ETAPS

Joint Conference On Theory & Practice Of Software

ETAPS Poster Book

Collection of posters presented at ETAPS 2024

Luxembourg, 6–12 April 2024

ESOP 2024 Posters

Efficient Matching with Memoization for Regexes with Look-around and Atomic Grouping (ESOP'24)

Hiroya Fujinami[1,2], Ichiro Hasuo[1,2]
[1] National Institute of Informatics, Tokyo, Japan
[2] SOKENDAI (The Graduate University for Advanced Studies), Japan

ReDoS, efficient and extended regex matching

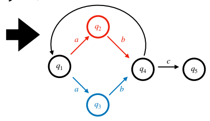
Catastrophic backtracking and ReDoS

ReDoS: a vulnerability by regex matching

- Depth-first matching can lead to catastrophic backtracking.
  Catastrophic backtracking is non-linear time backtracking.
- Catastrophic backtracking is the reason for ReDoS (Regular Expression Denial of Service).

For w = "ab^n" = "$ab\ldots ababab$",

`/(ab|ab)*c/`



"$ab\ldots ababab$" backtracking
"$ab\ldots ababab$" backtracking
"$ab\ldots ababab$" backtracking
"$ab\ldots ababab$" backtracking
⋮ 2^n cases...
"$ab\ldots ababab$" backtracking

Requirements for matching implementation

"Linear-time" and "easy to support extensions"

- A regex matching implementation we need is

Worst-case time complexity	To support extensions (look-around, atomic grouping)
linear $O(\|w\|)$	Easy

- Linear-time matching can be achieved by breadth-first matching.
- However, studies about extensions in breadth-first matching are few.
  In particular, atomic grouping has not been well studied.
- Then, we will introduce depth-first matching with memoization [Davis et al. S&P'21].

Memoization for regex matching

Memoization

Memoization and regex matching

- *Memoization* is a programming technique
  that makes recursive computations more efficient
  by recording arguments and the corresponding return values and reusing them.
- We can define a depth-first matching algorithm as a recursive function
  by the following signature (We show the entire definition later.)

$$\textsc{Match}_{\mathscr{A},w} \colon Q \times \mathbb{N} \to \{\mathsf{Failure}, \mathsf{Success}(i) \text{ for } i \in \mathbb{N}\}$$

- Therefore, we can apply memoization to depth-first matching.

Range of memoization tables

Efficient memoization

$$\textsc{Match}_{\mathscr{A},w} \colon Q \times \mathbb{N} \to \{\mathsf{Failure}, \mathsf{Success}(i) \text{ for } i \in \mathbb{N}\}$$

- The type of memoization tables for Match is naively

$$M \colon Q \times \mathbb{N} \rightharpoonup \{\mathsf{Failure}, \mathsf{Success}(i) \text{ for } i \in \mathbb{N}\}$$

- The previous study [Davis et al., S&P '21] shows that
  recording only failures is sufficient for (non-extended) regex.

$$M \colon Q \times \mathbb{N} \rightharpoonup \{\mathsf{Failure}\}$$

- However, this optimized memo. table type does not work with **extended** regex.
  We will show examples of that later.

Memoization for regex extensions (look-around and atomic grouping)

Look-around assertion

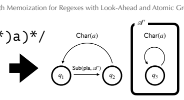
Look-around: a.k.a. zero-width assertion

- There are four kinds of look-around assertions.
  - `/(positive|negative) look-(ahead|behind)/`
- For simplicity, we only discuss positive look-ahead in discussions of look-around.
  Adaptation to other look-around operators, such as negative look-behind, is straightforward.

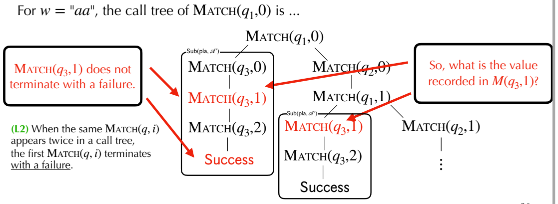
`/(?=r)/` positive look-ahead	`/(?!r)/` negative look-ahead
`/(?<=r)/` positive look-behind	`/(?<!r)/` negative look-behind

- Example:
  - `/abc(?=def)/.match("abcdef") == "abc"`
  - `/(?=abcdef)abc/.match("abcdef") == "abc"`

`/((?=a*)a)*/`



For w = "aa", the call tree of $\textsc{Match}(q_1,0)$ is ...

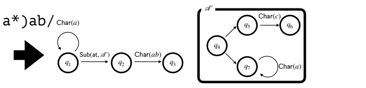


Atomic grouping

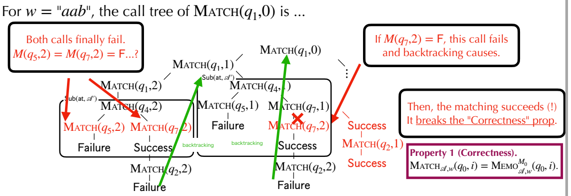
Atomic grouping: operator for backtracking

- Atomic grouping is an operator that controls backtracking.
  - This discards remaining backtrackings when the grouped regex is matched.
- Example: `/(?>r)/` atomic grouping
  - `/(?>a|ab)c/.match?("ac")`, not `match?("abc")`
  - not `/(?>a*)ab/.match?(w)` for `w = "ab", "aab", "aaab", ...`
- Typically, atomic grouping is used to prevent catastrophic backtracking.
  - However, we believe it is valuable to support this extension for legacy systems.

`/a*(?>c|a*)ab/`



For w = "aab", the call tree of $\textsc{Match}(q_1,0)$ is ...

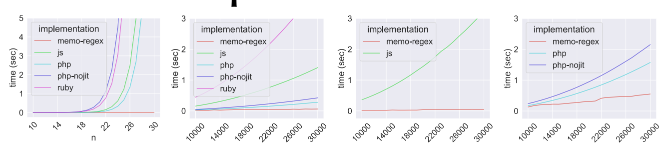


	Naive memoization	[Davis et al., S&P '21]	Our results
regex w/o extensions	$Q \times \mathbb{N} \rightharpoonup \{\mathsf{F}, \mathsf{S}(i) \text{ for } i \in \mathbb{N}\}$	$Q_{\mathrm{sel}} \times \mathbb{N} \rightharpoonup \{\mathsf{F}\}$	$Q \times \mathbb{N} \rightharpoonup \{\mathsf{F}\}$
w/ positive look-ahead	$Q \times \mathbb{N} \rightharpoonup \{\mathsf{F}, \mathsf{S}(i) \text{ for } i \in \mathbb{N}\}$	✗? $Q_{\mathrm{sel}} \times \mathbb{N} \rightharpoonup \{\mathsf{F}\}$ There is only informal discussion, and it is hard to interpret correctly.	Our contribution: $Q \times \mathbb{N} \rightharpoonup \{\mathsf{F}, \mathsf{S}\}$
w/ atomic grouping	$Q \times \mathbb{N} \rightharpoonup \{\mathsf{F}, \mathsf{S}(i) \text{ for } i \in \mathbb{N}\}$	✗	$Q \times \mathbb{N} \rightharpoonup \{\mathsf{F}(j) \text{ for } j \in \{0, \ldots, \nu(\mathscr{A})\}\}$ $\nu(\mathscr{A})$ is the maximum nesting depth of atomic groupings in $\mathscr{A}$. Typically, $\nu(\mathscr{A}) \le 1$.

Experiment results

Performance benchmark

Experiment results



(r1) (r2) (r3) (r4)

- Matching times grow $O(2^n)$ in (r1) and $O(n^2)$ in (r2-4) on other implementations,
  but on our implementation (`memo-regex`), they grow $O(n)$ in all regexes.
- Then, we can confirm that our algorithm works effectively for ReDoS vulnerable regexes.

A DENOTATIONAL APPROACH TO RELEASE/ACQUIRE CONCURRENCY

Authors: **Yotam Dvir, Ohad Kammar, Ori Lahav**

Moggi semantics — effects denote monads — [Moggi 1991]

+ [BHN 2016]

Brookes semantics — traces denote behaviors — [Brookes 1996]

+ [JPR 2012] For TSO

Relaxed memory — weakly consistent concurrent shared state

GOAL Moggi-style Brookes semantics for the Release/Acquire relaxed memory model

NEW CHALLENGES ABOUND

- Linear traces for a decentralized model
- More abstract and nuanced traces
- More closure rules
- First-class parallelism with causal propagation

Monad-based Denotational Semantics [Moggi 1991]

Modular framework for effectful semantics

$$[\![(\mathtt{k} := 1\,;\mathtt{m} := 1) \parallel \langle \mathtt{m}?, \mathtt{k}? \rangle]\!] = [\![\mathtt{k} := 1\,;\mathtt{m} := 1]\!] \mathbin{|||} [\![\langle \mathtt{m}?, \mathtt{k}? \rangle]\!]$$

compositionality: homomorphic semantics; sequencing denotes monadic bind

$$= ([\![\mathtt{k} := 1]\!] \gg\!= \lambda\langle\rangle.\, [\![\mathtt{m} := 1]\!]) \mathbin{|||} ([\![\mathtt{m}?]\!] \gg\!= \lambda v_{\mathtt{m}}.\, [\![\mathtt{k}?]\!] \gg\!= \lambda v_{\mathtt{k}}.\, \langle v_{\mathtt{m}}, v_{\mathtt{k}} \rangle)$$

Built-in: higher-order functions & **structural reasoning**, e.g.

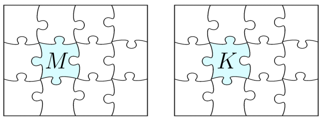
K effect-free $\Longrightarrow [\![\mathbf{if}\, K\, \mathbf{then}\, (M\,;N)\, \mathbf{else}\, (M\,;N')]\!] = [\![M\,; \mathbf{if}\, K\, \mathbf{then}\, N\, \mathbf{else}\, N']\!]$

Adequacy

$$[\![M]\!] \geq [\![K]\!] \implies M \twoheadrightarrow K$$

Abstraction

$$M \overset{\checkmark}{\twoheadrightarrow} K \overset{?}{\implies} [\![M]\!] \geq [\![K]\!]$$

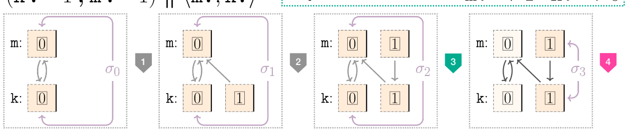


Release/Acquire Interleaving Semantics [KHLVD 2017]

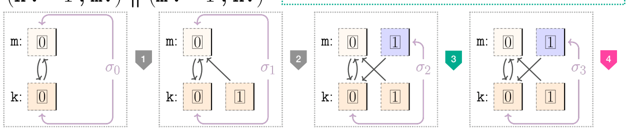
Fragment of the C/C++ model of causal propagation

Memory: msgs on timelines | **View:** accessible memory | **Threads** store/load views

$(\mathtt{k} := 1\,;\mathtt{m} := 1) \parallel \langle \mathtt{m}?, \mathtt{k}? \rangle$ — **Impossible** outcome: $\mathtt{m}? \mapsto 1 \quad \mathtt{k}? \mapsto 0$



$(\mathtt{k} := 1\,;\mathtt{m}?) \parallel (\mathtt{m} := 1\,;\mathtt{k}?)$ — **Possible** outcome: $\mathtt{m}? \mapsto 0 \quad \mathtt{k}? \mapsto 0$

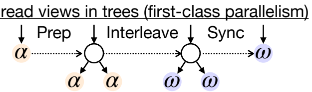


RA state invariants, e.g.

view σ point to msg $\nu \Longrightarrow \nu.\mathrm{view} \leq \sigma$

Admissible step: ADVANCE (pretend to load)

Thread views in trees (first-class parallelism)



JUSTIFIED TRANSFORMATIONS

Laws of Parallel Programming			
Symmetry	$M \parallel N$	$\twoheadrightarrow$	$\mathbf{swap}\ (N \parallel M)$
Generalized Sequencing	$(\mathbf{let}\ a = M_1\ \mathbf{in}\ M_2) \parallel (\mathbf{let}\ b = N_1\ \mathbf{in}\ N_2)$	$\twoheadrightarrow$	$\mathbf{match}\ M_1 \parallel N_1\ \mathbf{with}\ \langle a, b\rangle.\, M_2 \parallel N_2$
Eliminations			
Irrelevant Read	$\ell?\,;\langle\rangle$	$\twoheadrightarrow$	$\langle\rangle$
Write-Write	$\ell := v\,;\ell := w$	$\overset{\mathsf{Ab}}{\twoheadrightarrow}$	$\ell := w$
Write-Read	$\ell := v\,;\ell?$	$\twoheadrightarrow$	$\ell := v\,;v$
Write-FAA	$\ell := v\,;\mathrm{FAA}\,(\ell, w)$	$\overset{\mathsf{Ab}}{\twoheadrightarrow}$	$\ell := (v + w)\,;v$
Read-Write	$\mathbf{let}\ a = \ell?\ \mathbf{in}\ \ell := (a + v)\,;a$	$\twoheadrightarrow$	$\mathrm{FAA}\,(\ell, v)$
Read-Read	$\langle \ell?, \ell? \rangle$	$\twoheadrightarrow$	$\mathbf{let}\ a = \ell?\ \mathbf{in}\ \langle a, a\rangle$
Read-FAA	$\langle \ell?, \mathrm{FAA}\,(\ell, v)\rangle$	$\twoheadrightarrow$	$\mathbf{let}\ a = \mathrm{FAA}\,(\ell, v)\ \mathbf{in}\ \langle a, a\rangle$
FAA-Read	$\langle \mathrm{FAA}\,(\ell, v), \ell? \rangle$	$\twoheadrightarrow$	$\mathbf{let}\ a = \mathrm{FAA}\,(\ell, v)\ \mathbf{in}\ \langle a, a + v\rangle$
FAA-FAA	$\langle \mathrm{FAA}\,(\ell, v), \mathrm{FAA}\,(\ell, w)\rangle$	$\overset{\mathsf{Ab}}{\twoheadrightarrow}$	$\mathbf{let}\ a = \mathrm{FAA}\,(\ell, v + w)\ \mathbf{in}\ \langle a, a + v\rangle$
Others			
Irrelevant Read Introduction	$\langle\rangle$	$\twoheadrightarrow$	$\ell?\,;\langle\rangle$
Read to FAA	$\ell?$	$\overset{\mathsf{Di}}{\twoheadrightarrow}$	$\mathrm{FAA}\,(\ell, 0)$
Write-Read Deorder	$\langle (\ell := v), \ell'? \rangle$	$\overset{\mathsf{Ti}}{\twoheadrightarrow}$	$(\ell := v) \parallel \ell'?$ $\quad (\ell \neq \ell')$
Write-Read Reorder	$(\ell := v)\,;\ell'?$	$\overset{\mathsf{Ti}}{\twoheadrightarrow}$	$\mathbf{fst}\ \langle \ell'?, (\ell := v)\rangle$ $\quad (\ell \neq \ell')$

Trace-based Denotational Semantics [Brookes 1996]

Sequences of guarantees to/from the environment

Transition $\langle \mu, \varrho \rangle$ — Rely on memory; Guarantee memory

Trace $\alpha\,\boxed{\xi}\,\omega \mathbin{.\!\cdot\!.} r$ — Sequence of transitions; Guarantee to return; Rely on accessibility; Guarantee accessibility

Denotations

Sets of traces closed under REWRITE RULES

$$\tau \in [\![M]\!] \quad \tau \xrightarrow{\star} \pi \implies \pi \in [\![M]\!]$$

$$\star \in \{\mathsf{St}, \mathsf{Mu}, \mathsf{Rw}, \mathsf{Fw}, \mathsf{Ti}, \mathsf{Ab}, \mathsf{Di}\}$$

Composition

Sequential

$$\frac{(\alpha\,\boxed{\xi}\,\sigma \mathbin{.\!\cdot\!.} r) \in P \qquad (\sigma\,\boxed{\eta}\,\omega \mathbin{.\!\cdot\!.} s) \in f(r)}{(\alpha\,\boxed{\xi\eta}\,\omega \mathbin{.\!\cdot\!.} s) \in P \gg\!= f}$$

Parallel

$$\frac{(\alpha\,\boxed{\xi_1}\,\omega \mathbin{.\!\cdot\!.} r_1) \in P_1 \qquad (\alpha\,\boxed{\xi_2}\,\omega \mathbin{.\!\cdot\!.} r_2) \in P_2 \qquad \xi \in \xi_1 \parallel \xi_2}{(\alpha\,\boxed{\xi}\,\omega \mathbin{.\!\cdot\!.} \langle r_1, r_2\rangle) \in P_1 \mathbin{|||} P_2}$$

REWRITE RULES

4 CONCRETE

Rewind (Rw) strengthens reliance on initial accessibility

Forward (Fw) weakens the guarantee of final accessibility

Stutter (St) propagates reliance as a guarantee

$$\xi\eta \xrightarrow{\mathsf{St}} \xi\langle\mu, \mu\rangle\eta$$

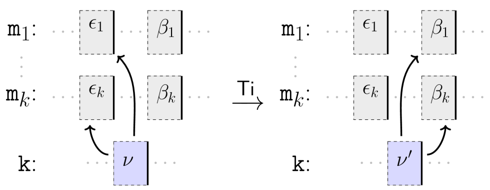
$$\xi\langle\mu, \varrho\rangle\langle\varrho, \theta\rangle\eta \xrightarrow{\mathsf{Mu}} \xi\langle\mu, \theta\rangle\eta$$

