leuven, Belgium
[2]Department of Computer Science, Vrije Universiteit Brussel, Brussels, Belgium
[3]Department of Mathematics and Data Science, Vrije Universiteit Brussel, Brussels, Belgium

1. INTRO

THE YANG-BAXTER EQUATION

- Yang-Baxter equation (YBE) originally introduced in context of statistical [Yan67] and quantum mechanics [Bax72].
- Applications in knot theory, quantum computing...
- We focus on a subset of solutions.
  - Finite, involutive, non-degenerate, combinatorial solutions.
  - These solutions have relations to group and ring theory.
  - They can be studied using an equivalent mathematical structure: non-degenerate cycle sets.

ENUMERATION

- Enumerating all (non-isomorphic) solutions to the YBE is an open problem!
- A database of solutions could inspire:
  - experimentation,
  - examples of algebraic structures to study,
  - and counterexamples to previous conjectures...
- [AMV22] enumerated solutions up to size 10 using a constraint programming approach with static symmetry breaking.

2. SAT MODULO SYMMETRIES

SAT

- Boolean Satisfiability (SAT) problem decides if there exists a satisfying assignment for a given propositional formula.
- SAT solvers can:
  - decide if a formula is satisfiable,
  - enumerate satisfying assignments (if they exist)...
- We encode the mathematical problem as a propositional formula,
  - hence, a satisfying assignment corresponds to a solution.

SAT MODULO SYMMETRIES

- SAT Modulo Symmetries (SMS) was originally introduced in [KS21,KS24].
- It forces a SAT solver to generate only non-isomorphic solutions during the search:
  - Obtain a partial interpretation from the SAT solver.
  - Perform a Minimality Check:
    - verify if the assignment can be extended to a complete assignment that is lexicographically minimal.
    - if this fails, the solver aborts the current branch of the search tree by learning a new clause.

3. MINIMALITY CHECK FOR YBE

PARTIAL SOLUTIONS

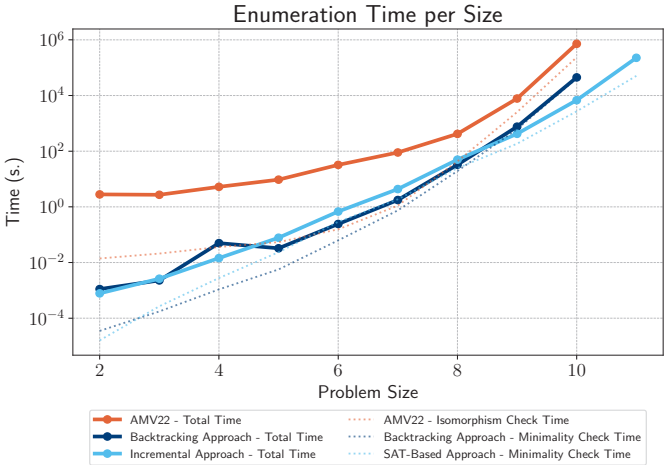
- SMS introduces breaking constraints during the search, so, only a partial solution is known...
- This is an incomplete solution that can still be extended to a well-defined solution.
- A partial solution P is lexicographically minimal iff one of its extensions $C \in \mathscr{X}(P)$ is lexicographically minimal,
  - where $\mathscr{X}(P)$ is the set of all complete solutions that P can be extended to.

MINIMALITY CHECK FOR YBE

- Does there exist a permutation π s.t. the image of the current (partial) solution P under π is strictly smaller than P?
- If so, use π to exclude P and its extensions!
- Find breaking permutations by either:
  - performing a backtracking search over the space of possible permutations,
  - or by encoding as a SAT problem what it means for a permutation to be useful.

4. RESULTS & CONCLUSION

RESULTS



FUTURE WORK

- Certifying the results.
  - We obtain the same results as [AMV22], but that only means that we are either both correct or both wrong...
- Enumerating related structures:
  - racks, used to enumerate skew cycle sets.
  - skew Cycle Sets, correspond to non-degenerate set-theoretic solutions.
  - biquandles, applications in knot theory.

[AMV22] Akgün, Ö., Mereb and M., Vendramin, L. (2022). Enumeration of Set-Theoretic Solutions to the Yang-Baxter Equation. In Mathematics of Computation. 91(335).
[YAN67] Yang, C. N. (1967). Some Exact Results for the Many-Body Problem in one Dimension with Repulsive Delta-Function Interaction. In Phys. Rev. Lett. 19 (23).
[BAX72] Baxter, R. J. (1972). Partition function of the Eight-Vertex lattice model. In Annals of Physics 70(1).
[KS21] Kirchweger, M. and Szeider, S. (2021). SAT Modulo Symmetries for Graph Generation. In CP LIPIcs, vol. 210.
[KS24] Kirchweger, M. and Szeider, S. (2024). SAT Modulo Symmetries for Graph Generation and Enumeration. In ACM Trans. Comput. Log., 25(3).

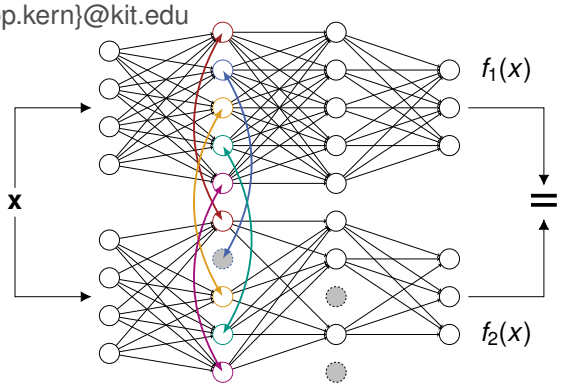
Contact: Daimy.Vancaudenberg@kuleuven.be

Karlsruhe Institute of Technology

Application-oriented Formal Verification Research Group
Institute of Information Security and Dependability (KASTEL)
Karlsruhe Institute of Technology

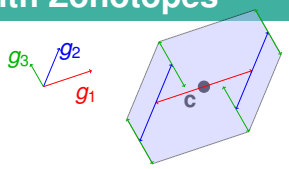
Revisiting Differential Verification: Equivalence Verification with Confidence

Samuel Teuber | Philipp Kern | Marvin Janzen | Bernhard Beckert
{teuber,philipp.kern}@kit.edu



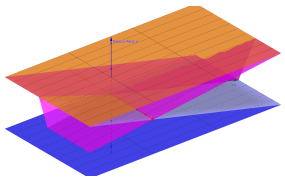
Differential Verification with Zonotopes

Single NN verification via Zonotopes
Propagate $\mathcal{Z}^i \supseteq f^{(i)}(x)$
- Abstract Transformer for $W \cdot \mathcal{Z} + \mathbf{b}$
- Abstract Transformer for $\mathrm{ReLU}(\mathcal{Z})$



$\mathcal{Z} = ((\mathbf{g}_1\ \mathbf{g}_2\ \mathbf{g}_3), \mathbf{c})$

Differential Verification via Zonotopes
Also propagate $\mathcal{Z}^i_\Delta \supseteq f^{(i)}_1(x) - f^{(i)}_2(x)$



A Zonotope Transformer for ReLU Differences

$$\mathrm{ReLU}(\mathcal{Z}_1) - \mathrm{ReLU}(\mathcal{Z}_2) = \\ = \mathrm{ReLU}(\mathcal{Z}_1) - \mathrm{ReLU}(\mathcal{Z}_1 - \mathcal{Z}_\Delta)$$

Requires distinction over nine cases

Differential Verification for Classification

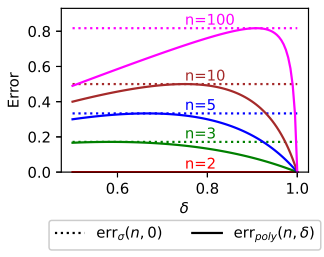
ε equivalence provides no guarantees for resulting classification!

- Use differential bounds for LP formulation
- Verify classification equivalence
- Threshold $T \geq 0$: Confidence of f_1

For all k:
Check $\max \leq 0$ for all $j \neq k$:

$$\max_x \ (\mathcal{Z}_2(x))_j - (\mathcal{Z}_2(x))_k$$
$$\text{s.t.} \ (\mathcal{Z}_1(x))_l + T \leq (\mathcal{Z}_1(x))_k \quad (l \neq k)$$
$$\mathcal{Z}_\Delta = \mathcal{Z}_1 - \mathcal{Z}_2$$

Linear Softmax Approximation



A **single polytope** over-approximation of outputs with confidence $\geq \delta$:

$$\left\{ \mathbf{z} \in \mathbb{R}^n \;\middle|\; \bigwedge_{\substack{j=1 \\ j \neq i}}^{n} \mathbf{z}_i - \mathbf{z}_j \geq \ln\left(\tfrac{\delta}{1-\delta}\right) \right\}$$

$$\bigcup |$$

$$\left\{ \mathbf{z} \in \mathbb{R}^n \;\middle|\; \mathrm{softmax}(\mathbf{z})_i \geq \delta \right\}$$

Equivalence Properties

For ReLU NNs f_1, f_2, input set $X \subseteq \mathbb{R}^I$ and a p-norm $\|\cdot\|_p$:

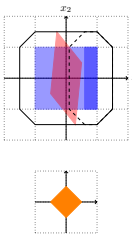
coNP-complete

ε equivalence [2, 3, 4]
$\forall x \in X. \|f_1(x) - f_2(x)\|_p \leq \varepsilon$

coNP-complete

Top-1 equivalence [2]
$\forall x \in X. \mathrm{argmax}_i\, f_{1,i}(x) = \mathrm{argmax}_i\, f_{2,i}(x)$

Classification Equivalence



- Differential verification significantly helps with ε-equivalence.
- Not helpful for Top-1-equivalence
- Lack of **asymmetry** in Top-1 property

Building on **confidence-based** verification [1]:

NP-hard

δ-Top-1 equivalence
f_1, f_2 have the same classification for every input where $\mathrm{softmax}(f_1(x)) \geq \delta$

Evaluation

Across **5 benchmarks**, VeryDiff (our tool) outperforms other (non-)differential NN verification tools:

	Benchmark	Variant	Equiv.		Counterex.		Speedup Median	Speedup Max
Standard ε eq.	ACAS	VeryDiff (ours)	**150**	(+24.0%)	**153**	(+2.0%)	—	—
		NNEquiv	37		142		37.3	8091.2
		MILPEquiv	16		3		7224.8	36297.0
		Marabou	110		109		141.3	10070.5
		α, β-CROWN	121		150		15.4	1954.1
	MNIST (VeriPrune)	VeryDiff (ours)	**352**	(+101.1%)	62	(-43.6%)	—	—
		NNEquiv	0		103		12.6	166.2
		Marabou	10		24		183.9	1390.8
		α, β-CROWN*	175		**110**		(516.9)	(4220.8)
NeuroDiff ε eq.	ACAS	VeryDiff (ours)	**169**	(+39.7%)	**161**	(+103.8%)	—	—
		NeuroDiff	121		79		43.1	16134.7
	MNIST (VeriPrune)	VeryDiff (ours)	**457**	(+11.5%)	**242**	(+404.1%)	—	—
		NeuroDiff	410		48		4.5	1086.8
δ-Top-1	LHC	VeryDiff (ours)	**77**	(327.8%)	—		—	—
		α, β-CROWN	18		—		324.5	11274.3

References

[1] Anagha Athavale et al. "Verifying Global Two-Safety Properties in Neural Networks with Confidence". In: *CAV'24*. DOI: 10.1007/978-3-031-65630-9_17.