Mumble (Mu) omits a guarantee and relies on it internally

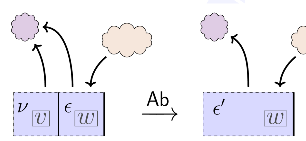
With only the CONCRETE rules the traces have an operational interpretation

3 ABSTRACT

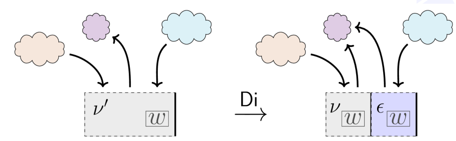
Challenge: non-operational traces **Solution:** percolate ABSTRACT rewrites out to induct



Tighten (Ti) guarantee less accessibility after reading



Absorb (Ab) guarantee fewer messages



Dilute (Di) touching value-view equals are indistinguishable

MAIN RESULTS

Moggi + Brookes + RA

A denotational semantics for Release/Acquire based on linear traces that is:

- **Standard** (monad base, truly compositional)
- **Adequate** (refinements are sound)
- **Abstract** (supports known transformations)

[Moggi 1991] Moggi, E.: Notions of computation and monads, Inf. Comput. [Brookes 1996] Brookes, S.D.: Full abstraction for a shared-variable parallel language, Inf. Comput. [JPR 2012] Jagadeesan, R., Petri, G., Riely, J.: Brookes is relaxed, almost!, FOSSACS
[KHLVD 2017] Kang, J., Hur, C., Lahav, O., Vafeiadis, V., Dreyer, D.: A promising semantics for relaxed-memory concurrency, POPL [BHN 2016] Benton, N., Hofmann, M., Nigam, V.: Effect-dependent transformations for concurrent programs, PPDP

yotamdvir@mail.tau.ac.il | yotamdvir.github.io

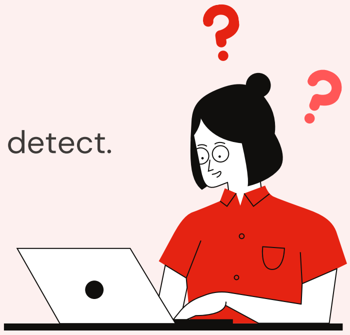
FASE 2024 Posters

VerCors

Verification of Concurrent and Distributed Software

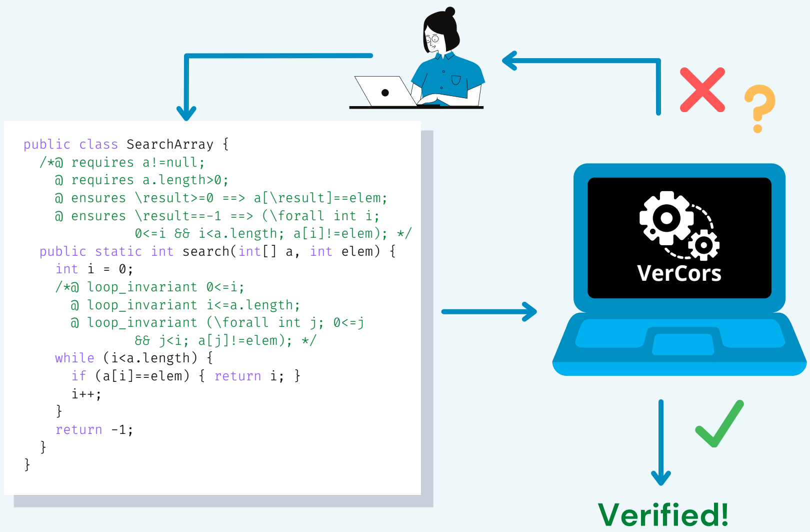
Problem

Concurrency in systems can cause subtle bugs that are difficult to detect.
As a result, concurrent systems are notoriously difficult to build.
To help build **correct software**, we develop **VerCors**, a tool for the verification of concurrent and distributed software.



How does it work?

- **Specification** describes the intended behaviour of the system
- The user provides the **program code** and **specifications** to VerCors
- **VerCors** determines whether the program is correct w.r.t. the specification using logical inference
- VerCors supports **multiple languages** including Java, C, CUDA and OpenCL!




```java
public class SearchArray {
  /*@ requires a!=null;
    @ requires a.length>0;
    @ ensures \result>=0 ==> a[\result]==elem;
    @ ensures \result==-1 ==> (\forall int i;
              0<=i && i<a.length; a[i]!=elem); */
  public static int search(int[] a, int elem) {
    int i = 0;
    /*@ loop_invariant 0<=i;
      @ loop_invariant i<=a.length;
      @ loop_invariant (\forall int j; 0<=j
              && j<i; a[j]!=elem); */
    while (i<a.length) {
      if (a[i]==elem) { return i; }
      i++;
    }
    return -1;
  }
}
```

Achievements

- Verified **Parallel Nested DFS**, an important verification algorithm
- Case study with **Technolution** to detect bugs in their tunnel control software
- **VeyMont**: Given a verified program, generate a correct parallelised version
- **Alpinist**: Automatic transformation of specifications for GPU optimisations
- **VeSUV**: Automatic encoding of embedded systems designs written in SystemC into PVL

What's next?

- Extend LLVM verification support with the **Pallas** project
- Generate specifications
- Apply VerCors to embedded & industrial systems
- Improve usability and scalability of the approach

To **your** project?

Want more? Scan me!

utwente.nl/vercors

Current collaborators
Marieke Huisman (Project lead), Lukas Armborst, Petra van den Bos, Pieter Bos, Paula Herber, Robert Mensing, Robert Rubbens, Alexander Stekelenburg, Ömer Şakar, Philip Tasche

Funding projects
Mercedes — MAXIMAL RELIABILITY OF CONCURRENT AND DISTRIBUTED SOFTWARE
CHEOPS — VERIFIED CONSTRUCTION OF CORRECT AND OPTIMISED PARALLEL SOFTWARE
SAVES
Pallas
CARP
VerDi

UNIVERSITY OF TWENTE.

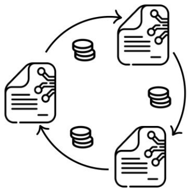
Monitoring the future of Smart Contracts

Margarita Capretto, Martín Ceresa, and César Sánchez

Contact: **firstname.lastname@imdea.org**

Smart Contracts

- programs running on blockchains.
- govern exchange of cryptocurrency.
- interact with other smart contracts.
- cannot be modified.
- their effects are permanent.



Runtime Verification

- Monitors enforce desired transaction properties (*P*).
- If *P* fails, the monitor fails the whole transaction. ✗
- If *P* holds, the transaction commits. ✓
- Improve the reliability of smart contracts.

Problem

Transactions can depend on future transactions.
Current blockchains force immediate decision (✗or ✓).

Solution: Future monitors

- State properties across multiple transactions.
- Delay transactions consolidation.

Monitors Hierarchy

Present	Future
Global Monitors*	Global future monitors*
Multicontract monitors*	Multicontract future monitors*
Transaction monitors^	Future monitors [this work]
Operation Monitors^	

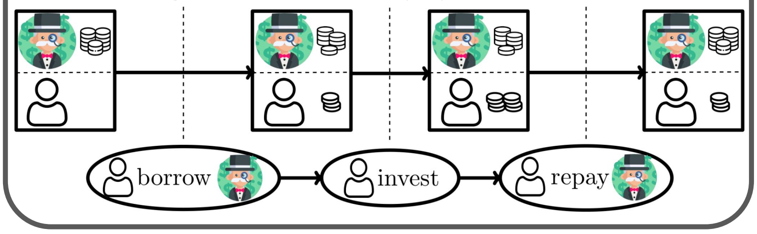
*Future work ^Previous work

Example: Multitransaction Flash Loan

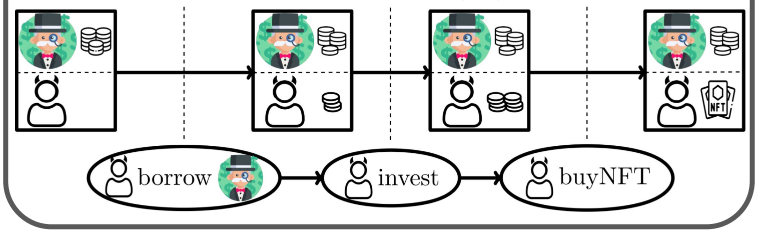
Safety: A loan is repaid to the lender within 2 transactions. **Progress:** lenders always grant loans.

Current blockchains

Case 1: good client repays the loan on time.



Case 2: bad client does not repay the loan on time.



By the end of the borrowing transaction, the lender does not know in which case it is but must make a decision.

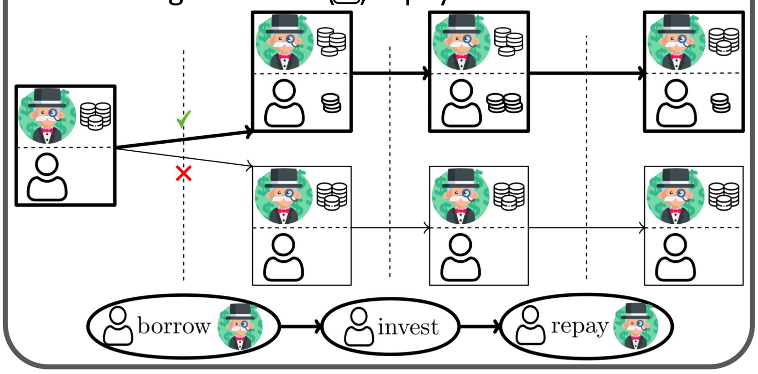
Commit it and trust the client ✓ — Violates safety

✗ Preemptively fail it — Violates progress

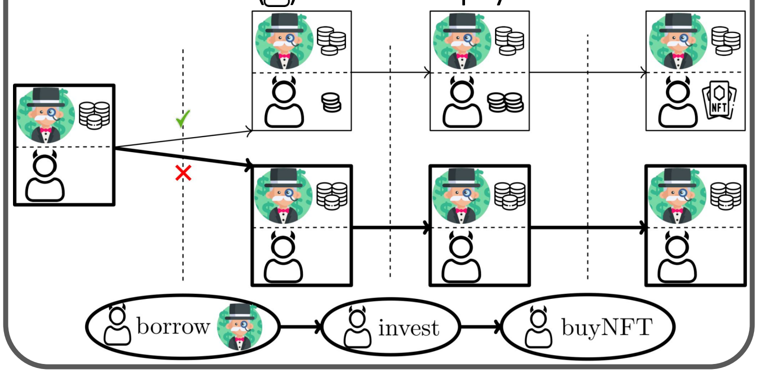
Future monitors

Delay transactions consolidation ⟶ branch the execution ⟶ allow differentiating the two cases.

Case 1: good client repays the loan on time.



Case 2: bad client does not repay the loan on time.



Loan is repaid ⟶ borrowing transaction *commits* ✓
⟶ top branch consolidates.

Loan is not repaid ⟶ borrowing transaction *fails* ✗
⟶ bottom branch consolidates.

FoSSaCS 2024 Posters

A Resolution-Based Interactive Proof System for UNSAT

Philipp Czerner, Valentin Krasotin, Javier Esparza
{czerner, krasotin, esparza}@in.tum.de
Technical University of Munich



link to paper

Efficient Certification for UNSAT

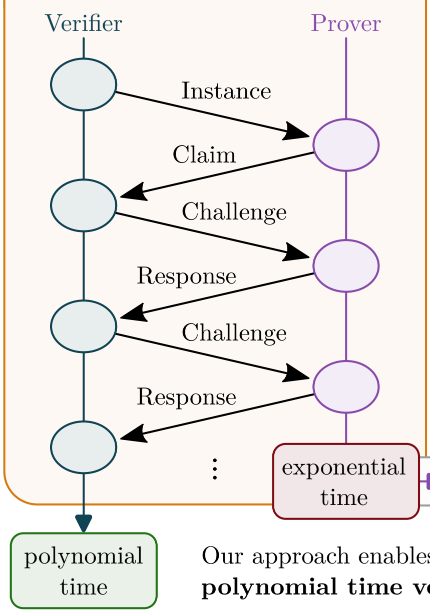
Full verification (proof of correctness for all inputs) is impractical for state-of-the-art SAT solvers. Certification instead checks the output as it is being produced. To be practical, the certificate checker must be efficient. Polynomially-sized non-interactive certificates do not exist for problems outside NP. For UNSAT, extended resolution proofs are used in practice. However, these can be exponentially long w.r.t. the input.

Goal: Fast Certification via IP = PSPACE

The famous IP = PSPACE breakthrough in complexity theory [1,2] proves existence of efficient (i.e. polynomial-time) certification through interactive protocols (IPs) for any PSPACE problem, e.g. for UNSAT. But their algorithm to generate the interactive certificates is impractical. We try to adapt existing decision procedures in automated reasoning to also generate interactive certificates. The overhead of the interactive protocol must be bounded, compared to just executing the decision procedure.

Interactive Protocols

Polynomial Verifier checks claims of unbounded, but untrusted, Prover



Our approach enables certification with **polynomial time verification cost**

Davis-Putnam Procedure [3]

A decision procedure for SAT:

1. Pick a variable x
2. Add all resolvents w.r.t. x
3. Remove all clauses with x or $\neg x$

$\bigwedge_i (x \vee a_i) \wedge \bigwedge_j (\neg x \vee b_j) \wedge c$ —pick x→ $\bigwedge_{i,j} (a_i \vee b_j) \wedge c$

Competitive IP

An IP is *competitive* with an algorithm A if

$$\frac{\mathsf{time}(\mathrm{IP}, x)}{\mathsf{time}(A, x)} \in \mathcal{O}(\mathrm{poly}\,|x|) \quad \forall \text{inputs } x$$

Intuitively, instances that are practical to solve with A can be practically certified with the IP.

Exploiting Arithmetisation

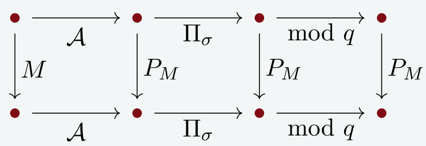
Arithmetisation is a fundemental technique for designing IPs. The idea is to assign a polynomial to each formula that extends its binary behaviour.

$\mathsf{true} \to 1$	$\varphi_1 \wedge \varphi_2 \to p_1 \cdot p_2$
$\mathsf{false} \to 0$	$\varphi_1 \vee \varphi_2 \to p_1 + p_2 - p_1 p_2$
$x \to x$	$\neg x \to 1 - x$

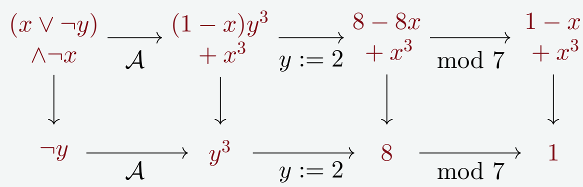
Prior IPs use a straightforward arithmetisation, e.g. the one shown above. However, it is unclear how to apply it to the Davis-Putnam Procedure.
Instead, we construct a competitive IP using a non-standard arithmetisation:

$\mathsf{true} \to 0$	$\varphi_1 \wedge \varphi_2 \to p_1 + p_2$
$\mathsf{false} \to 1$	$\varphi_1 \vee \varphi_2 \to p_1 \cdot p_2$
$x \to 1 - x$	$\neg x \to x^3$

An arithmetisation $\mathcal{A}$ is *compatible* with a macrostep algorithm if for every macrostep M there is a corresponding mapping on polynomials P_M.



The mapping P_M must additionally commute with partial evaluation Π_σ and remainder w.r.t. a prime q.



Here, $P_M(p) = p[x/0] \cdot p[x/1]$ works, but it fails for clauses without x. We use $P_M(a_3x^3 + a_1x + a_0) = -a_3a_1 + a_1 + a_0$ instead, which works in general.

A Framework for Competitive IPs

We give a theoretical framework to construct competitive IPs for certain classes of UNSAT algorithms. This framework gives sufficient conditions that an arithmetisation is compatible with an algorithm. Given a compatible arithmetisation, we construct a competitive IP in a generic fashion.

A *macrostep algorithm* transforms the formula by applying a polynomial number of *macrosteps.*

$$\varphi \xrightarrow{M_1} \varphi' \xrightarrow{M_2} \varphi'' \longrightarrow \cdots \longrightarrow \mathsf{false}$$

Each step maps the formula, s.t. $\varphi \equiv \varphi' \equiv \ldots \equiv \mathsf{false}$

Open Questions

- Implement further optimisations within this framework
- Adapt different decision procedures (e.g. DPLL)
- Exploit cryptographic assumptions
- Use multiple provers to certify resolution proofs directly

[1] Lund, Fortnow, Karloff, Nisan, 1990
[2] Shamir, 1992 [3] Davis, Putnam, 1960

Tighter Construction of Tight Büchi Automata

Marek Jankola
marek.jankola@sosy.ifi.lmu.de
LMU Munich, Munich, Germany

Jan Strejček
strejcek@fi.muni.cz
Masaryk University, Brno, Czechia

Motivation

Tight automata are useful in

- LTL model checking for shortest counterexamples
- LTL synthesis for maximally satisfying strategies

Previous constructions [4, 3] of tight *Büchi automata (BA)* from Büchi automata have large raise of states in the worst case and there is a big gap between the lower and the upper bound. In the following, n is the number of states of an input automaton.

$$2^{\Omega(n)} \longleftrightarrow \mathcal{O}((\sqrt{2}n)^{2n})$$

Preliminaries

- Lasso-shaped word $u = vw^\omega$ is an infinite word composed from a finite prefix (stem) - v and from infinite repetition of a finite word (loop) - w.
- Each lasso-shaped word has infinitely many stems and loops, we define $|minSL(u)| = min\{|vw| \mid u = vw^\omega\}$.

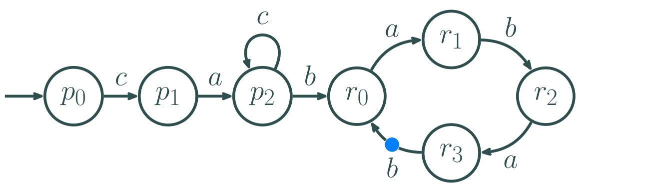
Example

$u = cb(abab)^\omega = c(ba)^\omega \Rightarrow |minSL(u)| = |c| + |ba| = 3$

- *Transition-based Büchi automaton (TBA)* is a type of ω-automaton that contains a set of accepting transitions (we depict them with the blue mark ●) and accepts an infinite word if there is a run (a sequence of transitions) over the word that passes an accepting transition infinitely often.

Definition: Tight Transition-Based Büchi Automata

A TBA $\mathcal{A}$ is *tight* iff for each lasso-shaped word $u \in L(A)$ there exists an accepting lasso-shaped run ρ satisfying $|minSL(u)| = |minSL(\rho)|$.



TBA $\mathcal{A}$: not tight

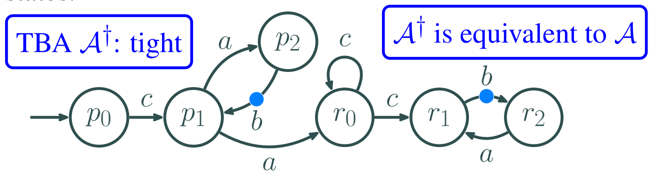
$\rho = p_0 \xrightarrow{c} p_1 \xrightarrow{a} p_2 \xrightarrow{b} (r_0 \xrightarrow{a} r_1 \xrightarrow{b} r_2 \xrightarrow{a} r_3 \xrightarrow{b} r_0)^\omega$

$|minSL(\rho)| = 7 \neq 3 = |minSL(c(ab)^\omega)|$

Main Results

We prove the following theorems:

- **Upper Bound:** For each TBA with n states, we can construct an equivalent tight TBA with at most $\mathcal{O}(n! \cdot n^3)$ states.



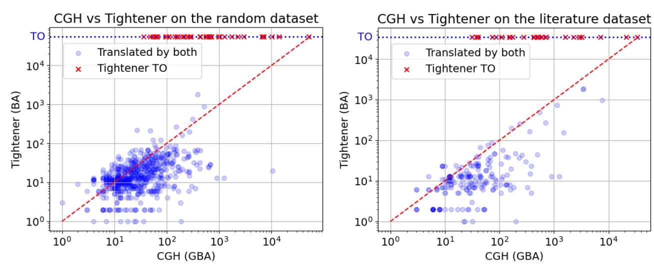
- **Tight TBA→ Tight BA:** For each tight TBA with n states, we can construct an equivalent tight BA with at most $2n$ states.
- **Lower Bound:** For each $n > 0$, there is a BA with $2n+1$ states such that every equivalent tight TBA has at least $\sum_{k=1}^{n} \frac{n!}{(n-k)!}$ states.
- **New Boundaries:** The resulting new boundaries for tight Büchi automata

$$2^{\Omega(n)} \prec \Omega\left(\frac{n-1}{2}!\right) \longleftrightarrow \mathcal{O}(n! \cdot n^3) \prec \mathcal{O}((\sqrt{2}n)^{2n})$$

- **Practical reductions:** Let $\mathcal{A}$ be a tight TBA and let $\sqsubseteq$ be a *good for quotienting* [1] preorder. The reduced automaton $\mathcal{A}/\sqsubseteq$ is tight.

Implementation and Evaluation

Comparison of our tool Tightener against the only known implemented algorithm CGH [4] that constructs tight Büchi automata from LTL formulas (TO=timeout). Tightener uses Spot [2] to obtain a TBA from LTL formula. We measure the number of states of the resulting automata.



References

[1] L. Clemente et al. "Efficient reduction of nondeterministic automata with application to language inclusion testing". In: *Log. Methods Comput. Sci.* 15.1 (2019).

[2] A. Duret-Lutz et al. "From Spot 2.0 to Spot 2.10: What's New?" In: *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II*. Ed. by S. Shoham et al. Vol. 13372. Lecture Notes in Computer Science. Springer, 2022, pp. 174–187.

[3] R. Ehlers. "How Hard Is Finding Shortest Counter-Example Lassos in Model Checking?" In: ed. by M. H. ter Beek et al. Vol. 11800. Springer, 2019, pp. 245–261.

[4] V. Schuppan. "Liveness checking as safety checking to find shortest counterexamples to linear time properties". PhD thesis. ETH Zurich, 2006.

M. Jankola was supported by the Deutsche Forschungsgemeinschaft (DFG) – 378803395 (ConVeY) and J. Strejček by Czech Science Foundation grant GA23-06506S

Checking History-Determinism is NP-hard for Parity Automata

Aditya Prakash
University of Warwick, UK
aditya.prakash@warwick.ac.uk

1 History-deterministic parity automata

History-determinism. Automata where nondeterminism can be resolved based on the prefix read so far. Equivalently, if Eve wins the corresponding HD game, proceeding in infinitely many rounds. In round i:

- Adam selects letter a_i
- Eve selects transition $q_i \xrightarrow{a_i} q_{i+1}$

Eve's winning condition: Eve's run is accepting if Adam's word is accepting.

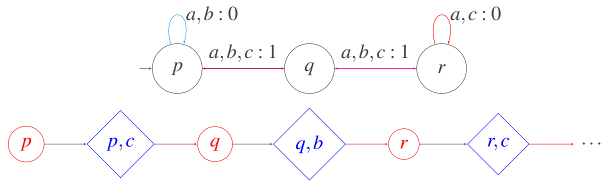


Figure 1: An HD Büchi automaton [3] and a play of history-determinism game on it. Eve's winning strategy in the HD game is to alternate between picking left and right transitions at the state q.

2 History of complexity of checking history-determinism

- Henzinger and Piterman, 2006 [5] - Can be decided in EXPTIME
- Kuperberg and Skrzypczak, 2015 [6] - As hard as solving Parity games, PTIME for coBüchi automata
- Bagnol and Kuperberg, 2018 [1] - PTIME for Büchi automata

3 NP-hardness: reduction from 2-dimensional parity games

2-D parity games: A game arena with each edge labelled by two natural numbers, forming two parity conditions χ_1 and χ_2.
Eve's winning condition: If the χ_1 parity condition is satisfied, then the χ_2 parity condition is satisfied.

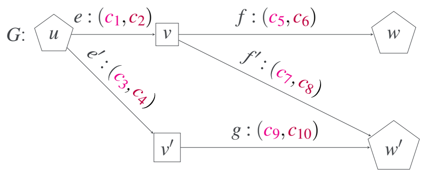


Figure 2: A snippet of a 2-D parity game. The pentagons represent Adam's vertices and the squares represent Eve's vertices.

Chatterjee, Henzinger and Piterman [4] have shown that deciding if Eve wins a 2-D parity game is NP-hard.

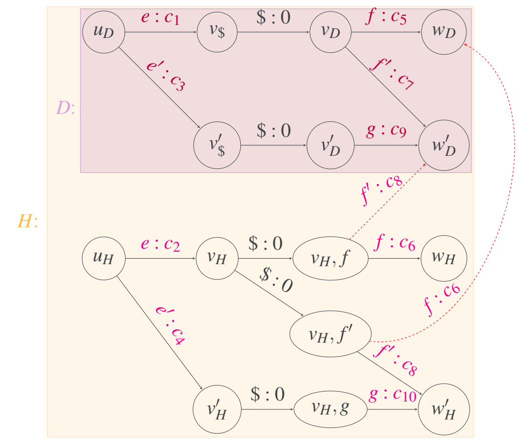


Figure 3: Reduction to simulation and checking history-determinism from 2-D parity game.

- H simulates D if and only if Eve wins G.

Theorem 1. *Deciding simulation between two parity automata is NP-complete.*
Good 2-D parity games: all paths that satisfy the χ_2 parity condition also satisfy χ_1. Deciding if Eve wins a good 2-D parity game is NP-complete.

- H is history-deterministic if and only if Eve wins G.

Theorem 2. *Checking history-determinism is NP-hard for parity automata.*

4 History-deterministic automata for model checking

Language inclusion: Given parity automata A and B, is $L(A) \subseteq L(B)$?
Deciding language inclusion is PSPACE-complete for nondeterministic parity automata.
If B is HD, however, $L(A) \subseteq L(B)$ if and only if B simulates A [5].
Simulation game. The simulation game of A and B proceeds in infinitely many rounds. In round i:

- Adam selects letter a_i
- Adam selects transition $p_i \xrightarrow{a_i} p_{i+1}$ in A.
- Eve selects a transition $q_i \xrightarrow{a_i} q_{i+1}$ in B.

Eve's winning condition: if Adam's run in A is accepting, Eve's run in B is accepting as well.

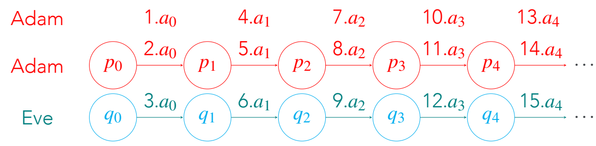


Figure 4: Order of moves in a simulation game. At the end of an infinite play, Adam constructs a word and run on that word, and Eve constructs a run on the same word as well.

Deciding simulation between two parity automata is in NP, but we show that we can do even better if B is history-deterministic.
Theorem 3. *Given nondeterministic parity automaton A and HD parity automaton B, checking if $L(A) \subseteq L(B)$ can be decided in quasi-polynomial time.*

5 Open: how hard is recognising HD parity automata?

For a Büchi or coBüchi automaton, one can decide history-determinism in PTIME by solving the 2-token game.
2-token game. Played between Eve and Adam, and proceeds in infinitely many rounds. In round i:

- Adam selects letter a_i
- Eve selects a transition $q_i \xrightarrow{a_i} q_{i+1}$
- Adam selects two transitions $p^1_i \xrightarrow{a_i} p^1_{i+1}$, $p^2_i \xrightarrow{a_i} p^2_{i+1}$.

Eve's winning condition: Eve's run is accepting if either of Adam's two runs are accepting.

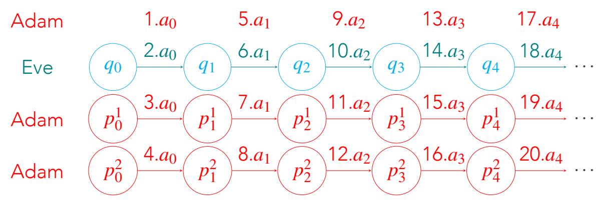


Figure 5: Order of moves in a 2-token game. At the end of an infinite play, Adam constructs a word and Eve and Adam construct one and two runs on that word respectively.

2-token conjecture. Eve wins the 2-token game on a parity automaton A if and only if A is history-deterministic [1].
The 2-token conjecture holds for Büchi [1] and coBüchi automata [2]. Assuming that the 2-token conjecture is true, we would only obtain a PSPACE-upper bound for the problem of deciding history-determinism.
Open: Given a parity automaton A, what is the complexity of deciding if Eve wins the 2-token game of A? What is the complexity of deciding history-determinism of A?

References

[1] Marc Bagnol and Denis Kuperberg. Büchi Good-for-Games Automata Are Efficiently Recognizable. In *FSTTCS*, 2018.

[2] Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak. On the Succinctness of Alternating Parity Good-For-Games Automata. *arxiv*, 2020.

[3] Antonio Casares, Thomas Colcombet, Nathanaël Fijalkow, and Karoliina Lehtinen. From muller to parity and rabin automata: Optimal transformations preserving (history-)determinism. *arxiv*, 2023.

[4] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized Parity Games. In *FoSSaCS*, 2007.

[5] Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *CSL*, 2006.

[6] Denis Kuperberg and Michal Skrzypczak. On determinisation of good-for-games automata. In *ICALP*, 2015.

Symbolic Solution of Emerson-Lei Games for Reactive Synthesis

Daniel Hausmann, Mathieu Lehaut and Nir Piterman

hausmann@chalmers.se, lehaut@chalmers.se, piterman@chalmers.se

University of Gothenburg, Sweden

Overview

- Winning regions in various ω-regular games are known to be nested fixpoints.
- Emerson-Lei objectives succinctly encode standard objectives.
- Zielonka trees characterize winning in Emerson-Lei games.

We show how to extract a nested fixpoint from any Zielonka tree, resulting in a *symbolic* fixpoint algorithm that solves Emerson-Lei games with n nodes, m edges and k colors in time $\mathcal{O}(k! \cdot m \cdot n^{\frac{k}{2}})$.

This generalizes previous fixpoint algorithms for Büchi, parity, GR[1], Rabin and Streett games, recovering previous upper bounds on runtime.

Emerson-Lei Games

Infinite-duration zero-sum games played by two players $\exists$ and $\forall$:

$$G = (V = V_\exists \cup V_\forall, E \subseteq V \times V, \mathsf{col} : V \to 2^C, \varphi) \qquad \varphi \in \mathbb{B}(\mathsf{GF}(C))$$

Player $\exists$ wins play $\pi \subseteq V^\omega$ in G if and only if $\mathsf{col}[\pi] \models \varphi$

Examples:

$$\varphi = \mathsf{GF}\, f \qquad \text{(Büchi)}$$

$$\varphi = \bigwedge_{1\leq i\leq k} \mathsf{GF}\, f_i \qquad \text{(gen. Büchi)}$$

$$\varphi = \bigwedge_{1\leq i\leq k_1} \mathsf{GF}\, p_i \to \bigwedge_{1\leq j\leq k_2} \mathsf{GF}\, q_j \qquad \text{(GR[1])}$$

$$\varphi = \bigvee_{i \text{ even}} \mathsf{GF}\, p_i \wedge \mathsf{FG} \bigwedge_{i<j\leq k} \neg p_j \qquad \text{(parity)}$$

$$\varphi = \bigvee_{1\leq i\leq k} \mathsf{GF}\, e_i \wedge \mathsf{FG}\, \neg f_i \qquad \text{(Rabin)}$$

$$\varphi = \bigwedge_{1\leq i\leq k} (\mathsf{GF}\, r_i \to \mathsf{GF}\, g_i) \qquad \text{(Streett)}$$

$$\varphi = \bigvee_{U\in\mathcal{U}} \bigwedge_{i\in U} \mathsf{GF}\, f_i \wedge \mathsf{FG} \bigwedge_{j\notin U} f_j \qquad \text{(Muller for } \mathcal{U} \subseteq 2^C\text{)}$$

Emerson-Lei games are determined, but not positional (e.g. Streett games).

Zielonka Trees

Tree $\mathcal{Z}_\varphi$ with vertices X labeled by $l(X) \subseteq C$, subject to certain maximality conditions. Vertex X is *green* if $l(X)^\omega \models \varphi$ and *red* otherwise.

Require for all children Y, Y' of X in $\mathcal{Z}_\varphi$:
X green $\Leftrightarrow$ Y red, $l(Y) \subsetneq l(X)$, $l(Y)$ and $l(Y')$ are incomparable.

Lemma: The Zielonka tree $\mathcal{Z}_\varphi$ has at most $e \cdot |C|!$ vertices.

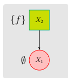
Play $\pi = v_0 v_1 \ldots$ induces *walk* ρ_π through Zielonka tree:

- start with v_0 and left-most leaf in Zielonka tree;
- at v_i and X, pick lowest ancestor Y of X s.t. $\mathsf{col}(v_i) \subseteq l(Y)$ and proceed with v_{i+1} and left-most leaf X' under Y that is to right of X

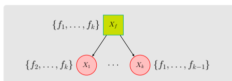
Dominating vertex: topmost node that is seen infinitely often in ρ_π.

Lemma: Player $\exists$ wins play π $\Leftrightarrow$ dominating vertex in ρ_π is green.

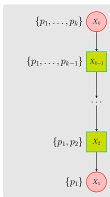
Zielonka Trees by Example



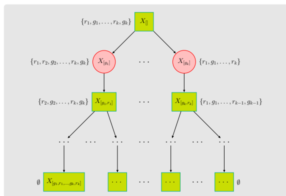
Büchi objective



generalized Büchi objective



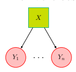
parity objective



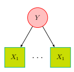
Streett objective

Fixpoint Extraction – Building Blocks

Inner vertices:

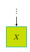


$\rightsquigarrow \; X =_\nu Y_1 \cap \ldots \cap Y_n$

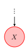


$\rightsquigarrow \; Y =_\mu X_1 \cup \ldots \cup X_n$

Leaf vertices:



$\rightsquigarrow \; X =_\nu \bigcup_{Y \geq X} \mathsf{Walk}(X,Y) \cap \mathsf{Cpre}(Y)$



$\rightsquigarrow \; X =_\mu \bigcup_{Y \geq X} \mathsf{Walk}(X,Y) \cap \mathsf{Cpre}(Y)$

where

$$\mathsf{Walk}(X,Y) = \{v \in V \mid Y \text{ is lowest ancestor of } X \text{ s.t. } \mathsf{col}(v) \subseteq l(Y)\}$$

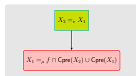
for vertices X, Y, and Cpre encodes one-step attraction for player $\exists$.