[2] Marko Kleine Büning et al. "Verifying Equivalence Properties of Neural Networks with ReLU Activation Functions". In: *CP'20*. DOI: 10.1007/978-3-030-58475-7_50.

[3] Brandon Paulsen et al. "ReluDiff: differential verification of deep neural networks". In: *ICSE'20*. DOI: 10.1145/3377811.3380337.

[4] Samuel Teuber et al. "Geometric Path Enumeration for Equivalence Verification of Neural Networks". In: *ICTAI'21*. DOI: 10.1109/ICTAI52525.2021.00035.

Weakly acyclic diagrams: A data structure for infinite-state symbolic verification

Michael Blondin[1], Michaël Cadilhac[2], **Xinyi Cui**[3], Philipp Czerner[3], Javier Esparza[3], and **Jakob Schulz**[3]

[1] Université de Sherbrooke, Canada; [2] DePaul University, Chicago, USA; [3] Technical University of Munich, Germany

1. Introduction

Ordered Binary Decision Diagrams (OBDDs) [1, 2]

- ◇ data structure representing fixed-length languages
- ◇ efficient top-down dynamic programming for operations
- ◇ used for finite-state symbolic model-checking

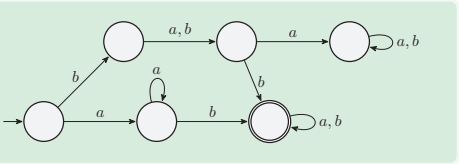
we generalize OBDDs to a class of infinite languages

Weakly Acyclic Diagrams

- ◇ data structure representing weakly acyclic languages
- ◇ maintains algorithmic advantages of OBDDs
- ◇ can be used for model checking in infinite-state systems

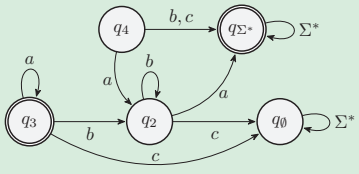
2. Weakly Acyclic Languages

- ◇ Weakly acyclic languages: all cycles in DFA are self-loops
- ◇ closed under union, intersection and complementation



3. Weakly Acyclic Diagrams

- ◇ data structure representing weakly acyclic languages using a table of nodes
- ◇ Node: state identifier, successors, accepting flag



id	succ. tuple s	flag b
$q_\emptyset$	[SELF, SELF, SELF]	0
q_{Σ^*}	[SELF, SELF, SELF]	1
q_2	$[q_{\Sigma^*}, \text{SELF}, q_\emptyset]$	0
q_3	$[\text{SELF}, q_2, q_\emptyset]$	1
q_4	$[q_2, q_{\Sigma^*}, q_{\Sigma^*}]$	0

References

[1] S. Akers. 1978.

[2] S. Chaki and A. Gurfinkel. 2018.

[3] A. Finkel and Ph. Schnoebelen. 2001.

[4] T. Geffroy, J. Leroux, and G. Sutre. 2017.

[5] A. Heußner, Gall T. L., and G. Sutre. 2009.

[6] P. Ganty, C. Meuter, G. Delzanno, G. Kalyon, J.F. Raskin, and L. Van Begin. 2007.

4. Example of a Binary Operation

$intersection(q_1, q_2) \rightsquigarrow q_{1\cap 2}$

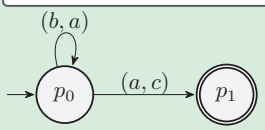
1. **check base cases**
   - ◇ $q_1 = q_\emptyset \vee q_2 = q_\emptyset \implies q_{1\cap 2} = q_\emptyset$
   - ◇ $q_1 = q_{\Sigma^*} \implies q_{1\cap 2} = q_2$ and $q_2 = q_{\Sigma^*} \implies q_{1\cap 2} = q_1$
2. **compute successor tuple $s_{1\cap 2}$ and flag $b_{1\cap 2}$ for $q_{1\cap 2}$**
   - ◇ $s_{1\cap 2}[a] = \begin{cases} \text{SELF} & \text{if } s_1[a] = q_1 \wedge s_2[a] = q_2 \\ intersection(s_1[a], s_2[a]) & \text{otherwise} \end{cases}$
   - ◇ $b_{1\cap 2} = b_1 \wedge b_2$

- ◇ given nodes for L_1 and L_2 compute node for $L_1 \cap L_2$
- ◇ memoization to avoid exponential complexity

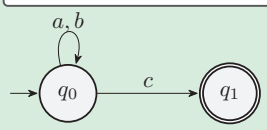
5. Pre Operation

- ◇ Transducers: automata over the alphabet $\Sigma \times \Sigma$ for fixed-length regular relations $R \subseteq \Sigma^* \times \Sigma^*$
- ◇ $\text{Pre}_R(L) := \{u \in \Sigma^* : (u, v) \in R, v \in L\}$ for a relation R and a language L

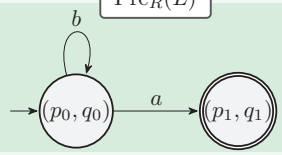
Transducer for relation R



Weakly acyclic DFA for L



$\text{Pre}_R(L)$



- ◇ algorithm computing $\text{Pre}_R(L)$ given transducer for R and weakly acyclic DFA for L under assumption: language $\text{Pre}_R(L)$ is weakly acyclic
- ◇ main idea: apply powerset construction on pairing of transducer and DFA contracting cycles in the process

$pre(M = \{(p_0, q_0)\} \rightsquigarrow q_M$

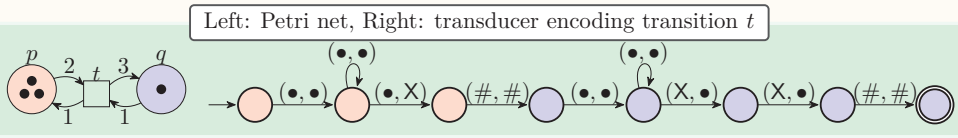
1. mark M
2. $M_a = \bigcup_{b\in\Sigma}\{(p', q') : (p, q) \in M, p \xrightarrow{(a,b)} p', q \xrightarrow{(b)} q'\}$
3. $s_M[a] = \begin{cases} \text{SELF} & \text{if } M_a \text{ is marked} \\ pre(M_a) & \text{otherwise} \end{cases}$
4. $b_M \iff \exists (p, q) \in M : p, q$ both accepting states
5. unmark M

- ◇ M set of state pairs, representing one state in DFA for $\text{Pre}_R(L)$
- ◇ use of markings to detect and remove cycles
- ◇ polynomial-time improvement if input transducer and DFA satisfy specific condition such that determinism is guaranteed

6. Application: Backwards Reachability

- ◇ system as a set of configurations with a transition relation like lossy channel systems, Petri nets or broadcast protocols
- ◇ Backwards Reachability algorithm: computes the set of all predecessors of a given upward-closed set of configurations, used for safety verification in systems [3]
- ◇ use Weakly Acyclic Diagrams for Backwards Reachability
  - ∗ set of configurations $\rightsquigarrow$ weakly acyclic language
  - ∗ transitions $\rightsquigarrow$ transducers
  - ∗ predecessor computation using *pre*

Left: Petri net, Right: transducer encoding transition t



7. Experimental Results

- ◇ library WADL for weakly acyclic diagrams implementing backwards reachability for lossy channel systems, Petri nets and broadcast protocols
- ◇ compared WADL to established safety verification tools BML[4], McScM[5] and MIST[6]
- ◇ results show that WADL is competitive, solving most of the instances

Benchmark	BML	McScM	WADL
`tcp_simplest_err`	0.04	0.01	0.04
`BAwCC_enh`	4.17	124.72	0.50
`tcp_simplest`	0.03	0.34	0.06
`peterson_3`	to	to	19.50
`ring2`	0.36	so	0.04
`brp_like_modified`	0.39	0.39	0.29
`simple_server`	0.13	0.25	0.06
`pop3`	400.20	4.73	2.07

SV-COMP'25 and Test-Comp'25 Posters

14th Competition on Software Verification

Dirk Beyer

SoSy-Lab Software Systems

LMU Ludwig-Maximilians-Universität München

Participants

Table 1: Competition candidates and representing jury members; Hors Concours participants are not listed since they are not represented by a jury member. new for first-time participants, meta for meta-verifiers

Participant	Jury member	Affiliation
2LS	V. Malík	BUT, Czechia
aise	Z. Chen	NUDT, China
AProVE	N. Lommen	RWTH Aachen, Germany
BRICK	L. Bu	Nanjing U., China
Bubaak	M. Chalupa	ISTA, Austria
Bubaak-SpLit	M. Chalupa	ISTA, Austria
CoOpeRace meta new	V. Vojdani	U. Tartu, Estonia
CPAchecker	M. Lingsch-Rosenfeld	LMU Munich, Germany
CPV	P.-C. Chien	LMU Munich, Germany
Dartagnan	H. Ponce de León	Huawei Dresden, Germany
Deagle	F. He	Tsinghua U., China
EmergenTheta	L. Bajczi	BME Budapest, Hungary
ESBMC-incr	T. Wu	U. Manchester, UK
ESBMC-kind	T. Wu	U. Manchester, UK
GDart	F. Howar	TU Dortmund, Germany
Goblint	S. Saan	U. Tartu, Estonia
Hornix new	M. Blicha	U. Lugano, Switzerland
Java-Ranger	S. Hussein	Ain Shams U., Egypt
JBMC	P. Schrammel	Diffblue, UK
Korn	G. Ernst	LMU Munich, Germany
MLB	L. Bu	Nanjing U., China
Mopsa	R. Monat	Inria & U. Lille, France
Nacpa meta new	H. Wachowitz	LMU Munich, Germany
Proton	R. Metta	TCS, India
RacerF new	T. Dacík	BUT, Czechia
SVF-SVC new	M. Richards	U. New South Wales, AU
sv-sanitizers	S. Saan	U. Tartu, Estonia
SWAT	N. Loose	U. Luebeck, Germany
Symbiotic	M. Jonáš	Masaryk U., Czechia
Theta	L. Bajczi	BME Budapest, Hungary
Thorn new	L. Bajczi	BME Budapest, Hungary
UAutomizer	M. Heizmann	U. Freiburg, Germany
UGemCutter	D. Klumpp	U. Freiburg, Germany
UKojak	M. Bentele	U. Freiburg, Germany
UTaipan	D. Dietsch	U. Freiburg, Germany

Cummulative Score

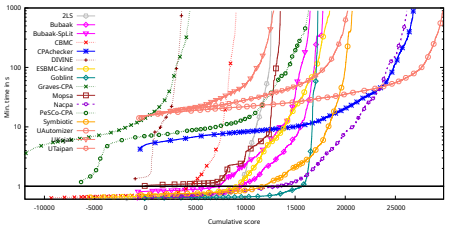


Figure 1: Quantile plot for the category *C-Overall*.