Main Result

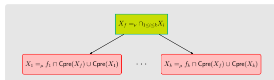
Theorem: The solution of the extracted fixpoint equation system is the winning region in the corresponding Emerson-Lei game.

Solve equation systems by fixpoint iteration to solve Emerson-Lei games with n nodes and k colors *symbolically* in time $\mathcal{O}(k! \cdot n^{\frac{k}{2}+2})$. For simpler conditions, this recovers previous fixpoint iteration algorithms.

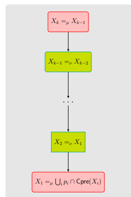
Extracted Fixpoint Systems by Example



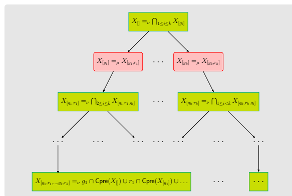
Büchi objective



generalized Büchi objective



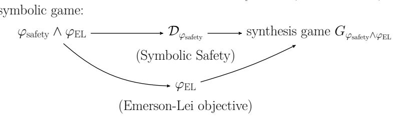
parity objective



Streett objective

Symbolic Reactive Synthesis

Reduction of safety and EL LTL formula $\varphi_{\text{safety}} \wedge \varphi_{\text{EL}}$ (with $\varphi_{\text{EL}} \in \mathbb{B}(\mathsf{GF}(C))$) to symbolic game:



Check realizability in time $2^{\mathcal{O}(m \cdot \log m \cdot 2^n)}$, where $n = |\varphi_{\text{safety}}|$ and $m = |\varphi_{\text{EL}}|$.

UNIVERSITY OF GOTHENBURG

CHALMERS

European Research Council
Established by the European Commission

More details and results in full paper: https://arxiv.org/pdf/2305.02793.pdf

Higher-Order Mathematical Operational Semantics

Weak Applicative Similarity

Cool Format

Passive operators are specified as

$$f(x_1,\dots,x_n) \to t \qquad f(x_1,\dots,x_n) \xrightarrow{x} t$$

Cool HO Specification must have

$$\frac{x_m \to y_m}{f(x_1,\dots,x_m,\dots x_n) \to f(x_1,\dots,y_m,\dots x_n)}$$

$$\frac{x_m \xrightarrow{x_i} x_i'}{f(x_1,\dots,x_n) \to t} \quad \text{or} \quad \frac{x_m \xrightarrow{x_i} x_i' \quad x_m \xrightarrow{x} x_i''}{f(x_1,\dots,x_n) \xrightarrow{x} t}$$

with $x_m, x_m' \notin \mathit{Vars}(t)$, for non-passive f

more abstractly

Soundness

Rules for **strong transitions** $\to$, $\xrightarrow{x}$ are sound for **weak transitions** $\Rightarrow$, $\xRightarrow{x}$, e.g.:

$$\frac{t \to t'}{ts \to t's} \quad \rightsquigarrow \quad \frac{t \Rightarrow t'}{ts \Rightarrow t's}$$

yet more abstractly

Lax-Bialgebra

$$\begin{array}{ccccc}
\Sigma\mu\Sigma & \xrightarrow{\iota} & \mu\Sigma & \xrightarrow{\tilde{\gamma}} & B(\mu\Sigma,\mu\Sigma) \\
\downarrow{\scriptstyle \Sigma\langle \mathrm{id},\tilde{\gamma}\rangle} & & \text{\rotatebox{90}{$\leq$}} & & \uparrow{\scriptstyle B(\mathrm{id},\hat{\iota})} \\
\Sigma(\mu\Sigma \times B(\mu\Sigma,\mu\Sigma)) & \xrightarrow{\rho} & B(\mu\Sigma,\Sigma^\star(\mu\Sigma+\mu\Sigma)) & \xrightarrow{B(\mathrm{id},\Sigma^\star\nabla)} & B(\mu\Sigma,\Sigma^\star\mu\Sigma)
\end{array}$$

for some weak transition semantics $\tilde{\gamma}\colon \mu\Sigma \to B(\mu\Sigma,\mu\Sigma)$

Central Result: Compositionality for Free

For suitably defined weak semantics $\tilde{\gamma}$, ensuing similarity relation $\precsim$ is a congruence

Logical Predicates and Strong Normalization

Liftings

Reasoning with predicates requires **liftings**:

$$\begin{array}{ccc}
\mathbf{Pred}(\mathbf{C}) & \xrightarrow{\overline{\Sigma}} & \mathbf{Pred}(\mathbf{C}) \\
\downarrow{\scriptstyle |-|} & & \downarrow{\scriptstyle |-|} \\
\mathbf{C} & \xrightarrow{\Sigma} & \mathbf{C}
\end{array}
\qquad
\begin{array}{ccc}
\mathbf{Pred}(\mathbf{C})^{\mathrm{op}} \times \mathbf{Pred}(\mathbf{C}) & \xrightarrow{\overline{B}} & \mathbf{Pred}(\mathbf{C}) \\
\downarrow{\scriptstyle |-|^{\mathrm{op}} \times |-|} & & \downarrow{\scriptstyle |-|} \\
\mathbf{C}^{\mathrm{op}} \times \mathbf{C} & \xrightarrow{B} & \mathbf{C}
\end{array}$$

Invariants

Coalgebraic S-relative invariant: for given predicate $S \subseteq \mu\Sigma$, such $Q \subseteq \mu\Sigma$ that

$$Q \leqslant \gamma^{-1}[\overline{B}(S,Q)]$$

Logical invariant = logical predicate is a coalgebraic invariant relative to itself

Key Construction: Logical predicate over P

We define predicate transformer $\Box$:

Given a program property P, $\Box P$ is a canonical logical predicate, contained in P

Key Result: Induction up to $\Box$

Induction up to $\Box$ is a lightweight proof principle sound for well-behaved (**relatively flat**) HO Specifications, which isolates the non-trivial core from the boilerplate part of the proof:

- Prove $\iota[\overline{\Sigma}(\Box P)] \implies P$
- By generalities: $\iota[\overline{\Sigma}(\Box P)] \implies \Box P$, hence $\top \implies \Box P \implies P$

Key Application: Strong Normalization

$P(t) = \mathrm{SN}(t) =$ "all reductions $t \to t' \to \cdots$ are finite"

Higher-Order Abstract GSOS

Categorical Framework for Higher-Order Operational Semantics

Language

Signature $\cong$ Endofunctor $\Sigma\colon \mathbf{C} \to \mathbf{C}$ on a category $\mathbf{C}$, e.g.:

- $\mathbf{C} = \mathbf{Set}$, $\Sigma = \{0/0, a_i/0, +/2, \cdot/2\}$
- $\mathbf{C} =$ "nominal sets", $\Sigma X = \mathbb{A} + [\mathbb{A}]X + X \times X$

Behaviour

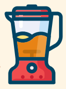
Behaviour = Mixed-variance functor $B\colon \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{C}$, e.g.:

- $B(X,Y) = Y^X + Y$ (deterministic)
- $B(X,Y) = \mathscr{P}_\omega(Y^X + Y)$ (non-deterministic)

HO Specification in GSOS Format

$$\frac{t \xrightarrow{a} t' \quad s \xrightarrow{\bar{a}} s'}{t \| s \xrightarrow{\tau} t' \| s'} \qquad \frac{t \xrightarrow{s} t'}{ts \longrightarrow t'} \qquad \overline{(\lambda x.f)s \longrightarrow f[s/x]}$$

$\Longrightarrow$ **Distributive law** ρ of Σ over B



Higher-Order Bialgebraic Semantics

Transition semantics is a unique solution $\gamma\colon \mu\Sigma \to B(\mu\Sigma,\mu\Sigma)$:

$$\begin{array}{ccccc}
\Sigma\mu\Sigma & \xrightarrow{\iota} & \mu\Sigma & \xrightarrow{\gamma} & B(\mu\Sigma,\mu\Sigma) \\
\downarrow{\scriptstyle \Sigma\langle \mathrm{id},\gamma\rangle} & & & & \uparrow{\scriptstyle B(\mathrm{id},\hat{\iota})} \\
\Sigma(\mu\Sigma \times B(\mu\Sigma,\mu\Sigma)) & \xrightarrow{\rho} & B(\mu\Sigma,\Sigma^\star(\mu\Sigma+\mu\Sigma)) & \xrightarrow{B(\mathrm{id},\Sigma^\star\nabla)} & B(\mu\Sigma,\Sigma^\star\mu\Sigma)
\end{array}$$

Generic Strong Applicative Bisimulation

Coalgebraic notion of **strong applicative bisimilarity** $\sim$ on initial Σ-algebra $\mu\Sigma$ (=algebra of programs) as a pullback

$$\begin{array}{ccc}
\sim & \longrightarrow & \mu\Sigma \\
\downarrow & \lrcorner & \downarrow{\scriptstyle \mathrm{coit}\,\gamma} \\
\mu\Sigma & \xrightarrow{\mathrm{coit}\,\gamma} & \nu\gamma.\,B(\mu\Sigma,\gamma)
\end{array}$$

Central Result: Compositionality for Free

Under certain general assumptions, $\sim$ is a congruence

Contextual Equivalence and Step-Indexing

"Lazy" Contextual Preorder

"Lazy" contextual preorder/equivalence for programs of type τ (w.r.t. convergence relation $\Downarrow$):

$$t \lesssim_\tau s \text{ if } \forall \tau', \text{contexts } C\colon \tau \rightsquigarrow \tau'.\, C[t]\Downarrow \implies C[s]\Downarrow$$
$$t \simeq_\tau s \text{ if } \forall \tau', \text{contexts } C\colon \tau \rightsquigarrow \tau'.\, C[t]\Downarrow \iff C[s]\Downarrow$$

E.g. $f \lesssim_\tau \lambda x.\, f x$, but $f \not\simeq_\tau \lambda x.\, f x$

Ground Contextual Preorder

Ground contextual preorder/equivalence for programs of type τ w.r.t. Booleans:

$$t \lesssim_\tau^{\mathrm{bool}} s \text{ if } \forall \text{contexts } C\colon \tau \rightsquigarrow \mathrm{bool}.\, C[t]\Downarrow \implies C[s]\Downarrow$$
$$t \simeq_\tau^{\mathrm{bool}} s \text{ if } \forall \text{contexts } C\colon \tau \rightsquigarrow \mathrm{bool}.\, C[t]\Downarrow \iff C[s]\Downarrow$$

E.g. $f \simeq_\tau^{\mathrm{bool}} \lambda x.\, f x$

more abstractly

more abstractly

Abstract Contextual Preorder

For a preorder $O \subseteq \mu\Sigma \times \mu\Sigma$, **contextual preorder** w.r.t. O is the greatest congruence $\lesssim^O \subseteq O$ on $\mu\Sigma$

Abstract Step-Indexed Logical Relations

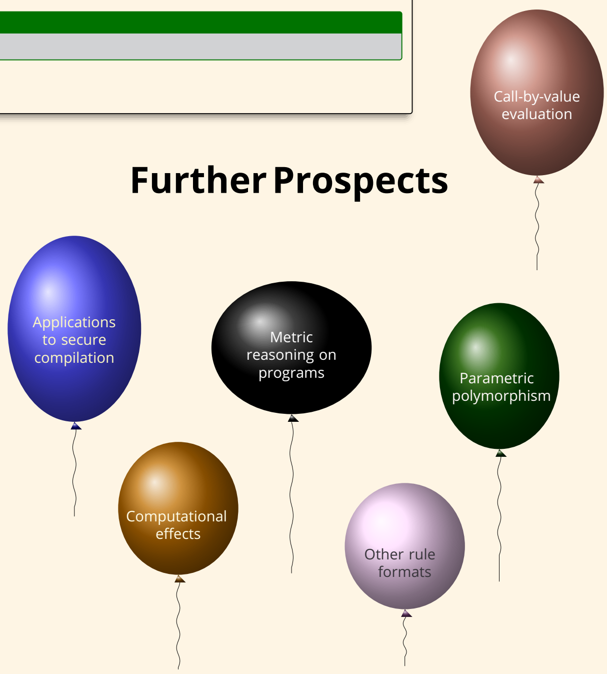
For a relation $R \subseteq \mu\Sigma \times \mu\Sigma$ on programs, by transfinite recursion:

$$\Box^0 R = R \qquad \Box^\alpha R = \textstyle\bigwedge_{\beta<\alpha} \Box^\beta R \quad \text{for limits ordinals } \alpha$$
$$\Box^{\alpha+1} R = \Box^\alpha R \wedge (\gamma \times \tilde{\gamma})^{-1}[\overline{B}(\Box^\alpha R, \Box^\alpha R)] \qquad \Box^\nu R = \textstyle\bigwedge_\alpha \Box^\alpha R$$

Central Result: Soundness of Abstract Logical Relation Method

- Every $\Box^\alpha \top$ is a congruence
- $\Box^\nu \top$ is sound for contextual preorder

Further Prospects



References

[1] S. Goncharov, S. Milius, L. Schröder, S. Tsampas, H. Urbat, *Towards a Higher-Order Mathematical Operational Semantics, POPL 2023*

[2] H. Urbat, S. Tsampas, S. Goncharov, S. Milius, L. Schröder, *Weak Similarity in Higher-Order Mathematical Operational Semantics, LICS 2023*

[3] S. Goncharov, A. Santamaria, L. Schröder, S. Tsampas, H. Urbat, *Logical Predicates in Higher-Order Mathematical Operational Semantics, FoSSaCS 2024*

Stochastic Window Mean-Payoff Games

Laurent Doyen[1], Pranshu Gaba[⋆2], and Shibashis Guha[2]

[1]CNRS & LMF, ENS Paris-Saclay, France

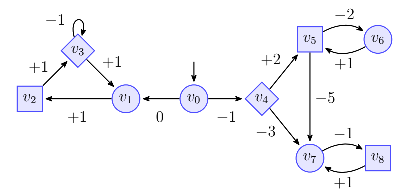
[2]Tata Institute of Fundamental Research, Mumbai, India,

[⋆]pranshu.gaba@tifr.res.in

Setup

- Played by 2 players
  - player ◯ (*system*)
  - player □ (*environment*)
- Played on a directed graph with no deadlocks
  - Vertices partitioned into (◯, □, ◊)
  - Probability distributions over out-edges of ◊
  - Edges have rational payoffs, $w(e)$

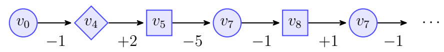
Example



Gameplay

1. Place token on initial vertex v_{init}.
2. If token is on ◯, then player ◯ chooses an out-edge. If token is on □, then player □ chooses an out-edge. If token is on ◊, then an out-edge is chosen by the probability distribution.
3. Move token along the chosen out-edge and go to step 2.

A *play* is an infinite path in the arena.

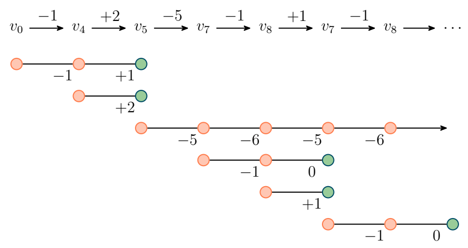


Window mean-payoff objective $\mathsf{WMP}(\ell)$

Given *window length* $\ell \geq 1$.

A play π satisfies $\mathsf{WMP}(\ell)$ if *eventually*, starting from every point in π, the mean-payoff becomes non-negative in at most ℓ steps.

The objective of player ◯ is to satisfy $\mathsf{WMP}(\ell)$.
The objective of player □ is to not satisfy $\mathsf{WMP}(\ell)$.



Strategies

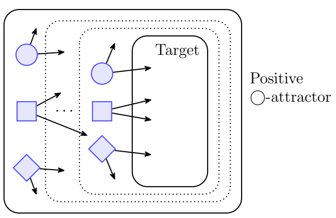
A function that reads the sequence of vertices seen so far, and returns the out-edge that the players should choose.

Decision problem

Given $0 \leq p \leq 1$, does player ◯ have a strategy to satisfy $\mathsf{WMP}(\ell)$ with probability at least p?

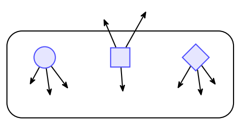
Positive ◯-attractor of a set

All vertices from which player ◯ can ensure that the token eventually reaches the set with positive probability.



Trap for player ◯

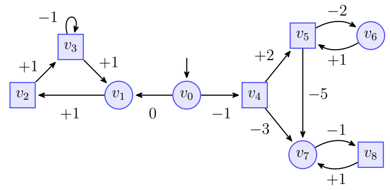
A subset from which player □ can ensure that the token never leaves.



Note: The complement of a positive ◯-attractor is a trap for player ◯.

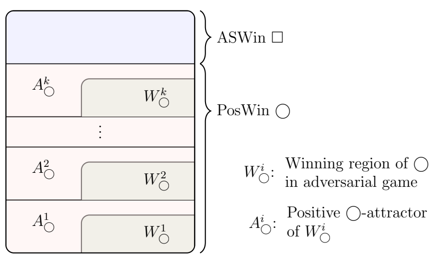
Adversarial non-stochastic game

Game obtained by changing every ◊ to □.