Ranking

Table 5: Overview of the top-three verifiers for each category; measurements for CPU time rounded to two significant digits.

Rank	Verifier	Score	CPU Time (in h)	Solved Tasks	Unconf. Tasks	False Alarms	Wrong Proofs
ReachSafety (11268 tasks, max. score 17860)							
1	CPAchecker	**10368**	150	6653	230	2	
2	ESBMC-kind	8717	69	6830	599	-	**14**
3	CPV	7755	160	6235	438	27	
MemSafety (4042 tasks, max. score 6469)							
1	CPAchecker	**4892**	18	3818	1		
2	Symbiotic	4479	2.3	3671	0		**1**
3	UAutomizer	3909	37	2280	2		**1**
ConcurrencySafety (3175 tasks, max. score 5733)							
1	Deagle	**4604**	3.1	2500	38	1	**4**
2	Dartagnan	3385	17	2012	30	3	**3**
3	UGemCutter	3144	50	1805	48		
NoOverflows (8211 tasks, max. score 13297)							
1	UAutomizer	**11074**	68	6724	13		
2	UTaipan	10736	74	6622	11	1	**2**
3	UKojak	8878	54	5910	2		
Termination (2328 tasks, max. score 4079)							
1	Proton	**3685**	24	1942	159	1	
2	UAutomizer	3334	18	1667	4		
3	AProVE	2219	32	1006	43		
SoftwareSystems (4329 tasks, max. score 7131)							
1	CPAchecker	**2178**	30	2022	55		
2	Mopsa	2086	20	2212	0		
3	Symbiotic	1822	6.1	1487	231		**1**
FalsificationOverall (30758 tasks, max. score 10675)							
1	CPAchecker	**6999**	100	7100	67	2	
2	Symbiotic	6459	28	6379	37		
3	Bubaak	5565	18	5739	236	9	
Overall (33353 tasks, max. score 55561)							
1	UAutomizer	**29710**	270	16677	196		**8**
2	CPAchecker	26786	240	20506	372	6	**1**
3	Symbiotic	20691	63	14324	628		**3**
JavaOverall (673 tasks, max. score 926)							
1	Java-Ranger	**676**	5.7	491	13		**1**
2	JBMC	628	0.32	430	91		
3	GDart	627	2.1	460	15		

Features

Table 2: Algorithms and techniques that the participating verification systems used; new for first-time participants, ∅ for hors-concours participation, and meta for meta-verifiers

Tool	CEGAR	Predicate Abstraction	Symbolic Execution	Bounded Model Checking	k-Induction	Property-Directed Reach.	Explicit-Value Analysis	Numeric Interval Analysis	Shape Analysis	Separation Logic	Bit-Precise Analysis	ARG-Based Analysis	Lazy Abstraction	Interpolation	Automata-Based Analysis	Concurrency Support	Ranking Functions	Evolutionary Algorithms	Algorithm Selection	Portfolio	Task Translation
2LS				✓	✓			✓		✓	✓					✓					
aise				✓												✓					
AProVE																	✓				
BRICK	✓			✓	✓			✓								✓					
Bubaak				✓							✓					✓	✓			✓	
Bubaak-SpLit				✓	✓						✓				✓	✓			✓	✓	
CBMC∅				✓							✓					✓					
COASTAL∅				✓												✓					
Concurrent-W2T																✓					
CoOpeRace meta new																✓			✓	✓	
CPAchecker	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	
CPALockator∅	✓	✓					✓				✓		✓	✓		✓					
CPA-bam-bnb∅	✓	✓					✓				✓		✓	✓							
CPA-bam-smg∅																					
CPA-w2t∅						✓							✓		✓						
CProver-w2t∅				✓																	
CPV	✓	✓		✓	✓	✓					✓			✓					✓	✓	✓
Crux∅				✓							✓					✓					
CSeq∅				✓							✓					✓					
Dartagnan				✓							✓					✓					
Deagle				✓							✓					✓					
DIVINE∅			✓				✓				✓								✓	✓	
EBF∅				✓																	
EmergenTheta	✓	✓		✓	✓						✓			✓					✓	✓	✓
ESBMC-incr				✓	✓						✓					✓					
ESBMC-kind				✓	✓		✓	✓			✓					✓					
Frama-C-SV∅								✓			✓										
Gazer-Theta∅	✓	✓		✓			✓				✓	✓	✓	✓						✓	
GDart∅			✓								✓									✓	
GDart-LLVM∅				✓							✓										
Goblint								✓			✓					✓			✓		
Graves-CPA∅ meta	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓		
GWIT∅			✓								✓									✓	
Hornix new						✓					✓										
Infer∅								✓	✓	✓										✓	
Java-Ranger			✓								✓										
JavHorn∅	✓	✓				✓		✓					✓	✓							
JBMC				✓							✓					✓					
JCWIT∅				✓																	
JDart∅			✓								✓									✓	
Korn	✓	✓	✓				✓				✓									✓	
Lazy-CSeq∅				✓							✓					✓					
LF-checker∅																					
LIV																					✓
Locksmith∅																✓					
MetaVal																				✓	
MetaVal-++ new																				✓	
MLB				✓							✓									✓	
Mopsa							✓														
Nacpa meta new																			✓	✓	
NITWIT∅							✓														
PeSCo-CPA∅ meta	✓	✓		✓	✓		✓	✓			✓	✓	✓	✓		✓	✓		✓	✓	
PIChecker∅	✓	✓									✓	✓		✓		✓			✓		
Pinaka∅				✓	✓						✓										
PredatorHP∅								✓													
Proton				✓																	
RacerF new																✓					
SPF∅				✓				✓								✓					
SVF-SVC new				✓																	
sv-sanitizers																✓					
SWAT				✓																	
Symbiotic				✓	✓		✓	✓			✓					✓				✓	
Symbiotic-Witch				✓																	✓
Theta	✓	✓			✓		✓				✓	✓		✓		✓			✓	✓	✓
Thorn new	✓	✓																	✓	✓	
UAutomizer	✓	✓									✓		✓	✓	✓	✓	✓		✓	✓	
UGemCutter	✓	✓									✓		✓	✓	✓	✓			✓	✓	
UKojak	✓	✓									✓		✓	✓							
URefferee new	✓	✓											✓	✓	✓	✓			✓	✓	
UTaipan	✓	✓					✓	✓			✓		✓	✓	✓	✓			✓	✓	
VeriAbs∅	✓			✓	✓		✓	✓									✓		✓	✓	
VeriAbsL∅	✓			✓	✓		✓	✓									✓		✓	✓	
VeriOover∅																					
Wit-Llva																					
Witch			✓																		

More Information

https://sv-comp.sosy-lab.org/2025/

Score Schema

Table 6: Scoring schema for SV-COMP 2025 (unchanged from 2021)

Reported result	Points	Description
Unknown	0	Failure to compute verification result
False correct	+1	Violation of property in program was correctly found and a validator confirmed the result based on a witness
False incorrect	−16	Violation reported but property holds (false alarm)
True correct	+2	Program correctly reported to satisfy property and a validator confirmed the result based on a witness
True incorrect	−32	Incorrect program reported as correct (wrong proof)

Reference

Report

D. Beyer. State of the art in software verification and witness validation: SV-COMP 2025. In *Proc. TACAS*, LNCS . Springer, 2025

Acknowledgment

We thank the verification community for contributing their tools to the evaluation.

Frameworks

Table 3: Solver libraries and frameworks that are used as components in the participating verification systems; new for first-time participants, ∅ for hors-concours participation and meta for meta-verifiers

Tool	CPAchecker	CProver	Esbmc	Jpf	Ultimate	JavaSMT	MathSAT	Cvc	SMTInterpol	Z3	MathSAT	Aprox
2LS		✓								✓	✓	
aise								✓		✓		
AProVE										✓		
BRICK										✓	✓	
Bubaak										✓		
CBMC∅		✓									✓	
COASTAL∅				✓								
CPAchecker	✓					✓	✓					✓
CPALockator∅	✓					✓	✓					
CPA-bam-bnb∅	✓					✓	✓					
CPA-bam-smg∅	✓					✓	✓					
CPV										✓	✓	
Crux∅		✓								✓		
CSeq∅		✓				✓					✓	
Dartagnan						✓						
Deagle											✓	
EBF∅			✓				✓					
ESBMC-incr			✓				✓					
ESBMC-kind			✓				✓					
GDart								✓		✓		
GDart-LLVM∅										✓		
Goblint												✓
Graves-CPA∅ meta	✓					✓	✓					
Hornix new										✓		
Java-Ranger				✓								
JBMC		✓									✓	
JDart∅				✓				✓		✓		
Korn										✓		
Lazy-CSeq∅		✓									✓	
Mopsa												✓
Nacpa meta new	✓											
PeSCo-CPA∅ meta	✓					✓	✓					
PIChecker∅	✓					✓	✓		✓			
SPF∅				✓						✓		
SWAT										✓		
Symbiotic										✓		
UAutomizer					✓		✓	✓	✓	✓		
UGemCutter					✓		✓	✓	✓	✓		
UKojak					✓				✓	✓		
URefferee new					✓		✓	✓	✓	✓		
UTaipan					✓		✓	✓	✓	✓		
VeriAbs∅	✓	✓								✓	✓	
VeriAbsL∅	✓	✓								✓	✓	

Results

Table 4: Quantitative overview over all regular results; empty cells are used for opt-outs, new for first-time participants, ∅ for hors-concours participation, and meta for meta-verifiers

Participant	ReachSafety 11268 tasks max. score 17860	MemSafety 4042 tasks max. score 6469	ConcurrencySafety 3175 tasks max. score 5733	NoOverflows 8211 tasks max. score 13297	Termination 2328 tasks max. score 4079	SoftwareSystems 4329 tasks max. score 7131	FalsificationOverall 30758 tasks max. score 10675	Overall 33353 tasks max. score 55561	JavaOverall 673 tasks max. score 926
2LS	6053	686	0	6887	1703	1	1938	12658	
aise									
AProVE					**2219**				
BRICK									
Bubaak	6684	3355	-190	6572	1491	1697	**5565**	17768	
Bubaak-SpLit	6053	3349	-189	6556	1122	1649	5562	16497	
CoOpeRace meta new									
CPAchecker	**10368**	**4892**	1770	8777	1301	**2178**	**6999**	**26786**	
CPV	**7755**								
Dartagnan			**3385**						
Deagle			**4604**						
EmergenTheta	2106			-361	620				
ESBMC-incr			2155						
ESBMC-kind	**8717**	3158	2155	8668	1115	-1948	3741	18444	
Goblint	2427	2198	2448	8486	969	545		17266	
Hornix new									
Korn									
Mopsa	2807	2697	0	8491	0	**2086**		13521	
Nacpa meta new	10270	4887	1453	8843	1290	2127	6997	26131	
Proton					**3685**				
RacerF new									
sv-sanitizers		861		1723					
SVF-SVC new	-68717	-10965		-19469	0				
Symbiotic	7097	**4479**	60	7704	1411	**1822**	**6459**	**20691**	
Theta	3277	600	2275	-170	712				
Thorn new	-2519		298	-286	536				
UAutomizer	5666	**3909**	2993	**11074**	**3334**	654	4762	**29710**	
UGemCutter			**3144**						
UKojak	4935	2939	0	**8878**	0	300	3848	12872	
UTaipan	6007	3711	2593	**10736**	0	288	4695	20244	
GDart									**627**
Java-Ranger									**676**
JBMC									**628**
MLB									579
SWAT									508
CBMC∅	1330	1885	819	7200	1199	-2586	-3581	9100	
CPA-bam-bnb∅						-2370			
CPA-bam-smg∅		3249				-4079			
CPALockator∅			-4967						
Crux∅	2133			608					
CSeq∅			-12720						
DIVINE∅	4629	502	351	0	0	72	352	3680	
EBF∅			360						
Frama-C-SV∅				1573					
Gazer-Theta∅									
GDart-LLVM∅									
Graves-CPA∅ meta	4041					-670	-820	4508	
Infer∅	-96489		-8970	-76213		-32987			
Lazy-CSeq∅			-15153						
LF-checker∅			396						
Locksmith∅									
PeSCo-CPA∅ meta	6269					-1598	2435	16328	
PIChecker∅			459						
Pinaka∅	2448			958	922				
PredatorHP∅		4733							
VeriAbs∅	11012								
VeriAbsL∅	11224								
VeriOover∅									
WitnessMap									
COASTAL∅									-3960
JayHorn∅									248
JDart∅									-1224
SPF∅									184

CPV: A Circuit-Based Program Verifier

Po-Chun Chien[1] **and Nian-Ze Lee**[2,1]

[1]LMU Munich [2]National Taiwan University

po-chun.chien@sosy.ifi.lmu.de nzlee@ntu.edu.tw

Motivation

HWMCC [2]
(Input: Btor2 circuit)

ABC [5], AVR [10], ...

Applicable?

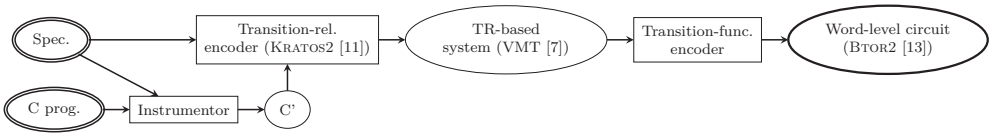
SV-COMP [1]
(Input: C program)

Try CPV!

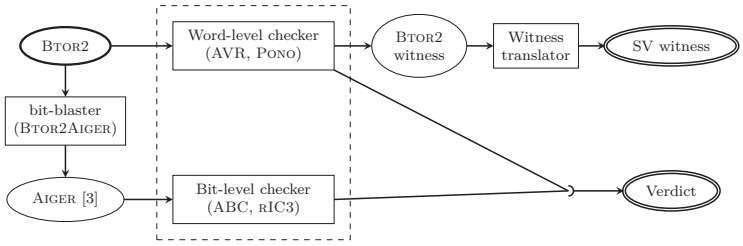
gitlab.com/sosy-lab/software/cpv

TACAS 2024 [6]

Software Architecture



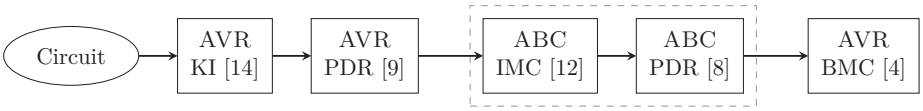
Frontend: Task Translation



Managed by CoVeriTeam

Backend: Model Checking

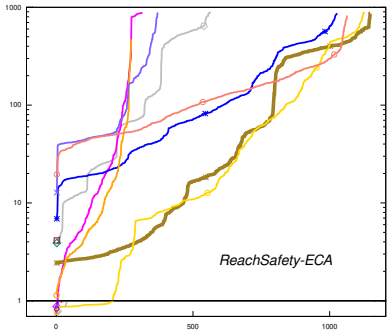
Strategy for SV-COMP 2025

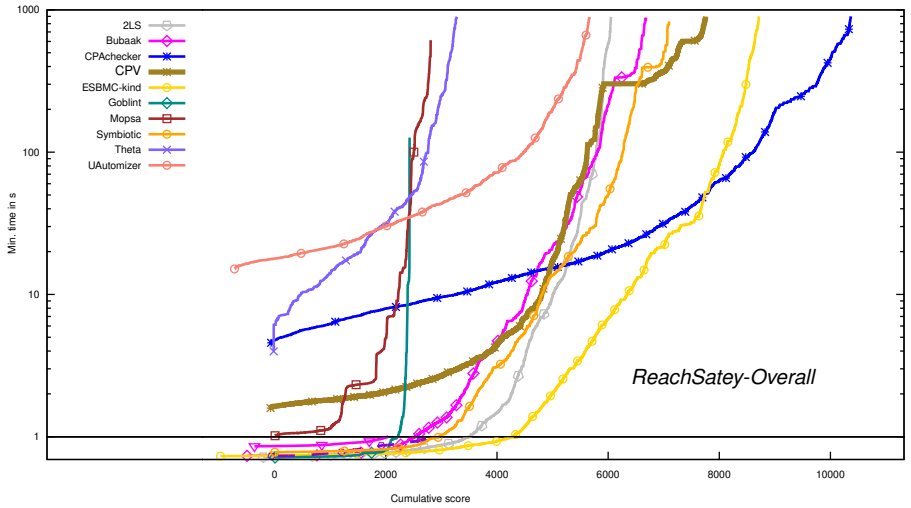


if Btor2-to-Aiger translation succeeds

Evaluation Results at SV-COMP 2025

- **3rd** place in *ReachSafety-Overall*, including
- **1st** in *ReachSafety-BitVectors* and *-ECA*
- **2nd** in *ReachSafety-Hardware* and *-ProductLines*
- **3rd** in *ReachSafety-Combinations* and *-Hardness*





Summary

- Sequential circuits can serve as an intermediate representation for software verification.
- Offer different encoding options.
- Leverage powerful word- and bit-level hardware model checkers as backend.
- Perform competitively against established verifiers in SV-COMP.
- Ongoing work:
  - Support more verification properties (e.g., no-overflow and termination)
  - Export correctness witnesses
  - Apply circuit optimization to improve the performance of verification

References

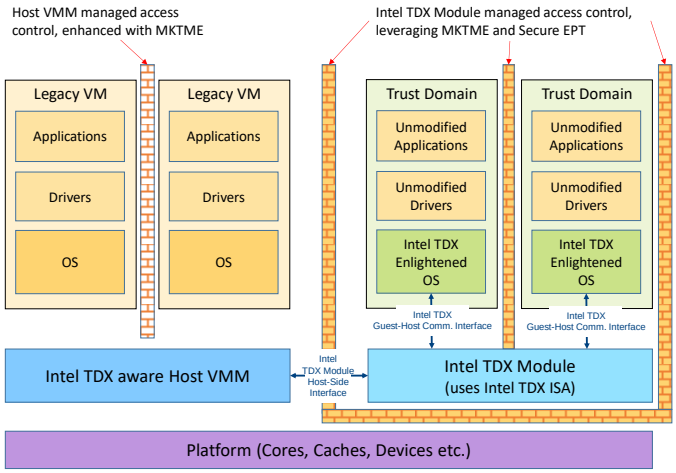
[1] Beyer, D., Strejček, J.: Improvements in software verification and witness validation: SV-COMP 2025. In: Proc. TACAS (3). LNCS 15698 (2025)
[2] Biere, A., Froleyks, N., Preiner, M.: Hardware model checking competition 2024. In: Proc. FMCAD. pp. 7–7 (2024)
[3] Biere, A., Heljanko, K., Wieringa, S.: AIGER 1.9 and beyond. Tech. Rep. 11/2, Institute for Formal Models and Verification, Johannes Kepler University (2011)
[4] Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers **58**, 117–148 (2003)
[5] Brayton, R., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: Proc. CAV. pp. 24–40. LNCS 6174 (2010)
[6] Chien, P.C., Lee, N.Z.: CPV: A circuit-based program verifier (competition contribution). In: Proc. TACAS (3). pp. 365–370. LNCS 14572 (2024)
[7] Cimatti, A., Griggio, A., Tonetta, S.: The VMT-LIB language and tools. In: Proc. SMT. CEUR Workshop Proceedings, vol. 3185, pp. 80–89 (2022)
[8] Eén, N., Mishchenko, A., Brayton, R.K.: Efficient implementation of property directed reachability. In: Proc. FMCAD. pp. 125–134 (2011)
[9] Goel, A., Sakallah, K.: Model checking of Verilog RTL using IC3 with syntax-guided abstraction. In: Proc. NFM. pp. 166–185 (2019)
[10] Goel, A., Sakallah, K.: AVR: Abstractly verifying reachability. In: Proc. TACAS. pp. 413–422. LNCS 12078 (2020)
[11] Griggio, A., Jonáš, M.: Kratos2: An SMT-based model checker for imperative programs. In: Proc. CAV. pp. 423–436 (2023)
[12] McMillan, K.L.: Interpolation and SAT-based model checking. In: Proc. CAV. pp. 1–13. LNCS 2725 (2003)
[13] Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2, BtorMC, and Boolector 3.0. In: Proc. CAV. pp. 587–595. LNCS 10981 (2018)
[14] Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: Proc. FMCAD, pp. 127–144. LNCS 1954 (2000)