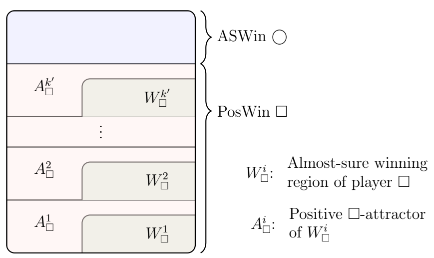


If player ◯ wins in the adversarial game, then she surely wins in the original game.

Positive winning $(p > 0)$



Almost-sure winning $(p = 1)$



Arbitrary $0 < p < 1$

Follow value class construction as illustrated in [2].

- *Guess* the probability p_v of player ◯ satisfying $\mathsf{WMP}(\ell)$ from each vertex v.
- This yields a partition of vertices in the graph, called *value classes*.
- For each value class, check if the players almost-surely win the objective $\mathsf{WMP}(\ell) \cup \mathrm{Reach}(\mathrm{Bnd})$.

Memory

The *memory* of a strategy is the minimum number of states required to describe the strategy.
Player 1 requires ℓ memory.
Player 2 requires $|V| \cdot \ell$ memory.

Results

For the $\mathsf{WMP}(\ell)$ objective,

- positive winning winning is in P
- almost-sure winning winning is in P
- arbitrary p is in $\mathsf{NP} \cap \mathsf{coNP}$

References

[1] K. Chatterjee, L. Doyen, M. Randour, and J-F. Raskin. "Looking at mean-payoff and total-payoff through windows". In: *Information and Computation* 242 (2015), pp. 25–52.

[2] K. Chatterjee, T. A. Henzinger, and F. Horn. "Stochastic Games with Finitary Objectives". In: *MFCS*. Springer Berlin Heidelberg, 2009, pp. 34–54.

TACAS 2024 Posters

Btor2-Cert: A Certifying Hardware-Verification Framework Using Software Analyzers

SoSy-Lab Software Systems

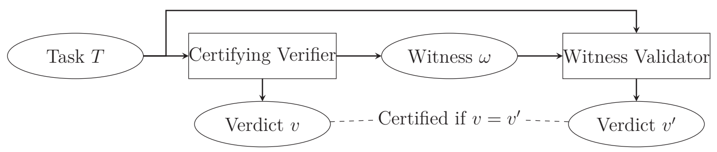
LMU Ludwig-Maximilians-Universität München

Zsófia Ádám, Dirk Beyer, Po-Chun Chien, Nian-Ze Lee, and Nils Sirrenberg

adamzsofi@edu.bme.hu, nils.sirrenberg@campus.lmu.de,
{dirk.beyer,po-chun.chien,nian-ze.lee}@sosy.ifi.lmu.de

Presentation at TACAS 2024: 12:00, Thursday, April 11, Room: TBD

Certifying and Validating Verification



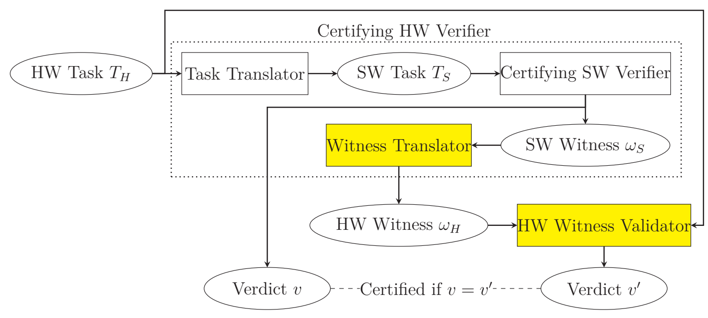
Our Motivation

- Explainable and trustworthy HW verification (HV)
- SW verification (SV) techniques for HW

Our Contributions

- A certifying HV framework using SV techniques
- A translator from SW witnesses to HW witnesses
- A witness validator for the Btor2 HW modeling language [6]
- Complementing HV with certified results from SV

Certifying HV Using Translation and SV



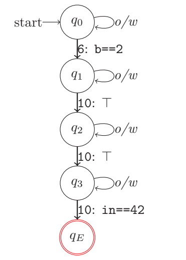
Btor2-Cert instantiates the framework with Btor2C [1] as frontend and SW verifiers that export GraphML witnesses [2] as backend.

HW-to-SW Translation via Btor2C [1]

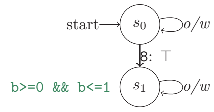
```
1 sort bitvec 8
2 sort bitvec 1
3 constd 1 42
4 constd 1 2
5 zero 1
6 state 1 ; a
7 state 1 ; b
8 input 1 ; in
9 init 1 6 4 ; a init to 2
10 init 1 7 5 ; b init to 0
11 eq 2 6 5 ; a == 0
12 eq 2 7 4 ; b == 2
13 eq 2 8 3 ; in == 42
14 and 2 11 12
15 and 2 13 14
16 bad 15
17 one 1
18 srl 1 6 17
19 xor 1 7 17
20 next 1 6 18
21 next 1 7 19
```

```c
extern void abort(void);
extern unsigned char nondet_uchar();
void main() {
  typedef unsigned char SORT_1;
  SORT_1 a = nondet_uchar();
  SORT_1 b = nondet_uchar();
  a = 2;
  b = 0;  // Omit for unsafe version
  for (;;) {
    SORT_1 in = nondet_uchar();
    if (a == 0 && b == 2 && in == 42) {
    ERROR:
      abort();
    }
    a = a >> 1;
    b = b ^ 1;
  }
}
```

Witness Translation

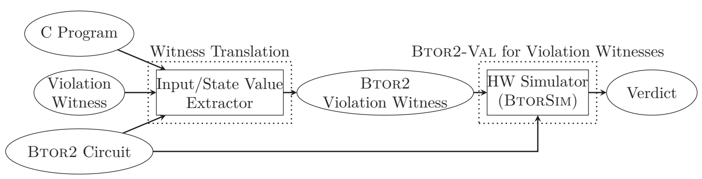



```
sat
b0
#0
1 00000010 ; b==2
@0
@1
@2
0 00101010 ; in==42
.
```

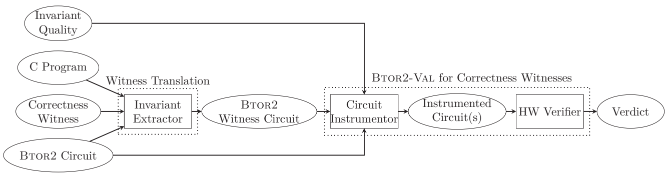



```
1 sort bitvec 8
2 sort bitvec 1
3 zero 1
4 one 1
5 input 1 ; state "b"
6 ugte 2 5 3 ; b >= 0
7 ulte 2 5 4 ; b <= 1
8 and 2 6 7
9 output 8
```

Violation Witness Validation



Correctness Witness Validation



Summary of Experimental Results

On 758 safe and 456 unsafe Btor2 verification tasks, Btor2-Cert achieved:

- Translation of all violation and 97 % correctness witnesses,
- Effective and efficient validation vs. compared validators, e.g., LIV [4] and CPA-w2t [3], and
- Certified bugs in 8 % of the unsafe tasks with CBMC [5] that HV overlooked

Invariant Quality

Three user-defined quality levels for invariants:

- Invariant (containing all reachable states)
- Safe invariant (implying safety property)
- Safe and inductive invariant

References

[1] Beyer, D., Chien, P.C., Lee, N.Z.: Bridging hardware and software analysis with Btor2C: A word-level-circuit-to-C translator. In: Proc. TACAS. pp. 1–21. LNCS 13994 (2023)

[2] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Lemberger, T., Tautschnig, M.: Verification witnesses. ACM Trans. Softw. Eng. Methodol. **31**(4), 57:1–57:69 (2022)

[3] Beyer, D., Dangl, M., Lemberger, T., Tautschnig, M.: Tests from witnesses: Execution-based validation of verification results. In: Proc. TAP. pp. 3–23. LNCS 10889 (2018)

[4] Beyer, D., Spiessl, M.: LIV: A loop-invariant validation using straight-line programs. In: Proc. ASE. pp. 2074–2077 (2023)

[5] Clarke, E.M., Kröning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proc. TACAS. pp. 168–176. LNCS 2988 (2004)

[6] Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2, BtorMC, and Boolector 3.0. In: Proc. CAV. pp. 587–595. LNCS 10981 (2018)

Try Btor2-Cert!

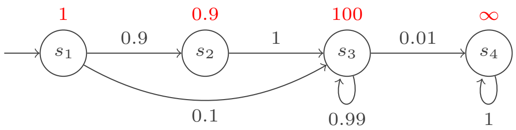
Artifact DOI: 10.5281/zenodo.10548597

Accurately Computing Expected Visiting Times and Stationary Distributions in Markov Chains

Hannah Mertens, Joost-Pieter Katoen, Tim Quatmann, Tobias Winkler

Expected Visiting Times (EVTs) [2]

- Describe the expected time a Markov chain spends in each state.
- Characterized as the unique solution of a linear equation system.
- Useful for obtaining reachability probabilities for multiple states, stationary distributions, and expected rewards.



Contributions

- **Sound and scalable** algorithms for computing EVTs.
- Optimized methods for computing stationary distributions and conditional expected rewards by leveraging EVTs.
- An implementation in Storm [1].
- An experimental evaluation.

Applications of EVTs

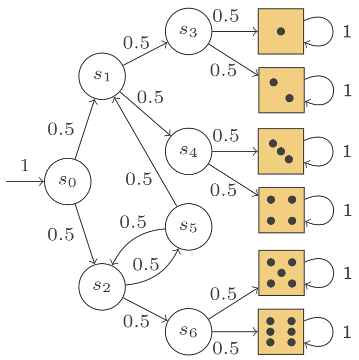
Reachability probabilities:
- Computing reachability probability of each BSCC reduces to EVTs [2].
- One linear equation system instead of one per BSCC.

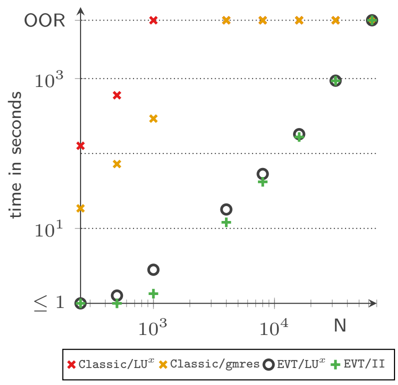
Stationary distribution:
- Sound bounds on the stationary distribution via EVTs.
- Significantly faster than existing techniques [3, 5].

Conditional expected reward:
- Given the EVTs, compute the expected rewards conditioned on reaching each BSCC.
- One linear equation system rather than one per BSCC.

Uniform Distribution Generator

For a given parameter $N \geq 1$, we verify that Lumbroso's Fast Dice Roller [4] program produces a uniformly distributed output in $\{1, \dots, N\}$ by computing the stationary distribution of the corresponding DTMC.



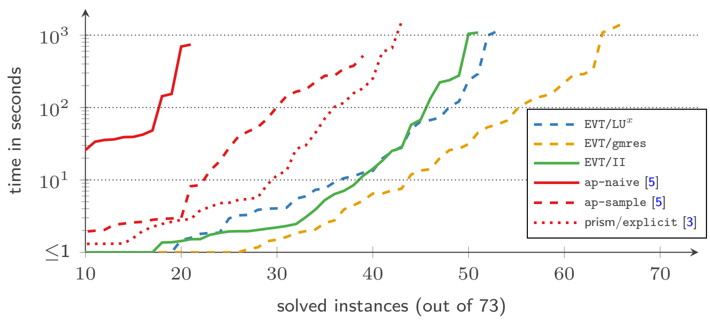


Approximating EVTs

Value iteration (VI):
- Characterize EVTs as the fixed point of an operator.
- Iteratively apply the operator.
- Converges to the unique fixed point in the limit, but no sound stopping criterion.

Interval iteration (II):
- Converge to the fixed point from *above and below*.
- Stop when the difference between under- and over-approximations is small enough
  ⇝ Sound precision guarantees.

Computing Stationary Distributions via EVTs



References

[1] Christian Hensel et al. "The Probabilistic Model Checker Storm". In: *Int. J. Softw. Tools Technol. Transf.* 2022.

[2] J.G. Kemeny and J.L. Snell. *Finite Markov Chains.* Undergraduate Texts in Mathematics. 1976.

[3] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. "PRISM 4.0: Verification of Probabilistic Real-Time Systems". In: *CAV.* 2011.

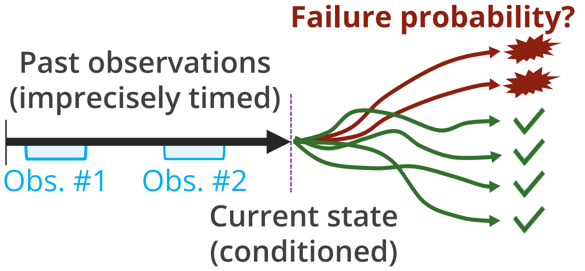
[4] Jérémie O. Lumbroso. "Optimal Discrete Uniform Generation from Coin Flips, and Applications". In: *CoRR* abs/1304.1916 (2013).

[5] Tobias Meggendorfer. "Correct Approximation of Stationary Distributions". In: *TACAS.* 2023.

UnRAVeL Research Training Group – Uncertainty and Randomness in Algorithms, Verification, and Logic
Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

CTMCs with Imprecisely Timed Observations

Thom Badings, Matthias Volk, Sebastian Junges, Mariëlle Stoelinga, Nils Jansen



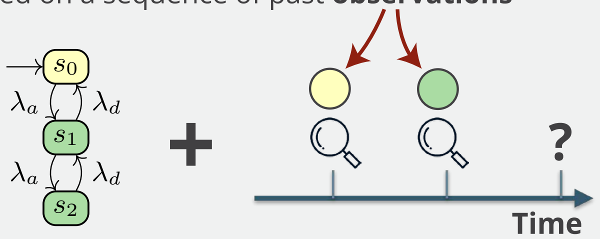
We compute reachability probabilities* for CTMCs, conditioned on a sequence of (imprecisely timed) observations.

**and other measures, such as weighted reachability or rewards*

1. Motivation

Continuous-time Markov chains (CTMCs)

- Stochastic processes subject to **random timing**
- Assumption: states are **observed as colors**
- **System monitoring:** determine current state, based on a sequence of past **observations**



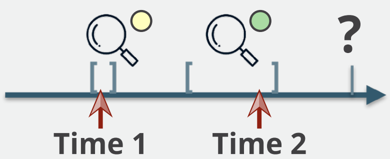
- Observation times might be **imprecisely known**
- Example: an inspection was done **last week,** but the **precise time is unknown**

Main question

How to compute the current state (and predict failures), if observation times are uncertain?

2. Problem statement

- Sequence of timed observations ("**evidence**"), each of which consists of:
  1. A **known color** (state label)
  2. An **uncertain observation time** (e.g., interval)
- Fixing a **precise time** for every observation gives an **instance** of the imprecise evidence



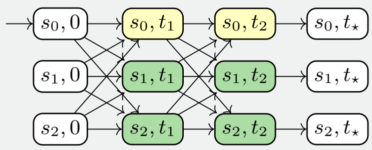
Every instance gives a different conditional reachability probability

Problem

Compute the maximum (weighted) probability of reaching a subset of (failed) states, over all instances of the evidence

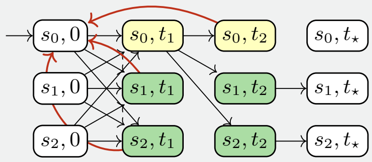
3. Our 3-step unfolding method

1. Unfold CTMC + evidence into an MDP



- Infinitely many states/actions
- Actions model evidence times

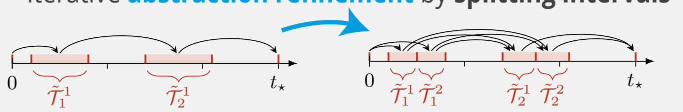
2. Condition unfolded MDP on the evidence



- Loop back every transition that violates the evidence

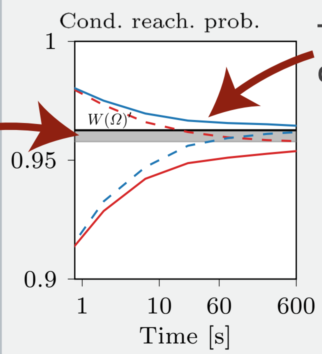
3. Abstract ***(infinite)*** MDP into ***(finite)*** interval MDP

- Each interval MDP state represents a **time interval**
- Iterative **abstraction refinement** by **splitting intervals**



4. Evaluation

- Implemented in the model checker Storm
- Tested on 5 CTMCs with 3 - 576 states
- Evidences with 2 - 15 observations



Tight lower/upper bounds on conditional failure probability

- Open challenges:
  - Tighter bounds for transient CTMC probabilities
  - Better refinement strategies
  - Improve analysis techniques for iMDPs (policy iteration)

Check out our paper!