SV-COMP Benchmark: Verifying Intel TDX Module

Dirk Beyer[1], Po-Chun Chien[1], Nian-Ze Lee[2,1], & Thomas Lemberger[1]

[1]LMU Munich [2]National Taiwan University

Intel Trust Domain Extensions



Source: Fig. 2-1 in Intel TDX Module v1.5 Base Arch. Spec. [2]

Defining Verification Tasks

```yaml
- name: tdh_mng_create__requirement__expected
  target:
    filename: formal/harness/tdh_mng_create_harness.c
    method: tdh_mng_create__valid_entry
  before_target:
    - filename: formal/src/initialization.c
      method: init_tdx_general
    - filename: formal/src/initialization.c
      method: init_vmm_dispatcher
    - filename: formal/harness/tdh_mng_create_harness.c
      method: tdh_mng_create__common_precond
  after_target:
    - filename: formal/harness/tdh_mng_create_harness.c
      method: tdh_mng_create__common_postcond
  properties:
    - property_file: unreach-call.prp
      expected_verdict: true
```

Nondet Initialization of `struct`

Havoc memory: by assigning a nondeterministic value to each byte

Havoc object: by nondeterministically initializing each field of the type (if a field is a non-primitive type, recursively initialize it)

Verifier builtin: e.g., `__CPROVER_havoc_object` in CBMC

```c
void _NONDET_struct_tdvps_t(tdvps_t* dest) {
  _NONDET_custom_type(dest, sizeof(tdvps_t));
}
void _NONDET_custom_type(void* base, unsigned int size) {
  for (int i = 0; i < size; i++)
    *((char*)base + i) = _NONDET_uint8t();
}
```

Initialization by havocking memory

```c
void _NONDET_struct_tdvps_s(struct tdvps_s *dest) {
  _NONDET_struct_tdvps_ve_info_s(&((*dest).ve_info));
  _NONDET_array_1D_unsigned_char(&((*dest).reserved_0), 128);
  // ... snipped ...
}
void _NONDET_array_1D_unsigned_char(unsigned char (*dest)[],
                                    int dim0) {
  for (int i = 0; i < dim0; i++)
    (*dest)[i] = _NONDET_uchar();
}
```

Initialization by havocking object

References

[1] Intel TDX Module v1.5 ABI Specification, https://www.intel.com/content/www/us/en/content-details/795475/intel-tdx-module-v1-5-abi-specification.html, accessed: 2024-05-01

[2] Intel Trust Domain Extensions, https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html, accessed: 2024-05-01

ABI Specifications

- TDs and VMM communicate through **A**pplication **B**inary **I**nterfaces
- **Goal**: Verify TDX ABIs (implemented in C + assembly) adhere to the specification under all inputs

Table 5.145: TDH.MNG.CREATE Input Operands Definition

Operand	Description		
RAX	SEAMCALL instruction leaf number and version, see 5.3.1		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the SEAMCALL interface function
	23:16	Version Number	Selects the SEAMCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RCX	The physical address of a page where TDR will be created (HKID bits must be 0)		
RDX	**Bits**	**Name**	**Description**
	15:0	HKID	The TD's ephemeral private HKID
	63:16	Reserved	Reserved: must be 0

Table 5.146: TDH.MNG.CREATE Output Operands Definition

Operand	Description
RAX	SEAMCALL instruction return code – see 5.3.1
Other	Unmodified

Example: Specification of ABI `TDH.MNG.CREATE` [1]

Verification Harnesses

- Initialize global data and assume preconditions
- Mock access to externally defined data and model inline assembly
- Check postconditions

```c
_STATIC_INLINE_ tdx_module_local_t *get_local_data(void) {
#ifdef TDXFV_NO_ASM
  return &local_data_fv;
#else
  uint64_t local_data_addr;
  _ASM_("movq %%gs:%c[local_data], %0\n\t"
        : "=r"(local_data_addr)
        : [local_data] "i"(offsetof(
            tdx_module_local_t, local_data_fast_ref_ptr)));
  return (tdx_module_local_t *)local_data_addr;
#endif
}
```

Contributions

- 290 tasks from 16 host-side ABIs (`TDH`) and 5 guest-side ABIs (`TDG`)
- HarnessForge (`gitlab.com/sosy-lab/software/harnessforge`):
  - Assembles single-file verification tasks from real-world C projects
  - Slices off code irrelevant to verification tasks
- Next steps:
  - Experiment with effect of different initialization strategies
  - Develop more tooling to support harness generation (e.g., harness-specific linter)
  - Establish custom annotations for initializing complex types in SV-COMP community

More Information

Intel TDX Module | Verification tasks | HarnessForge

This work is supported by a research gift from Intel.

AProVE (KoAT + LoAT)

Automatic Termination Analysis of C Programs

Nils Lommen, Florian Frohn, and Jürgen Giesl

Overview

- AProVE (KoAT + LoAT) [1] is a framework to analyze termination of **C** Programs
- Programs are transformed into *Integer Transition Systems (ITSs)*
- ITSs are analyzed by our tools KoAT [2] and LoAT [3]

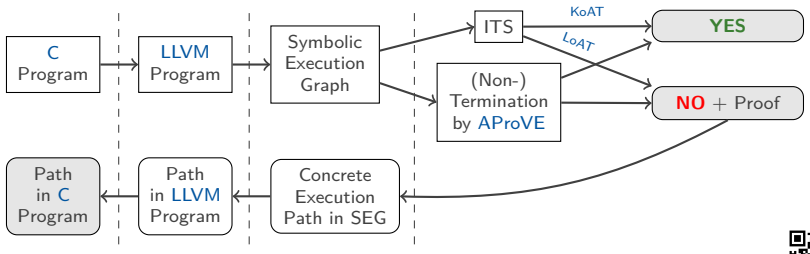


Figure 1: AProVE (KoAT + LoAT) for (Dis)proving Termination

Exemplary C Program

Does the following program terminate?

```
1  extern int _nondet(void);
2
3  int main() {
4    int x = _nondet();
5    int y = _nondet();
6
7    while(x < y) {
8      x = 3*x;
9      y = 2*y;
10   }
11   return 0;
12 }
```

LLVM Program

- **C** program is compiled into **LLVM** code using Clang.
- **LLVM** fragment of the loop body:

```
%10 = load %1        # load x
%11 = mul 3 %10      # multiply x by 3
store %11, %1        # store x
%12 = load %2        # load y
%13 = mul 2 %13      # multiply y by 2
store %13, %2        # store y
br %6                # jump to loop guard
```

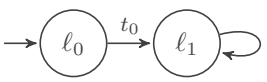
Symbolic Execution Graph (SEG) & ITS

SEG represents all possible program runs, augmented with invariants:

- Its nodes are *abstract states* that represent sets of actual program states
- SEG handles the heap, pointer arithmetic, and recursive data structures
- **LLVM** code is transformed automatically into an SEG

ITSs are a simple language for integer programs:

- Turing-complete formalism with only integer variables over $\mathbb{Z}$
- SEG is transformed into ITS



$t_1 : \varphi = (x < y)$
$\eta(x) = 3 \cdot x$
$\eta(y) = 2 \cdot y$

KoAT (Termination & Upper Time Bounds)

- Automated complexity and termination analysis of ITSs
- Alternating modular inference of runtime and size bounds
- How *often* can a transition be executed?
  - Multiphase Linear Ranking Functions
    ↪ Use SMT-solver Z3 to infer well-founded relation
  - TWN-Loops
    ↪ Reduce termination problem to SMT problem
    Completeness for the class of so-called TWN-loops
- How *large* are the variables?
  - Compute bounds for each change of a variable
    ↪ Over-approximate the number of changes by runtime bounds
  - Use runtime bounds and closed forms of loops

LoAT (Non-Termination and more)

Features

- *non*-termination
- *lower* time bounds
- safety / unsafety

Techniques

ADCL DFS + acceleration
ABMC BFS + acceleration
TRL BFS + recurrence analysis

Non-term. via Acceleration Driven Clause Learning

- Depth-first exploration of state space
- Applies *acceleration* when a loop is encountered

under-approximation of the loop's transitive closure

- Non-term. proofs as "by-product" of acceleration
- Exploits *redundancy* to cut off infinite branches

References

[1] Nils Lommen and Jürgen Giesl. AProVE (KoAT + LoAT) Website: `https://koat.verify.rwth-aachen.de/svcomp25`.

[2] Nils Lommen, Éléanore Meyer, and Jürgen Giesl. KoAT Website: `https://koat.verify.rwth-aachen.de/`.

[3] Florian Frohn and Jürgen Giesl. LoAT Website: `https://loat-developers.github.io/LoAT/`.

AProVE

RWTH AACHEN UNIVERSITY

SoSy-Lab Software Systems

CPAchecker

A Tool for Configurable Program Analysis

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

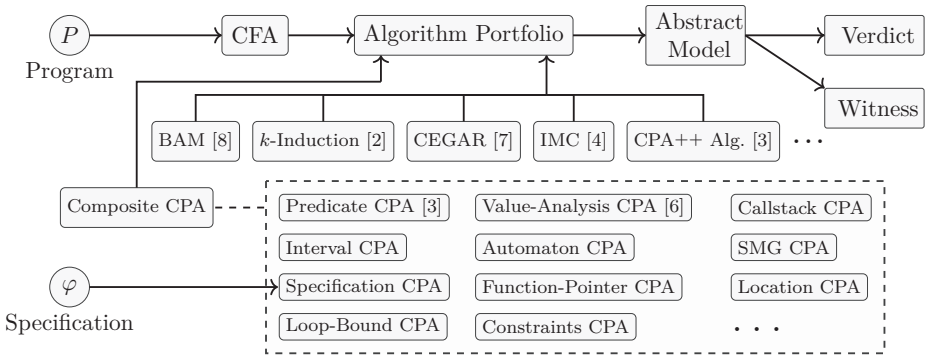
Daniel Baier, Dirk Beyer, Marek Jankola, Matthias Kettl, Marian Lingsch-Rosenfeld, and Philipp Wendler

CPAchecker

CPAchecker is a modern and versatile framework for building software-verification analyses from well-known concepts that match the user's requirements.

cpachecker.sosy-lab.org

Overview



Verification

For SV-COMP 2025, we used CPAchecker 4.0 with strategy selection to choose a parallel portfolio of analyses suitable for a given verification task. More details can be found in our tutorial [1].

- Strategy selection chooses a ***parallel*** portfolio of analyses. The parallel composition is denoted by the || symbol
- Support all properties and categories of C programs
- 1st place in the categories *FalsificationOverall*, *ReachSafety*, *MemorySafety*, and *SoftwareSystems*
- 2nd place in the category *Overall*
- Only 7 wrong results out of 33 353 tasks (0.02 %)
- New and improved analyses for:
  - Reachability
  - Memory safety
  - Termination

Paper [1] available here

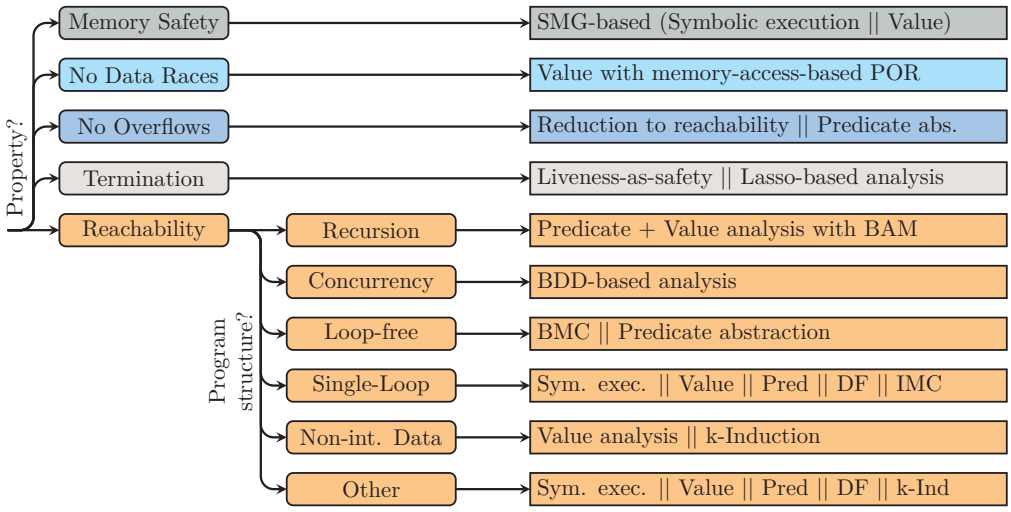
Witness Validation

For witness validation in SV-COMP 2025, we used CPAchecker 4.0 with strategy selection to choose (based on the input) a mature analysis. More details can be found in our competition contribution [5].

- Strategy selection chooses a mature analysis that is suitable for a given validation task
- Support all witness types and formats, all properties for which witnesses exist, and all categories of C programs
- 1st place in the categories *MemorySafety*, *Termination*, and *SoftwareSystems* for violation witnesses 1.0
- 2nd place in all categories for correctness witnesses 2.0, most categories for correctness witnesses 1.0, and most categories for violation witnesses 2.0

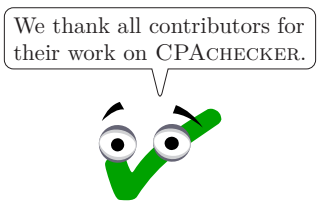
Paper [5] available here

Verification Strategy for SV-COMP 2025



Contributors

CPAchecker is an open-source project, mainly developed by the Software and Computational Systems Lab at LMU Munich, and is used and extended by international associates from U Passau, U Oldenburg, U Paderborn, ISP RAS, TU Prague, TU Vienna, TU Darmstadt, and VERIMAG in Grenoble, along with several other universities and institutes.



References

[1] Baier, D., Beyer, D., Chien, P.C., Jakobs, M.C., Jankola, M., Kettl, M., Lee, N.Z., Lemberger, T., Lingsch-Rosenfeld, M., Wachowitz, H., Wendler, P.: Software verification with CPAchecker 3.0: Tutorial and user guide. In: Proc. FM. pp. 543–570. LNCS 14934, Springer (2024). https://doi.org/10.1007/978-3-031-71177-0_30

[2] Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: Proc. CAV. pp. 622–640. LNCS 9206, Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_42

[3] Beyer, D., Dangl, M., Wendler, P.: A unifying view on SMT-based software verification. J. Autom. Reasoning **60**(3), 299–335 (2018). https://doi.org/10.1007/s10817-017-9432-6

[4] Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. J. Autom. Reasoning **69** (2025). https://doi.org/10.1007/s10817-024-09702-9, preprint: https://doi.org/10.48550/arXiv.2208.05046

[5] Beyer, D., Lingsch-Rosenfeld, M.: CPAchecker Validator 4.0 (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)

[6] Beyer, D., Löwe, S.: Explicit-state software model checking based on CEGAR and interpolation. In: Proc. FASE. pp. 146–162. LNCS 7793, Springer (2013). https://doi.org/10.1007/978-3-642-37057-1_11

[7] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643

[8] Friedberger, K.: CPA-BAM: Block-abstraction memoization with value analysis and predicate analysis (competition contribution). In: Proc. TACAS. pp. 912–915. LNCS 9636, Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_58

Theta

Modular, Highly Configurable, Automatic Model Checker

Competitions

- SV-COMP — Since 2022 Concurrency focus 3 configurations
- CHC-COMP — Since 2023 Gold medal (array)
- HWMCC — Planned AIGER (bit-level) BTOR2 (word-level)

Properties

- LTL properties
- (Timed) state reachability
- Signed integer overflow
- Array under- and overindexing
- Data races
- Dynamic memory safety
- Liveness (incl. termination)
- Concurrent reachability
- Petri Net props (deadlock, safety)
- Memory cleanup
- ...

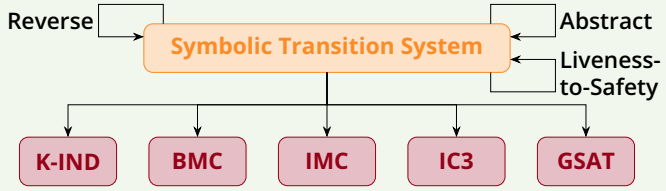
Portfolio

Dynamic Portfolio with **Algorithm Selection**

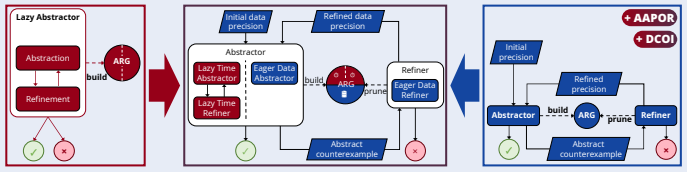
Push-button verification **without FM expertise**

Based on **theoretical** knowledge and **empirical** experience

Transition System Analyses



CEGAR: Combined Lazy and Eager Abstraction



Verification with Horn Clauses

Reachability:
Invariants by construction (location, loop, function)

Termination:
Recurrence sets and ranking functions

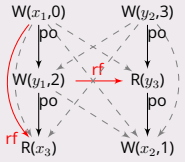
Multithreading:
Invariants for weakly consistent software (WiP)

Supported solvers: Z3, Eldarica, Golem, ...

Language Frontends

Timed Automata
LLVM IR
AIGER
CHC
PLC
Statecharts
Petri Nets
BTOR2

Ordering Consistency



- **Happens-before** partial orders
- **Memory-Model-Aware** Verification
- BMC via **SMT encoding**
- Supported decision procedures:
  - **Integer Difference Logic**
  - **Step-by-step refinement**
  - **Propagator-based consistency**

WSC (natively) **SC (by extension)** **MCA (by modification)**

Witness Generation

Traceable witnesses: Proofs and counterexamples

Software: Correctness and violation witnesses

Hardware: Certificates (WiP)

CHC: Models and counterexamples

Statecharts: Traces

Test generation: ConcurrentWitness2Test

Direct SMT encoding: termination

Witness Validation

Witness Generation & Validation

SMT Solvers

- **Native solvers** — Thoroughly tested, mature integration; Minimal overhead, diverse OS-support
- **JavaSMT** — Broad solver support; New API features (*ex*: User Propagator)
- **SMT-LIB** — Flexible: any solver, any version; Slower interaction

Industrial Projects

- **thyssenkrupp** — Verifying a **steer-by-wire** system design
- **CERN** — **PLCverif**: verifying safety interlocks for superconducting test benches
- **NASA JPL, IncQuery** — **Dynamic Verification Toolkit** (Model Checking as a Service)
- **Prolan** — **Railway interlocking system** verification

Collaborations

- SoSy-Lab — Summer sojourns; Active collaboration
- FBK — Seminar series; Integrated research
- CERN — PLCverif Summer students
- ADVANCE — V&V in Future CPS

Ongoing/Planned Extensions

- **CAR/DAR** — **Reimplement** successful algorithms in Theta
- **Saturation** — Extensions and generalization of the **saturation algorithm**
- **Liveness** — Advanced liveness-checking algorithms (`rlive`, `k-fair`)
- **ProbTheta** — **Probabilistic** Model Checking in Theta
- **Metasolver** — Online **SMT-solver-portfolio** to handle bugs and improve interpolants
- **ITP, Proof** — Better **interpolants** with CHC-solvers, and **proofs** instead of unsat cores

Code · Papers · Wiki · JavaDoc · Maven · Docker · Release · FmTools

Theta · Budapest, Hungary · Critical Systems Research Group · https://ftsrg.mit.bme.hu