PrimaVera
DEUCE. Data-Driven Verification and Learning under Uncertainty.
erc
Radboud University
TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF TWENTE.
RUHR UNIVERSITÄT BOCHUM
thom.badings@ru.nl
www.thombadings.nl

CESAR: Control Envelope Synthesis via Angelic Refinements

Aditi Kabra
akabra@cs.cmu.edu

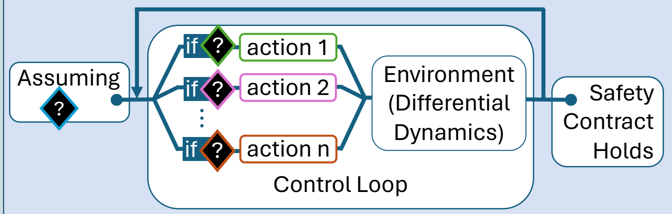
Jonathan Laurent
jonathan.laurent@kit.edu

Stefan Mitsch
smitsch@depaul.edu

André Platzer
platzer@kit.edu

1 CESAR: Formally Justified Synthesis

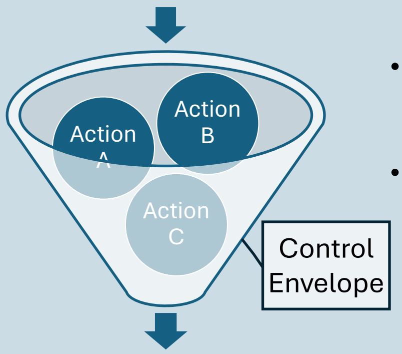
Human provides shape of hybrid system's model



Synthesis procedure fills holes (?). Which action is safe when?

Correct by Construction **Control Envelope**

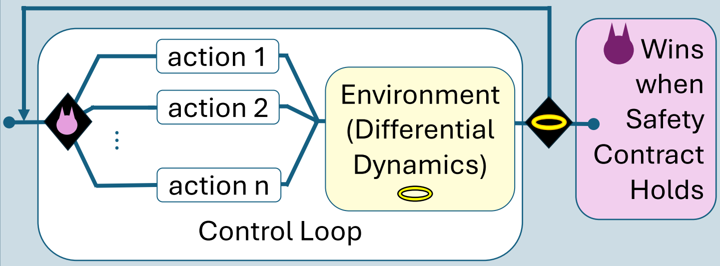
2 Control **Envelopes**



- Define ***families*** of safe controllers
- Full system monitored for adherence at runtime
- Higher order constraint compared to controllers: good solution permits as many safe control solutions as possible

3 Characterize Solution Implicitly using Hybrid Systems Game Theory ...

- Differential Game Logic (dGL) formalizes how two adversarial agents Angel ( ) and Demon ( ) take decisions to attain win conditions
- Use agent decisions to characterize the behavior of optimal control in a maximally difficult environment

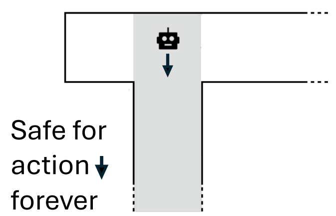


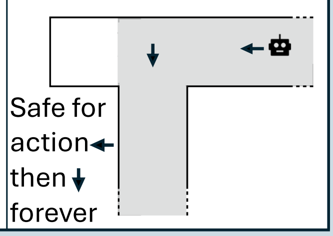
4 ... Then Extract an Explicit Solution

Reduce dGL formulas to solution formulas in propositional arithmetic using the axioms of dGL and ***refinements.***

Refinements transform games to be easier to reduce yet harder for the controller to win.

One-Shot Unrolling	**Bounded Unrolling**
What if the controller can only run one action for unbounded time?	What happens when the controller switches actions?





5 Evaluation: Varying Control Challenges

Benchmark	Synthesis Time (s)	Checking Time (s)	Optimal	Needs Unrolling	Non Solvable Dynamics
ETCS Train [1]	14	9	✓		
Sled	20	8	✓		
Intersection	49	44	✓		
Parachute [2]	46	8			✓
Curvebot	26	9			✓
Coolant	49	20	✓	✓	
Corridor	20	8	✓	✓	
Power Station	26	17	✓	✓	

CESAR automatically generates control conditions for all benchmarks.

Some benchmarks have non-solvable dynamics, some require a sequence of clever control actions to reach an optimal solution, and some have state-dependent fallbacks where the current state of the system determines which action is "safer".

[1] Platzer, A., Quesel, J.: European train control system: A case study in formal verification. In: Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009

[2] Fulton, N., Mitsch, S., Bohrer, R., Platzer, A.: Bellerophon: Tactical theorem proving for hybrid systems. In: Ayala-Rincon, M., Munoz, C.A. (eds.) ITP. LNCS, vol. 10499, pp. 207–224. Springer (2017).

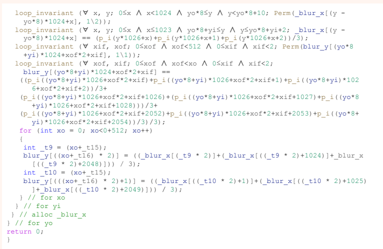
HaliVer: Deductive Verification and Scheduling Languages Join Forces

CHEOPS
VERIFIED CONSTRUCTION OF CORRECT AND OPTIMISED PARALLEL SOFTWARE

Problem

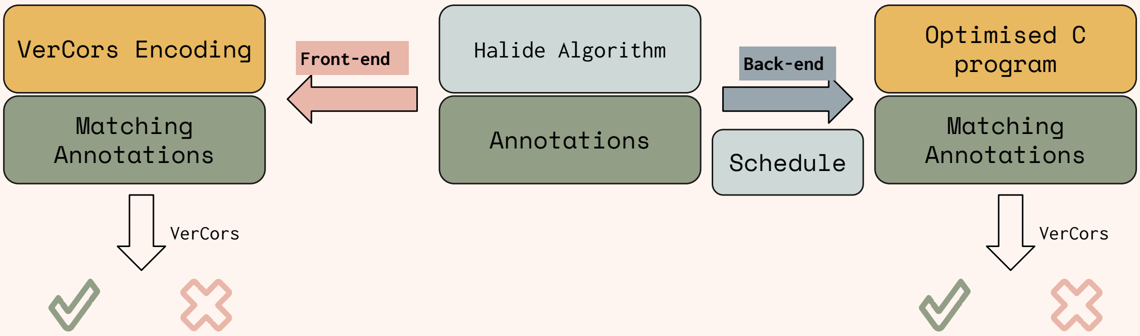
Verifying **optimised parallel** code is difficult because it

- uses **intricate** features that are hard to reason about.
- requires **precise** annotations that match the code, which is often harder than writing the code.



Idea

Use **Halide**, an existing DSL targeting the image & tensor domain.



HaliVer

1. The code is written in an **algorithm** part that captures the functionality.
2. Add **annotations** to the algorithm.
3. **Front-end approach:**
   a. Encode the algorithm with matching annotations.
   b. Verify with **VerCors**.
4. The algorithm is transformed using a **schedule**, producing optimised C code.
5. **Back-end approach:**
   a. **HaliVer** produces matching annotations for the optimised code, using similar transformations as the Halide compiler.
   b. Verify the optimised code using VerCors, proving **memory safety** and **functional correctness** properties.

Results & Future work

Results

- 8 different *algorithms*.
- 23 optimisation *schedules*.
- Without annotation effort proves **memory safety** for almost all programs.
- With annotation proves **functional correctness** properties.
- **Reduces** manual annotation **effort** by an order of magnitude.

Future work

- Target **GPUs**.
- **Vectorisation** optimisation.
- Verify arbitrary bounds (leads to non-linear arithmetic).
- Add **Axiomatic Data Types** and user defined pure functions to annotation language.

Contact info

l.b.v.d.haak@tue.nl
cheops.win.tue.nl/

Try it out yourself!

github.com/sakehl/HaliVerExperiments

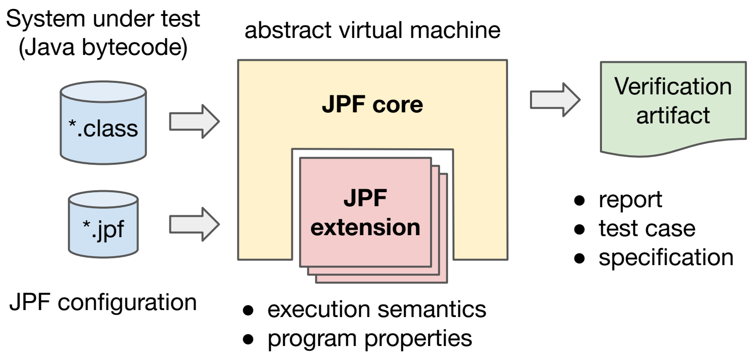
Lars B. van den Haak[1], Anton Wijs[1], Marieke Huisman[2] & Mark van den Brand[1]
[1]Eindhoven University of Technology [2]University of Twente

JPF: From 2003 to 2023

Cyrille Artho, Pavel Parízek, Daohan Qu, Varadraj Galgali, Pu (Luke) Yi

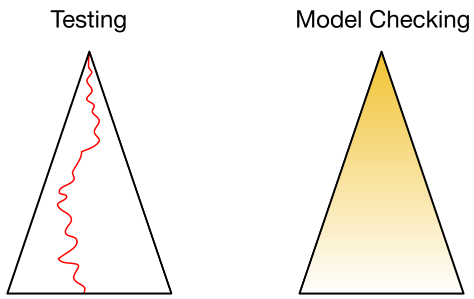
KTH Royal Institute of Technology; Charles University; Nanjing University; Belgaum; Stanford University

JPF: A bytecode analysis framework



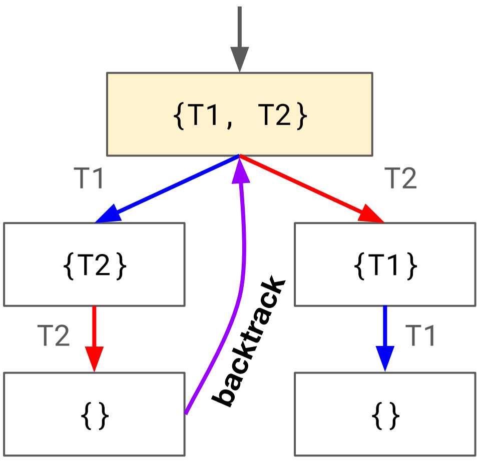
JPF core runs in Java (on host JVM)

Beyond Testing



Testing	Model Checking
only one trace	all (many) traces
may miss defects	finds all defects
scalable	resource-hungry

Software Model Checking



Strength: counterexample on failure

JPF successes

- Reliability analysis of NASA software components
- Locking protocol analysis of real-time kernel
- Analysis of java.nio libraries
- Teaching concurrency in Master's courses
- Detection of flaky tests

Challenges

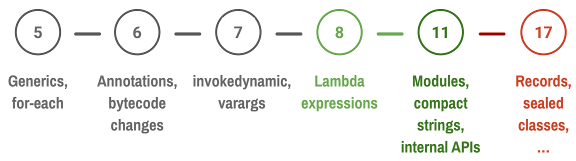
Native methods

```
private native int encode(...)
@MJI public int encode__Ljava_lang_...
```

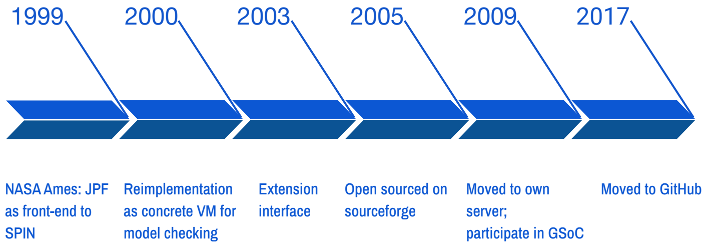
Bootstrap methods

```
invokedynamic makeConcatWithConstants
BootstrapMethods: REF_.... makeConcatWith...
```

Java version

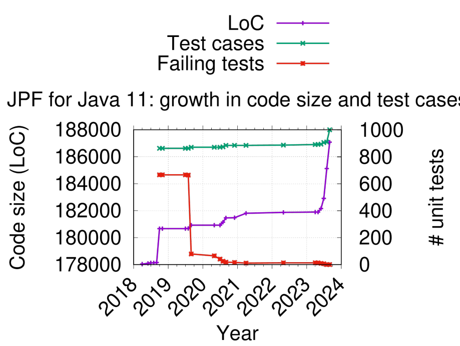


History of JPF



2005–today: Many extensions: Symbolic execution, native methods, analysis of networked software, analysis of Android apps

JPF for Java 11 development



Thanks to Google Summer of Code!

https://github.com/javapathfinder/jpf-core

MATA: A Fast and Simple Finite Automata Library

David Chocholatý Tomáš Fiedor Vojtěch Havlena
Lukáš Holík Martin Hruška Ondřej Lengál Juraj Síč
Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic

What is MATA

Mata is a well-engineered, fast, and simple automata library in C++. It is maintainable and understandable. It has a simple architecture allowing a new user, a researcher, to quickly prototype new algorithms and thoroughly optimize the final implementation. Mata targets string constraint solving, reasoning about regular expressions, regular model checking, student projects, and research prototypes. It comes with a large benchmark from string constraint solving, regular model checking, and reasoning about regular expressions.

Distinctive Features

- **Fast** and **simple**.
- **Explicit** representation of the transition relation.
- SOTA algorithms to work with **nondeterminism**.
- Modern development workflow and technologies.
- Easily **extensible** and **modifiable**.
- Well-documented, examples, testing infrastructure.
- **High-level API** with sane defaults, **low-level API** for maximal optimization.
- **Python interface**.
- A basis for a modular automata format `.mata`.

Usage

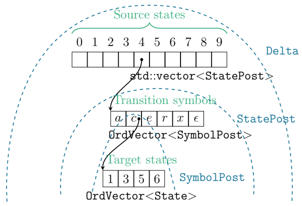
An example of using the C++ interface for Mata. The code loads automata from a file in the `.mata` format with bitvectors on transitions, mintermizes them, constructs NFAs from the loaded intermediate representations over the alphabet {`a`, `b`, `c`}, trims and determinizes the NFAs, adds a new transition with a new final state. It then creates a second automaton accepting the word `cbba`, and optionally concatenates the initial NFA with itself and prints the result in the `.mata` format, shown in the right-hand side.

```cpp
#include "mata/parser/mintermization.hh"
#include "mata/nfa/nfa.hh"
#include "mata/nfa/builder.hh"

#include <fstream>

using namespace mata;
using namespace mata::nfa;

int main(int argc, char *argv[]) {
    std::fstream file(argv[1], std::ios::in);
    for (IntermediateAut& inter_aut:
            IntermediateAut::parse_from_mf(parser::parse_mf(file, true))) {
        if (inter_aut.alphabet_type == IntermediateAut::AlphabetType::BITVECTOR) {
            inter_aut = Mintermization{}.mintermize(inter_aut);
        }
        EnumAlphabet alphabet{ 'a', 'b', 'c' };
        Nfa aut{ nfa::builder::construct(inter_aut, &alphabet) };
        nfa::determinize(aut.trim());
        State new_final{ aut.add_state() };
        aut.delta.add(*aut.initial.begin(), 'a', new_final);
        aut.final.insert(new_final);
        Nfa word_aut{ nfa::builder::create_single_word_nfa(Word{ 'c', 'b', 'b', 'a' })
          };
        if (nfa::is_included(word_aut, aut) && aut.is_in_lang(Word{ 'a', 'b' })) {
            aut = nfa::concatenate(aut, aut);
            aut.print_to_mata(std::cout);
        }
    }
    return EXIT_SUCCESS;
}
```

```
@NFA-explicit
%Alphabet-auto
%Initial q0
%Final q5 q6 q7
q0 97 q0
q0 97 q3
q0 98 q1
q0 99 q0
q1 97 q2
q1 97 q4
q1 97 q7
q1 98 q1
q1 98 q5
q1 99 q4
q2 97 q4
q2 97 q7
q2 98 q5
q2 99 q4
q3 97 q4
q3 97 q7
q3 98 q5
q3 99 q4
q4 97 q4
q4 97 q7
q4 98 q5
q4 99 q4
q5 97 q6
q5 98 q5
```

Architecture

The main determinant of Mata is its three-layered data structure `Delta` for the transition relation: an ordered vector indexed by states. For each state, an ordered vector of transitions over symbols, for each symbol, an ordered vector of target states.



Supported Operations

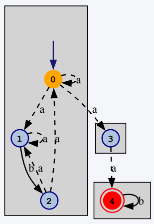
- Fine-grained modification of NFAs.
- Boolean language operations ($\cap, \cup, \bar{\cdot}$).
- **Mintermization** to handle large alphabets.
- **Antichain-based language inclusion**, equivalence, membership, emptiness.
- Determinization, minimization, **simulation reduction**.
- **ϵ-transitions**, ϵ-product, ϵ-removal.
- Rich **visualization** interface.
- **Parsing** of regexes (from RE2) and `.mata` format.

Python Interface

Mata provides an easy-to-use Python interface, as fast as C++ (`$ pip install libmata`).