SoSy-Lab
Software Systems

7th Competition on Software Testing

Dirk Beyer

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Features

Table 2: Technologies and features that the test generators used

Tester	Algorithm Selection	Bit-Precise Analysis	Bounded Model Checking	CEGAR	Concurrency Support	Evolutionary Algorithms	Explicit-Value Analysis	Floating-Point Arithmetics	Guidance by Coverage Measures	Portfolio	Predicate Abstraction	Random Execution	Symbolic Execution	Targeted Input Generation
CETFUZZ∅	✓					✓								
COVERITEST		✓		✓		✓	✓	✓		✓	✓	✓		
ESBMC-INCR		✓	✓		✓									
ESBMC-KIND		✓	✓		✓		✓	✓						
FDSE								✓	✓			✓	✓	
FIZZER		✓												
FUSEBMC			✓					✓	✓	✓				✓
FUSEBMC-AI∅			✓					✓	✓	✓				✓
HYBRIDTIGER∅				✓			✓	✓			✓			
KLEEF		✓						✓	✓				✓	✓
KLEE∅								✓					✓	✓
OWI∅		✓						✓				✓	✓	✓
PRTEST								✓				✓		
RIZZER∅		✓											✓	
SIKRAKEN new													✓	
SYMBIOTIC		✓		✓				✓	✓	✓			✓	✓
TRACERX			✓					✓					✓	✓
TRACERX-WP														
UTESTGEN				✓						✓				
WASP-C∅								✓				✓	✓	

Participants

Table 1: Competition candidates with tool references and representing jury members; new indicates first-time participants

Tester	License	Jury member	Affiliation
CETFUZZ∅	Apache	–	–
COVERITEST	Apache	M.-C. Jakobs	LMU Munich, Germany
ESBMC-INCR	Apache	C. Wei	U. of Manchester, UK
ESBMC-KIND	Apache	C. Wei	U. of Manchester, UK
FDSE	Apache	Z. Chen	National U. Defense Techn., China
FIZZER	Zlib	M. Trtík	Masaryk U., Brno, Czechia
FUSEBMC	MIT	K. Alshmrany	U. of Manchester, UK and Inst. Public Admin., Saudi Arabia
FUSEBMC-AI∅	MIT	–	–
HYBRIDTIGER∅	Apache	–	–
KLEEF	NCSA	A. Misonizhnik	Independent Researcher, Neutral
KLEE∅	NCSA	–	–
OWI∅	AGPL	–	–
PRTEST	Apache	T. Lemberger	LMU Munich, Germany
RIZZER∅	Zlib	–	–
SIKRAKEN new	LGPL	C. Meudec	South East Technological U., Ireland
SYMBIOTIC	MIT	M. Jonáš	Masaryk U., Brno, Czechia
TRACERX	Apache	J. Jaffar	National U. of Singapore, Singapore
TRACERX-WP	Apache	J. Jaffar	National U. of Singapore, Singapore
UTESTGEN	LGPL	M. Barth	LMU Munich, Germany
WASP-C∅	Apache	–	–
TESTCOV	Apache	M. Kettl	LMU Munich, Germany

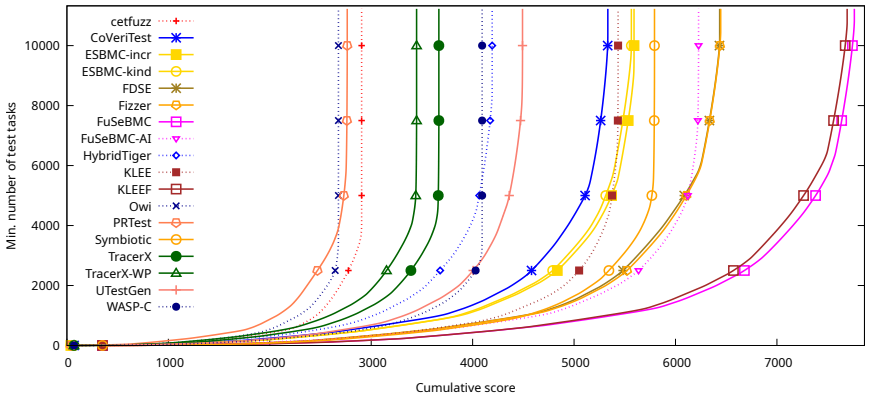
Results

Table 3: Quantitative overview over all results

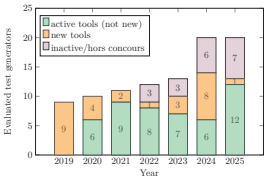
Participant	Cover-Error 1215 tasks	Cover-Branches 10011 tasks	Overall 11226 tasks
Cetfuzz∅	323	2524	2906
CoVeriTest	552	4959	5333
ESBMC-incr	679	4380	5591
ESBMC-kind	680	4323	5565
FDSE	729	**5468**	6435
Fizzer	736	5429	**6446**
FuSeBMC	**994**	**5656**	**7763**
FuSeBMC-AI∅	853	4077	6228
HybridTiger∅	438	3866	4193
KLEE∅	804	3065	5434
KLEEF	**969**	**5734**	**7692**
Owi∅	281	2462	2677
PRTest	211	3191	2764
Rizzer∅	608		
Sikraken new		2469	
Symbiotic	**743**	4207	5793
TracerX	390	3327	3667
TracerX-WP	349	3275	3447
UTestGen	439	4393	4492
WASP-C∅	554	2740	4094

Final Score

Figure 1: Quantile functions for category *Overall*.



Participation



Ranking

Table 4: Overview of the top-three test generators for each category (measurement values for CPU time in hours, rounded to two significant digits)

Rank	Tester	Score	CPU Time
Cover-Error			
1	FuSeBMC	**994**	75
2	KLEEF	969	9.5
3	Symbiotic	743	5.5
Cover-Branches			
1	KLEEF	**5734**	1500
2	FuSeBMC	5656	2500
3	FDSE	5468	2200
Overall			
1	FuSeBMC	**7763**	2600
2	KLEEF	7692	1500
3	Fizzer	6446	2100

Report

https://test-comp.sosy-lab.org/2025/

References

Reference

D. Beyer. Automatic testing of C programs: Test-Comp 2025. Springer, 2025

Funding

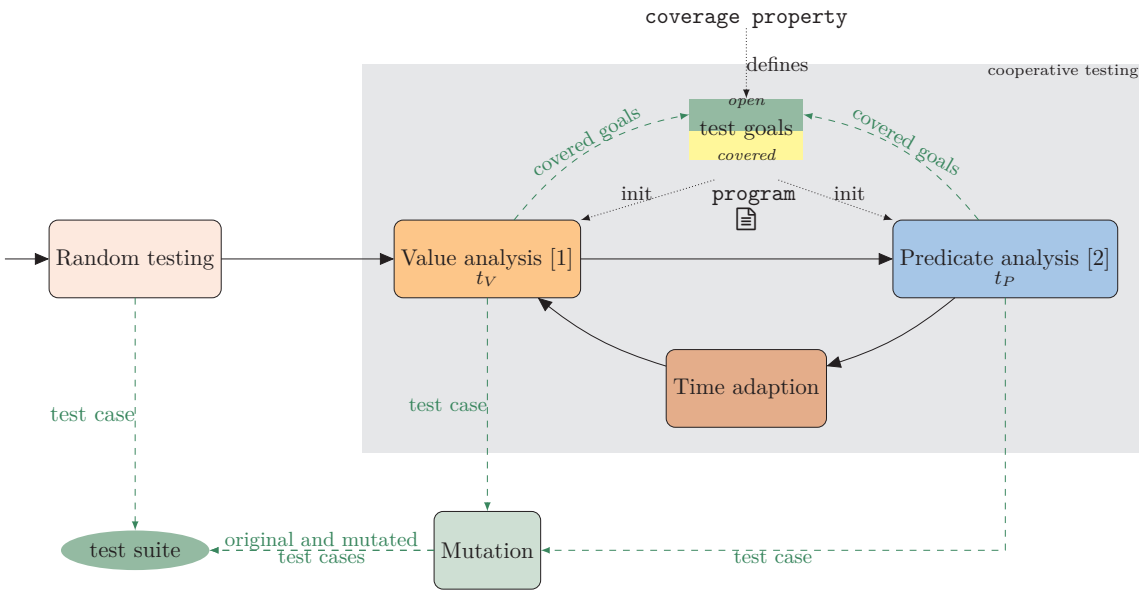
This project was funded in part by the Deutsche Forschungsgemeinschaft (DFG) — 418257054 (Coop).

CoVeriTest

Cooperative Verifier-Based Testing

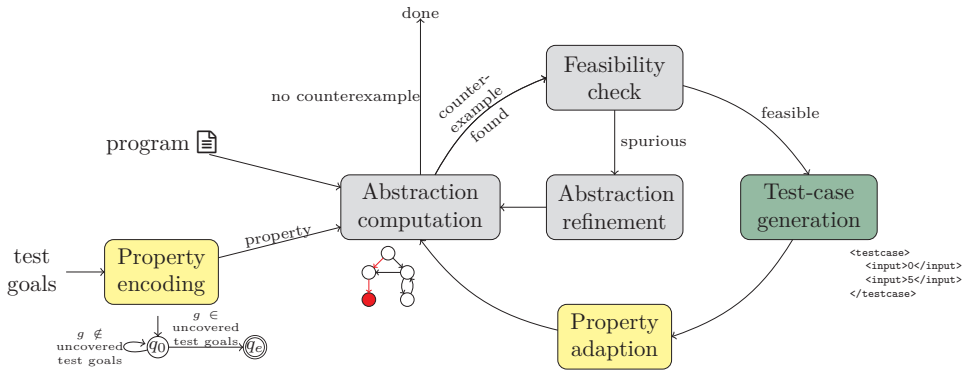
Marie-Christine Jakobs (m.jakobs@lmu.de)

Overview of CoVeriTest @Test-Comp



Implemented in CPA **https://cpachecker.sosy-lab.org/**

Test-Case Generation with CoVeriTest's Analyses



More about CoVeriTest

D. Beyer, M.-C. Jakobs, CoVeriTest: Cooperative Verifier-Based Testing, FASE, Springer, 2019.

D. Beyer, M.-C. Jakobs, Cooperative verifier-based testing with CoVeriTest, STTT 23(3), 2021.

M.-C. Jakobs, CoVeriTest with Dynamic Partitioning of the Iteration Time Limit (Competition Contribution), FASE, Springer, 2020.