An example of using Python interface for Mata. The code loads automata from regular expressions, concatenates them, and displays the trimmed concatenation with conditional formatting.

```python
from libmata import nfa, alphabets, parser, plotting
aut1 = parser.from_regex('((a+b)*a)*')
aut2 = parser.from_regex('aab*')
con_aut = nfa.nfa.concatenate(aut1, aut2).trim()
plotting.store()['alphabet'] = \
    alphabets.OnTheFlyAlphabet.from_symbol_map({'a':97, 'b':98})
e_h = [
    (lambda aut, e: e.symbol == 98, {'color':'black'}),
    (lambda aut, e: e.symbol == 97, {'style':'dashed','color':'black'})
]
n_h = [
    (lambda aut, q: q in aut.final_states,
        {'color':'red','fillcolor':'red'}),
    (lambda aut, q: q in aut.initial_states,
        {'color': 'orange', 'fillcolor': 'orange'}),
]
plotting.plot(con_aut, with_scc=True,
              node_highlight=n_h, edge_highlight=e_h)
```



Tool

Available on GitHub.

References

[1] Almeida, A., Almeida, M., Alves, J., Moreira, N., Reis, R.: Fado and guitar: Tools for automata manipulation and visualization. CIAA'09, (2009).

[2] Evans, C.: Automata (2023), https://github.com/caleb531/automata

[3] Isberner, M., Howar, F., Steffen, B.: AutomataLib, https://learnlib.de/projects/automatalib/

[4] Lengál, O., Šimáček, J., Vojnar, T.: VATA: A library for efficient manipulation of non-deterministic tree automata. TACAS'12, (2012)

[5] Lombardy, S., Marsault, V., Sakarovitch, J.: Awali, a library for weighted automata and transducers (version 2.0) (2021), http://vaucanson-project.org/Awali/2.0/

[6] Mller, A., et al.: Brics automata library, https://www.brics.dk/automaton/

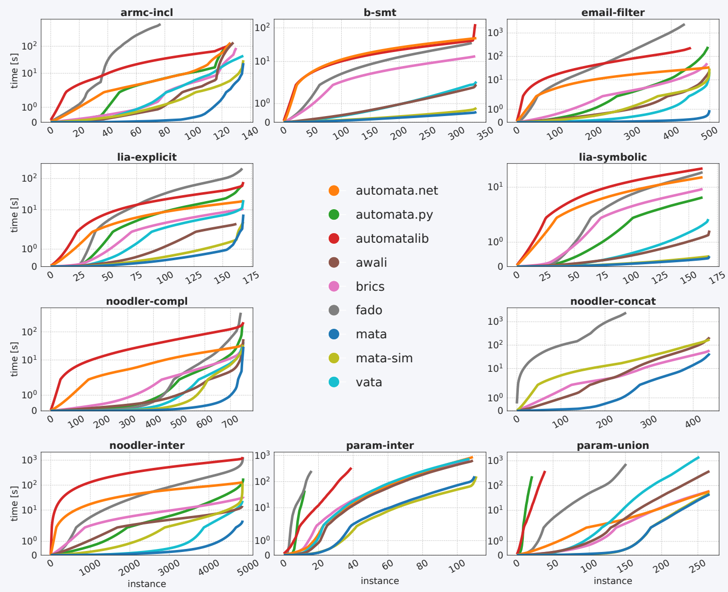
[7] Veanes, M.: A .NET automata library, https://github.com/AutomataDotNet/Automata

Experimental Evaluation

We compared Mata against Vata [4], Brics [6], Awali [5], Automata.net [7], AutomataLib [3], FAdo [1], and Automata.py [2], on a benchmark from string constraint solving, reasoning about regexes, regular model checking, and solving arithmetic formulae. Mata consistently outperforms all other libraries on all benchmarks in all operations. Mata is also the backbone of the efficiency of the **SMT solver Z3-Noodler** (with a poster nearby), which outperforms the state of the art on many standard benchmarks.

Cactus plots show cumulative run time. Time axes are logarithmic.
Tables show statistics for the benchmarks. We list the number of timeouts (TO, 60 s), average time on solved instances (*Avg*), median time over all instances (*Med*), and standard deviation over solved instances (*Std*). Best values are in **bold**, times are in milliseconds unless seconds are explicitly stated. ~0 means a value close to zero.

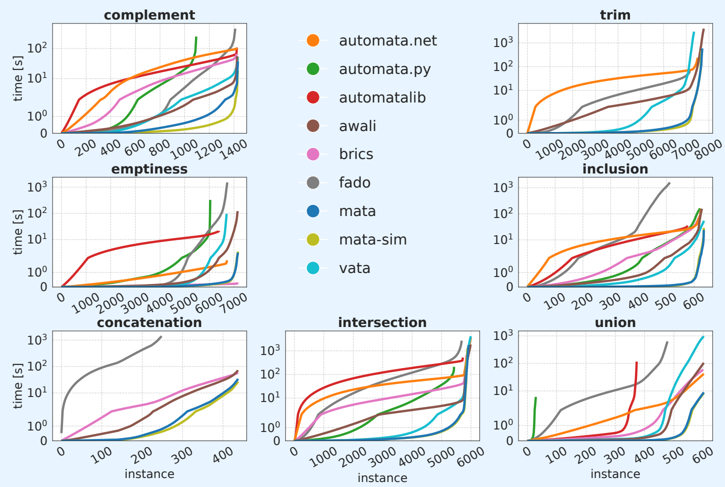
Results per Benchmark



	armc-incl (136)				b-smt (384)				email-filter (500)				lia-explicit (169)				lia-symbolic (169)			
	TO	*Avg*	*Med*	*Std*	*TO*	*Avg*	*Med*	*Std*	*TO*	*Avg*	*Med*	*Std*	*TO*	*Avg*	*Med*	*Std*	*TO*	*Avg*	*Med*	*Std*
Mata	**0**	**174**	**2**	1 s	**0**	**1**	**1**	1	**0**	**1**	**~0**	9	**0**	42	**6**	356	**0**	**2**	**2**	6
Awali	7	1 s	17	3 s	**0**	6	6	4	**0**	46	4	162	6	**21**	21	16	**0**	8	7	14
Vata	**0**	324	43	577	**0**	7	7	10	**0**	42	2	322	**0**	121	51	671	1	11	10	11
Automata.net	9	1 s	125	3 s	**0**	148	153	30	**0**	69	66	30	**0**	113	117	49	6	103	107	33
Brics	5	659	34	2 s	4	43	43	19	6	103	17	280	**0**	66	62	63	6	55	60	33
AutomataLib	10	843	669	1 s	7	390	126	3 s	48	516	390	521	**0**	458	285	1 s	6	164	173	52
FAdo	58	8 s	22 s	10 s	9	109	112	67	64	6 s	1 s	11 s	1	1 s	727	2 s	6	135	149	105
Automata.py	10	913	133	3 s	334	24	TO	15	4	520	19	2 s	1	372	167	894	6	35	35	25

	noodler-compl (751)				noodler-conc (438)				noodler-inter (4872)				param-inter (267)				param-union (267)			
	TO	*Avg*	*Med*	*Std*	*TO*	*Avg*	*Med*	*Std*	*TO*	*Avg*	*Med*	*Std*	*TO*	*Avg*	*Med*	*Std*	*TO*	*Avg*	*Med*	*Std*
Mata	**0**	**39**	**~0**	401	**0**	**100**	**10**	286	**0**	**~0**	**~0**	3	**156**	**1 s**	TO	4 s	**0**	**166**	**7**	326
Awali	**0**	73	2	638	**0**	490	55	1 s	6	3	1	7	157	6 s	TO	7 s	**0**	1 s	81	3 s
Vata	**0**	57	2	296		-			2	4	**~0**	22	159	7 s	TO	8 s	14	6 s	270	12 s
Automata.net	**0**	53	39	110		-			**0**	26	24	9	157	8 s	TO	10 s	**0**	220	47	314
Brics	**0**	47	8	190	**0**	136	35	204	**0**	7	3	21	159	6 s	TO	6 s	**0**	223	50	307
AutomataLib	**0**	293	143	793		-			17	276	216	675	227	8 s	TO	13 s	227	10 s	TO	15 s
FAdo	10	646	5	4 s	189	10 s	25 s	13 s	10	271	52	2 s	250	15 s	TO	20 s	115	5 s	12 s	11 s
Automata.py	3	263	5	2 s		-			5	38	3	353	254	4 s	TO	6 s	245	11 s	TO	16 s

Results per Operation



	complement			concatenation			emptiness			inclusion			intersection			trim			union		
	Avg	*Med*	*Std*	*Avg*	*Med*	*Std*	*Avg*	*Med*	*Std*	*Avg*	*Med*	*Std*	*Avg*	*Med*	*Std*	*Avg*	*Med*	*Std*	*Avg*	*Med*	*Std*
Mata	**25**	**1**	315	**78**	**8**	235	**~0**	**~0**	2	**37**	**~0**	576	295	**~0**	3 s	76	**~0**	828	**14**	**~0**	45
Awali	38	2	462	166	22	402	17	**~0**	138	250	2	2 s	312	**~0**	2 s	516	**~0**	4 s	173	**~0**	527
Vata	36	3	294		-		14	**~0**	130	85	1	374	699	**~0**	4 s	408	**~0**	3 s	2 s	**~0**	5 s
Automata.net	73	59	89		-		**~0**	**~0**	~0	245	43	1 s	621	14	4 s	31	9	165	69	6	163
Brics	46	24	140	136	35	204	**~0**	**~0**	~0	204	10	1 s	115	4	1 s		-		99	2	232
AutomataLib	75	31	657		-		3	2	5	60	42	102	91	59	748		-		311	2	3 s
FAdo	320	3	2 s	6 s	10 s	10 s	223	**~0**	2 s	3 s	84	8 s	479	48	3 s	**10**	3	70	1 s	84	6 s
Automata.py	226	25	2 s		-		53	**~0**	1 s	263	6	1 s	**39**	2	479		-		203	TO	377

Auction-Based Scheduling

Guy Avni[1], Kaushik Mallik[2], and Suman Sadhukhan[1]

[1]University of Haifa, Haifa, Israel

[2] Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

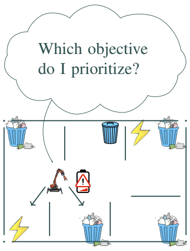
ArXiv link:

Multi-objective Control Problems

Consider a robot in a workspace with the following two objectives:

- *Reach* trash cans (and empty them) whenever they are full.
- *Reach* a charging station before the robot's battery runs out.

The goal: Synthesize a policy for the robot that satisfies both objectives in every run (possibly infinite).



Problem: Given an environment model, like a graph $G = \langle V, E\rangle$, and a pair of LTL objectives Φ_1 and Φ_2 over V, with $\Phi_1 \wedge \Phi_2 \neq$ `False`, synthesize a *policy* (for G, a policy is an infinite path) that satisfies $\Phi_1 \wedge \Phi_2$.

Traditional monolithic approach: Synthesize a policy by treating $\Phi_1 \wedge \Phi_2$ as *a single objective*.

Our decentralized approach: Synthesize *local* policies for Φ_1 and Φ_2 and *compose* them at runtime.

Advantages of the decentralized approach:

- *Modularity*. If only one of the objectives changes, a recomputation of the policy for the other objective may be avoided.
- *Parallel computation*. The local policies, for the given objectives, can be created independently and in parallel—even by different parties.

The Auction-Based Scheduling Framework

The composition of local policies is nontrivial, because the policies may disagree on their actions at any given time. Auction-based scheduling is a novel runtime policy-composition framework, where the policies participate in *auctions* (aka *biddings*) for the privilege of executing their favorite actions.

Tenders: policies augmented with bidding capabilities. Let $G = \langle V, E\rangle$ be a graph and ϕ be an arbitrary objective over V. We distinguish two types of policies, ones that select actions and ones that select bids:

- An *action policy* for ϕ is a function $\alpha : V^* \to V$ that chooses the next vertex for any given finite path. Applying α repeatedly from an initial vertex generates an infinite path that satisfies ϕ.
- A *bidding policy* is a function $\beta : V \times [0, 1] \to [0, 1]$ with the constraint that $\beta(v, B) \leq B$ for every v, B. Intuitively, $\beta(v, B)$ is the proposed bid if the current vertex is v and the current budget is B; the constraint $\beta(v, B) \leq B$ ensures that the bid does not exceed the budget.

A *tender* τ for ϕ is a tuple $\langle \alpha, \beta, \mathbb{B}\rangle$, consisting of an action policy α for ϕ, a bidding policy β, and a real number $\mathbb{B} \in [0, 1]$ called the *threshold budget*.

Each tender requires a sufficient initial budget to be able to bid correctly and "serve" the objective it was designed for. The threshold budget $\mathbb{B}$ is the infimum of the set of sufficient initial budgets; a formal explanation of the role of $\mathbb{B}$ will be provided in $\{*\}$. The heart of our approach is the composition operation on two tenders:

The composition of two tenders. Let $G = \langle V, E\rangle$ be a graph, $\tau_1 = \langle \alpha_1, \beta_1, \mathbb{B}_1\rangle$ and $\tau_2 = \langle \alpha_2, \beta_2, \mathbb{B}_2\rangle$ be two tenders (for a pair of given objectives). The **pre-requisite** for the composition: $\mathbb{B}_1 + \mathbb{B}_2 < 1$. The composition generates an infinite path defined inductively as follows:

- Let $v^0 \in V$ be the initial vertex, and $B_1 > \mathbb{B}_1$ and $B_2 > \mathbb{B}_2$ be the initial budgets allotted to τ_1 and τ_2, respectively, such that $B_1 + B_2 = 1$ (feasible, because of the pre-requisite stated above).
- For each prefix $v^0 \ldots v^k \in V^*$ and for any current budgets B_1, B_2, let $b_1 = \beta_1(v^k, B_1)$ and $b_2 = \beta(v^k, B_2)$ be the two bids proposed by the respective tenders.
  - If $b_1 > b_2$ then τ_1 wins the current round of auction, pays b_1 to τ_2 so that $B_1 \coloneqq B_1 - b_1$ and $B_2 \coloneqq B_2 + b_1$ are the new budgets, and chooses $v^{k+1} = \alpha_1(v^0 \ldots v^k)$ as the next vertex.
  - If $b_2 > b_1$ then τ_2 wins the current round of auction, pays b_2 to τ_1 so that $B_1 \coloneqq B_1 + b_2$ and $B_2 \coloneqq B_2 - b_2$ are the new budgets, and chooses $v^{k+1} = \alpha_2(v^0 \ldots v^k)$ as the next vertex.
  - If $b_1 = b_2$ then it is tie which is resolved in a predetermined way.

$\{*\}$ **The role of threshold budgets.** Let G be a graph and ϕ be an objective. The threshold budget $\mathbb{B}$ guarantees that there exist α and β such that the composition of the tender $\tau = \langle \alpha, \beta, \mathbb{B}\rangle$ with any other tender τ' (fulfilling the pre-requisite) generates an infinite path satisfying ϕ.

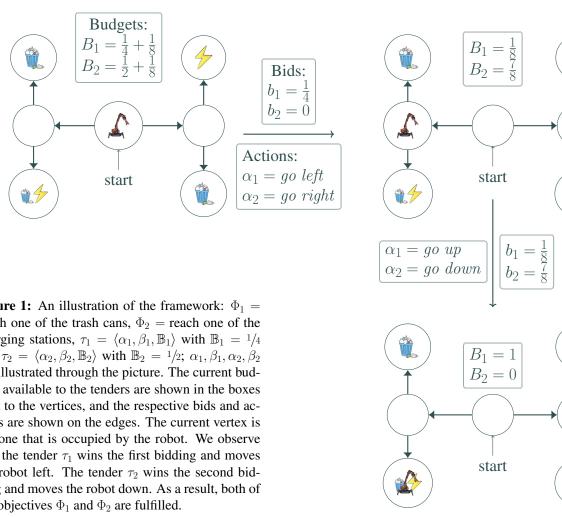


Figure 1: An illustration of the framework: Φ_1 = reach one of the trash cans, Φ_2 = reach one of the charging stations, $\tau_1 = \langle \alpha_1, \beta_1, \mathbb{B}_1\rangle$ with $\mathbb{B}_1 = 1/4$ and $\tau_2 = \langle \alpha_2, \beta_2, \mathbb{B}_2\rangle$ with $\mathbb{B}_2 = 1/2$; $\alpha_1, \beta_1, \alpha_2, \beta_2$ are illustrated through the picture. The current budgets available to the tenders are shown in the boxes next to the vertices, and the respective bids and actions are shown on the edges. The current vertex is the one that is occupied by the robot. We observe that the tender τ_1 wins the first bidding and moves the robot left. The tender τ_2 wins the second bidding and moves the robot down. As a result, both of the objectives Φ_1 and Φ_2 are fulfilled.

Decentralised Synthesis w/ Varying Degrees of Synchronization

The decentralized synthesis problem: Given a graph G and a pair of objectives Φ_1 and Φ_2, synthesize tenders τ_1 and τ_2, respectively for Φ_1 and Φ_2, such that their composition fulfills $\Phi_1 \wedge \Phi_2$.