Random Testing

Generation of 10 test cases with random inputs

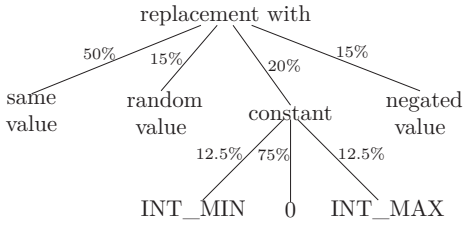
```
<testcase>
  <input>...</input>
    ...
  <input>...</input>
</testcase>
```

random length between 0 and 20

Each input is a a random integer from [0, 20]

Mutation of Test Cases

Per test case create 5 mutated test cases using the following mutation per input



Adaption of Time Limits

Idea

- Reward *past* behavior of analysis
- Adaption based on relative progress p_i, i.e., goals covered by analysis in round i in relation to all goals covered in round i
  $\Longrightarrow$ only adapt if new goals covered

$$\text{limit}_i^{\text{new}} = 10\,s + \frac{\frac{p_i}{\text{limit}_i}}{\frac{p_V}{\text{limit}_V} + \frac{p_P}{\text{limit}_P}} * 80\,s$$

Initial limits

- Value analysis 20 s
- Predicate analysis 80 s

References

[1] D. Beyer and S. Löwe. Explicit-state software model checking based on CEGAR and interpolation. In *Proc. FASE*, LNCS 7793, pages 146–162. Springer, 2013.
[2] D. Beyer, M. E. Keremoglu, and P. Wendler. Predicate abstraction with adjustable-block encoding. In *Proc. FMCAD*, pages 189–197. FMCAD, 2010.

TracerX: Enhancing Dynamic Symbolic Execution with Weakest Precondition & Deletion Interpolation

Arpita Dutta[1], Rasool Maghareh[2], Joxan Jaffar[3], Sangharatna Godboley[4], and Xiao Liang Yu[5]

{arpita, joxan, xiaoly}@comp.nus.edu.sg[1,3,5], sajjadrsm@gmail.com[2], sanghu@nitw.ac.in[4]

School of Computing, National University of Singapore, Singapore[1,3,5], Lemurian Labs, Toronto, Canada[2], National Institute of Technology Warangal, India[4]

TracerX Framework

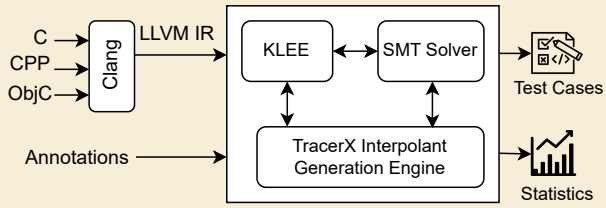


Figure 1. The overall architecture of TracerX

Highlights

- TracerX is Dynamic Symbolic Executor (DSE) which is built upon KLEE [1].
- The major advantage of DSE is its path-by-path exploration of the program execution space. However, this often leads to the path explosion problem.
- To address this issue, a method of abstraction learning has been used. The key step here is the computation of an interpolant to represent the learned abstraction [2].
- We use two different approaches of interpolant generation viz., 1. Deletion Interpolation [3] and 2. Weakest Precondition (WP) Interpolation [4]
- Deletion Interpolation (TracerX-Del) is our more stable and mature system and is briefly discussed in [3].
- Here, we present the Weakest Precondition (WP) Interpolation approach for TracerX, i.e., TracerX-WP.

From KLEE (No Interpolation) to TracerX (With Interpolation)

- Forward Symbolic Execution to find feasible paths (Similar to KLEE).
- Intermediate execution states preserved (Unlike KLEE).
- Half interpolant aka (PATH Interpolants) are generated during backward tracking and Full interpolants aka (TREE Interpolants) are generated by merging the half interpolants.
- Full interpolants used for subsumption at similar program points.
- TracerX uses information from already traversed subtree to prune other subtrees.

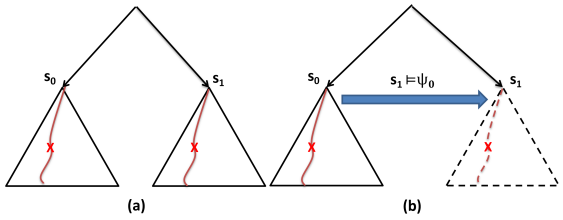
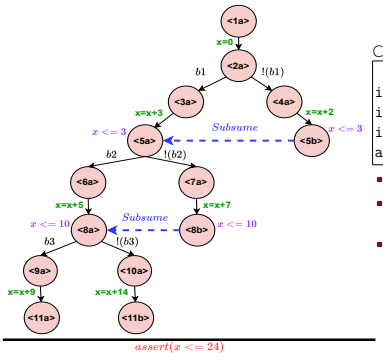


Figure 2. Exploration of Symbolic Execution Tree in Non-pruning DSE vs. Pruning DSE

Symbolic Execution Tree with Interpolation



Consider this program:

```
 x = 0;
if (b1) x += 3 else x += 2
if (b2) x += 5 else x += 7
if (b3) x += 9 else x += 14
assert(x <= 24)
```

- DFS traversal.
- **Without interpolation:** The full tree is traversed.
- **With interpolation:**
  1. $\langle 8b \rangle$ context contains $x = 10$. It is subsumed with the tree interpolant from $\langle 8a \rangle$: $x \leq 10$.
  2. $\langle 5b \rangle$ context contains $x = 2$. Subsumed with the tree interpolant from $\langle 5a \rangle$: $x \leq 3$.
  3. Big subtree traversal is avoided.

Interpolation: Weakest Precondition

- Ideal interpolant is the weakest precondition (WP) of the target. Unfortunately, WP is intractable to compute, which means it is difficult or impossible to find an exact solution.
- For example, in the above code snippet: assume $(not(b1 \wedge \neg b2 \wedge \neg b3))$.
- Hence, the WP before the first if-statement is:
  WP is: $b1 \longrightarrow (\neg b2 \wedge b3 \wedge x \leq 7) \vee (b2 \wedge x \leq 4)$
  $\neg b1 \longrightarrow x < 3$
- Essentially, WP is exponentially disjunctive. This means that any one of the conditions can be satisfied for the target to be reached. (As shown here)
- One way to approximate WP is to use a conjunctive approximation, which involves expressing the WP as a conjunction of simpler conditions (Challenge is to obtain a conjunctive approximation).

Approximation of Weakest Precondition

A Path is a sequence of assignments and assume instructions:

1. Interpolant of Assignment instruction:
   - $wp(inst, \omega) = \cdots$ inverse transition of $inst$ over ω
   - Implemented at LLVM IR level: LD/ST, add, sub, cmp, cast, GEP, etc.
   - e.g. $\omega : x \leq 15$ and $inst : x = z + 2$, then $wp(inst, \omega) : z \leq 13$
2. Interpolant of Assume instruction (C is incoming Context):
   $\{C\}$ assume(B) $\{\omega\}$
   - WP Approximation: find $\bar{C}$ to replace C
   - ABDUCTION PROBLEM !!!

Following algorithm is the heart of TracerX:

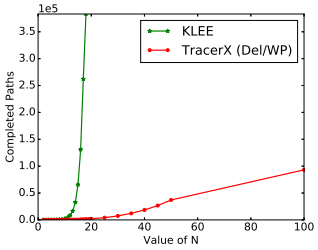
1. We compute finest partition so that $var(C_i) * var(C_j)$ $s.t.$ $i \neq j$: $\{C_1 * C_2 * C_3 * \ldots * C_n\}$ assume(B) $\{\omega_1 * \omega_2 * \omega_3 * \ldots * \omega_m\}$ ($*$ is as in separation logic).
2. Bunch C_i into three:

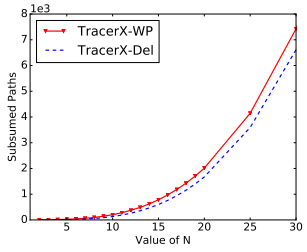
   - **Target independent:** The C_i which are separate from B and ω.
     Action: Replace C_i with $true$, i.e. remove C_i.
   - **Guard independent:** Consider $C_{gi} \equiv C_i$ s.t. $C_i * B$; and, $\omega_{gi} \equiv \omega_j$ s.t. $B * \omega_j$.
     Action: Replace C_{gi} by ω_{gi}.
   - **Remainder of the C_i:** We do not capture exact WP for this group.
     e.g. $\{z == 5\}$ assume$(x > z - 2)$ $\{x > 0\}$ (Here, $z > 2$ is the WP)
     Action: No change to C_i, i.e. keep C_i.

KLEE [1] v/s TracerX-Del [3] v/s TracerX-WP [4]

Consider this program; here, function `f(N)`, returns the sum of the first `N` natural numbers.

```
int counter=0;
char input[N+1];
klee_make_symbolic(input,(N+1)*
sizeof(char),"input");
for (int i=0;i<=N; i++){
    if(input[i]==B)
        counter=counter+1;
    else
        counter+=i;
}
klee_assert(counter!=f(N)+1);
```





- KLEE suffers from path explosion and hits the timeout for **N=17**.
- Both TracerX-Del and TracerX-WP scale up to **N=50** [**timeout=600 secs** for each tool]
- Clear gap in the number of subsumed paths between TracerX-Del and TracerX-WP.
- TracerX-WP generates better interpolants than TracerX-Del for subsumption.
- **Reason:** TracerX-Del generates interpolants as a subset of the incoming context whereas TracerX-WP generates interpolants from the weakest precondition of a path.

Experimental Results

Data set: All C-programs from RERS-2012 Challenge [6].

- **Total targets:** 1159
- All three systems **KLEE** [1], **CBMC** [5] and **TracerX-WP** [4] are run for 3600 seconds

Observations:

1. TracerX-WP able to detect 348 targets, while KLEE and CBMC are detected 245 and 117 targets respectively.
2. All the targets reached by TracerX-WP are super set of targets covered by CBMC and KLEE.
3. TracerX-WP is 29.59x faster than KLEE and 66.37x faster than CBMC.

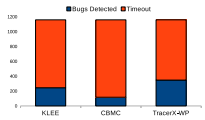


Figure 3. Targets detected by the tools

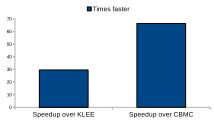


Figure 4. Speedup obtained by TracerX-WP

References

[1] C. Cadar et al. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: OSDI, 2008.

[2] J. Jaffar et al. TRACER: A symbolic execution tool for verification. In: CAV, 2012.

[3] J. Jaffar et al. TracerX: Dynamic symbolic execution with interpolation (competition contribution). In: FASE, 2020.

[4] A. Dutta et al. TracerX: Pruning Dynamic Symbolic Execution with Deletion and Weakest Precondition Interpolation (competition contribution). In: FASE, 2024.

[5] D. Kroening D et al. CBMC-C Bounded Model Checker. In: TACAS 2014.

[6] http://rers-challenge.org/