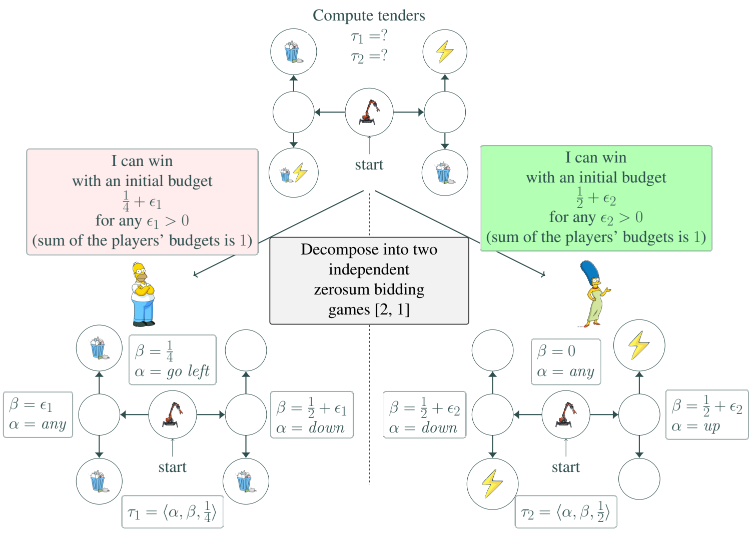
Ideally, the synthesis of τ_1 and τ_2 should be possible in isolation, without the knowledge of the other objective. In practice, this may not be always possible, because the pre-requisite $\mathbb{B}_1 + \mathbb{B}_2 < 1$ may not be achievable. Luckily, the thresholds $\mathbb{B}_1$ and $\mathbb{B}_2$ can be lowered by incorporating some additional *assumptions* about the other tender. Based on the strength of the assumption, we consider three classes of decentralized synthesis problems; they are listed below in the order of strengths of the assumptions:

Strong < Assume-Admissible < Assume-Guarantee

Strong Synthesis: Assume the *Worst Case* (Weakest Assumption)

Advantage: Complete modularity: Each tender remains valid no matter how the other objective is altered.

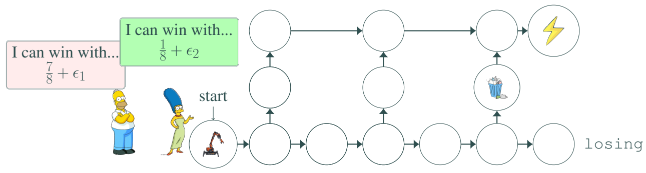
Figure 2: The gist of the algorithm for strong synthesis: We solve two independent zerosum bidding games on the same graph with the individual objectives. The solution of the respective game provides the respective tender. In the two bidding games, it can be shown that the protagonists—Homer and Marge—can win against any adversary with initial budgets strictly greater than $1/4$ and $1/2$, respectively. Their respective winning strategies provide us the required $\alpha_1, \beta_1, \alpha_2, \beta_2$, and the thresholds $1/4$ and $1/2$ provide $\mathbb{B}_1$ and $\mathbb{B}_2$, respectively. The strong synthesis is successful because $\mathbb{B}_1 + \mathbb{B}_2 < 1$.



Theorem: If strong synthesis generates a pair of tenders τ_1 and τ_2 with $\mathbb{B}_1 + \mathbb{B}_2 < 1$, then the composition of τ_1 and τ_2 fulfills $\Phi_1 \wedge \Phi_2$.

The following is an example where strong synthesis fails to generate tenders with $\mathbb{B}_1 + \mathbb{B}_2 < 1$.

Figure 3: Homer and Marge require initial budgets strictly larger than $7/8$ and $1/8$, respectively. Therefore, $\mathbb{B}_1 = 7/8$ and $\mathbb{B}_2 = 1/8$, and $\mathbb{B}_1 + \mathbb{B}_2 \not< 1$.



Assume-Admissible Synthesis: Assume *Rational* (Admissible) Behavior

When strong synthesis fails, we may make the tenders aware of each other's objectives and let them assume that the other tender acts *rationally* towards its own objective. For example, in both local synthesis problems from Fig. 3, the players become aware that the vertex `losing` will not be visited by the other tender if it plays rationally. Therefore, `losing` can be removed from both games, lowering the amounts of required initial budgets (which become $3/4 + \epsilon_1$ and $0 + \epsilon_2$, respectively).

Advantage: Modularity modulo unchanged rational behavior: Each tender τ_i remains valid as long as the rational actions of the other tender τ_{3-i} remain unchanged. In particular, if the other objective Φ_{3-i} remains unchanged, the tender τ_{3-i} can be swapped with a different tender τ'_{3-i} (possibly implementing an alternate policy) and no adjustment in τ_i will be needed.

Theorem: For every graph with maximum out-degree 2 and for every pair of ω-regular objectives, assume-admissible synthesis will have non-empty solutions.

Assume-Guarantee Synthesis: Assume Fulfillment of *Contracts* (Strongest Assumption)

When even assume-admissible synthesis fails, we can use assume-guarantee synthesis where the tenders are synchronized through pre-computed assume-guarantee contracts; the details can be found in the paper.

References

[1] G. Avni and T. A. Henzinger. A survey of bidding games on graphs. In *Proc. 31st CONCUR*, volume 171 of *LIPIcs*, pages 2:1–2:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[2] A. J. Lazarus, D. E. Loeb, J. G. Propp, W. R. Stromquist, and D. H. Ullman. Combinatorial games under auction play. *Games and Economic Behavior*, 27(2):229–264, 1999.

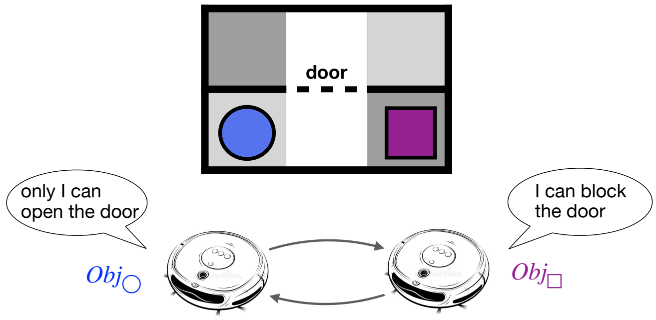
Contact: gavni@cs.haifa.ac.il, kaushik.mallik@ist.ac.at, and ssadhukh@campus.haifa.ac.il

Most General Winning Secure Equilibria

Satya Prakash Nayak and Anne-Kathrin Schmuck

MPI-SWS, Germany

Rational Robots in a Workspace



to win myself, I should open the door infinitely often

$assump_{\bigcirc}$

to win myself, I should not block the door forever

$assump_{\square}$

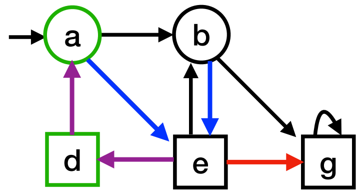
Secure equilibrium = cooperative strategy + punishment strategy

cooperative strategy: alternately use the middle passage

punishment strategy: block the door forever

How to generalize secure equilibria to have more flexibility for the systems?

Rational Players in a Graph Game



○ vs □

$Obj_{\bigcirc}$ = infinitely often **a** $Obj_{\square}$ = infinitely often **d**

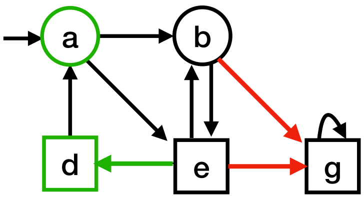
$Goal_{\bigcirc} = (Obj_{\bigcirc}, \neg Obj_{\square})$ $Goal_{\square} = (Obj_{\square}, \neg Obj_{\bigcirc})$

Winning Secure equilibrium (WSE) $(Str_{\bigcirc}, Str_{\square})$

- $(Str_{\bigcirc}, Str_{\square}) \models$ both wins
- $Str_{\bigcirc} \models$ ○ loses $\Rightarrow$ □ loses
- $Str_{\square} \models$ □ loses $\Rightarrow$ ○ loses

cooperative	+	punishment
..a → e ..b → e	+	
..e → d ..d → a	+	..e → g ..d → a

Generalizing Winning Secure Equilibria



○ vs □

$Obj_{\bigcirc}$ = infinitely often **a** $Obj_{\square}$ = infinitely often **d**

$assump_{\bigcirc}$ = never (**b** → **g**)

$assump_{\square}$ = never (**e** → **g**) ∧ inf (**e**) ⇒ inf (**e** → **d**)

Most General WSE $(\Psi_{\bigcirc}, \Psi_{\square})$

- $\Psi_{\bigcirc} \wedge \Psi_{\square} \equiv Obj_{\bigcirc} \wedge Obj_{\square}$
- each Ψ_i is realizable by Player i
- every $(Str_{\bigcirc}, Str_{\square})$ with $Str_i \models \Psi_i$ forms a **WSE**

$\Psi_{\bigcirc} = assump_{\bigcirc} \wedge (assump_{\square} \Rightarrow Obj_{\bigcirc})$

$\Psi_{\square} = assump_{\square} \wedge (assump_{\bigcirc} \Rightarrow Obj_{\square})$

Contribution

- most general WSE = collection of equilibria as **independently** realizable specifications
- **sound** and **efficient** but incomplete algorithm
- generalized to k-player games (even with **Env**)

Future Works

- extend the notion to other equilibria, e.g., subgame-perfect equilibria
- quantitative settings

Pareto Curves for Compositionally Model Checking String Diagrams of MDPs

Kazuki Watanabe[1,2*], Marck van der Vegt[3*], Ichiro Hasuo[1,2], Jurriaan Rot[3] , Sebastian Junges[3]

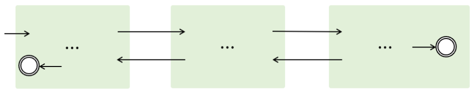
1: National Institute of Informatics, Tokyo, 2: SOKENDAI (The Graduate University for Advanced Studies), Hayama,
3: Radboud University, Nijmegen, the Netherlands
kazukiwatanabe@nii.ac.jp *Equal contribution

Accepted in 30th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2024

Setting: Optimizing Reachability Probabilities of String Diagrams of MDPs
Scheduler Synthesis + Its Performance Guarantee

Target problem

Single-objective problem

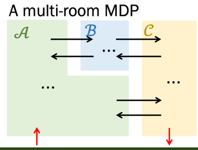


Our target problem: optimization of reachability probabilities for MDPs

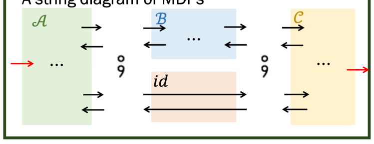
Overview: Compositional Formalism

String diagram of MDPs [Watanabe, Eberhart, Asada, Hasuo, CAV'23]
- A graphical expression with algebraic operations

A multi-room MDP



A string diagram of MDPs



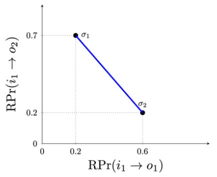
Compositional Exact Algorithm for String Diagrams of MDPs

Open MDP [Watanabe+, CAV'23]

Open MDP $\mathcal{D}$

- Open ends: i_1, i_2, o_1, o_2
- Entrances: i_1, i_2
- Exits: o_1, o_2
- Given an entrance i_1 , choose a scheduler σ that optimizes $RPr^{\sigma}(i_1 \to o_1)$ and $RPr^{\sigma}(i_1 \to o_2)$
  ⇒ Multi-objective optimization!

Pareto curve of $\mathcal{D}$



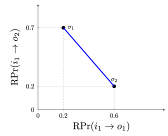
- Every point on the Pareto curve is achievable by a memoryless scheduler.

Overview: Compositional Exact Algorithm [Watanabe+, CAV'23]

Exact Compositional Solution of $\mathcal{D}$ from the specified entrance i_1

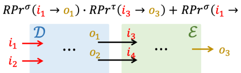
$$\{\text{ scheduler } \sigma \mid RPr^{\sigma}(i_1 \to o_1), RPr^{\sigma}(i_1 \to o_2) \text{ are } \textit{Pareto optimal}\}$$

- Deterministic memoryless schedulers do suffice for the vertices of Pareto curves.
  ⇒ Only finitely many vertices!



Composing solutions of $\mathcal{D}$ and $\mathcal{E}$ from the specified entrance i_1

$$RPr^{\sigma,\tau}(i_1 \to o_3) = RPr^{\sigma}(i_1 \to o_1) \cdot RPr^{\tau}(i_3 \to o_3) + RPr^{\sigma}(i_1 \to o_2) \cdot RPr^{\tau}(i_4 \to o_3)$$



- called decomposition equalities

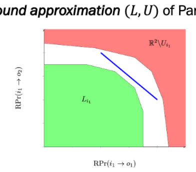
Main Contribution: Compositional Approximation Algorithm for String Diagrams of MDPs

Overview: Compositional Approximation Algorithm (New!)

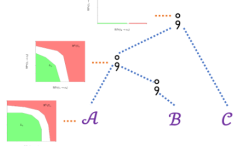
Problem on compositional exact algorithm:
- Exponentially many vertices on Pareto curves
  ⇒ too costly to compute

Our Approach

Sound approximation (L, U) of Pareto curve



Composing sound approximations with ensuring a given precision η



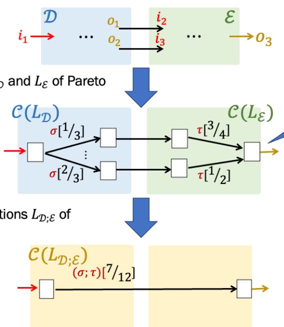
Composing Sound Approximations

Rem: Same for over-approximations

Given under-approximations $L_{\mathcal{D}}$ and $L_{\mathcal{E}}$ of Pareto curve

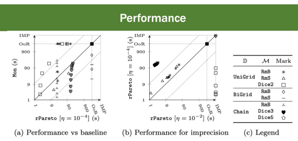
Shortcut open MDPs $\mathcal{C}(L_{\mathcal{D}})$ and $\mathcal{C}(L_{\mathcal{E}})$

Compute an under-approximations $L_{\mathcal{D};\mathcal{E}}$ of the composition $\mathcal{C}(L_{\mathcal{D}}); \mathcal{C}(L_{\mathcal{E}})$



Experiments and Related Work:

Performance



(a) Performance vs baseline (b) Performance for imprecision (c) Legend

- Implementation on top of the model checker Storm [Hensel, Junges, Katoen, Quatmann, Volk, Int. J. Softw. Tools Technol. Transf.'22]
- Reuse of approximations for different occurrences
- Create 8 benchmarks families with 50 different instances
- Setup: Intel i9-10980XE processor, 16GB, TO = 15 minutes
- Baseline: monolithic algorithm (optimistic value iteration)
- Show performance for imprecision (error bounds)

Discussion

Table 1: Benchmark details (time in s, MO=memory out, TO=time out)

$\mathbb{D}$	$\mathcal{M}$	$\lvert S_{[\![\mathbb{D}]\!]}\rvert$	$\#\mathcal{A}$	$\lvert S_{\mathcal{A}}\rvert$	Mon t	Mon t_m	Prec t	rPareto [$\eta = 10^{-2}$] t	t_m	E	p	rPareto [$\eta = 10^{-4}$] t	t_m	E	p
UniGrid100	RmB	1.1e+08	14	148582	MO	MO	MO	17	2	7e-11	58	32	2	4e-12	155
UniGrid200	RmB	4.2e+08	14	148582	MO	MO	MO	37	2	2e-19	58	84	2	1e-20	155
UniGrid250	RmB	6.6e+08	14	148582	MO	MO	MO	58	2	5e-23	58	141	2	3e-24	155
UniGrid100	Dice	5.4e+07	14	74424	170	41	MO	5	<1	3e-01	56	7	<1	2e-04	94
UniGrid200	Dice	2.1e+08	14	65424	MO	MO	MO	25	<1	8e-01	56	45	<1	5e-04	94
UniGrid250	Dice	3.3e+08	14	74424	MO	MO	MO	46	<1	1e+00	56	88	<1	7e-04	94
Chain1000	RmB	1.6e+07	6	42463	181	16	MO	11	<1	2e-50	44	22	<1	6e-52	82
Chain2000	RmB	2.1e+07	6	42463	258	24	MO	11	<1	3e-68	44	22	<1	7e-70	82
BiGrid100	RmS	8.5e+05	16	1353	19	1	TO	TO	TO	TO	TO	TO	TO	TO	TO
BiGrid200	RmS	3.4e+06	16	1352	145	5	TO	TO	TO	TO	TO	TO	TO	TO	TO

Ours

- Baseline: monolithic (Mon) and exact algorithm (Prec) [Watanabe et al., CAV'23]
- Overall, experiments show that our compositional approximation algorithm can solve a big MDP, where the number of state is more than 10^8
- Disadvantage: increasing number of exits can significantly worsen the performance
  ⟹ keeping number of objectives small is important for performance.

Related Work

Compositional Approach for Sequential Composition ;
- Widely studied: [Barry et al., IJCAI'11], [Jothimurugan et al., NeurIPS'21], [Junges & Spaan, CAV'22], [Neary et al., AAAI'22], [Watanabe et al., CAV'23], etc.
- Our approach: composing approximation of Pareto curves
- Many of them study expected rewards

Probabilistic Model Checking wrt. Parallel Composition ∥ [Kwiatkowska et al., Inf. Comp. '13]
- Compositional model checking of parallel composition $\mathcal{A} \parallel \mathcal{B}$
- Using Pareto curves for obtaining sound approximations
- Assume-guarantee "contracts" betw. $\mathcal{A}$ and $\mathcal{B}$ must be devised

Sequential Value Iteration [Hahn & Hartmanns, SETTA'16], [Hartmanns et al., J. Autom. Reason. '20]
- Essentially rely on unidirectional composition
- Similar to the topological value iteration
- Our approach can work with bidirectional composition

VeSCMul: Verified Implementation of S-C-Rewriting for Multiplier Verification

Mertcan Temel

mert.temel@intel.com

Intel Corporation

2024 TACAS

intel.

Abstract

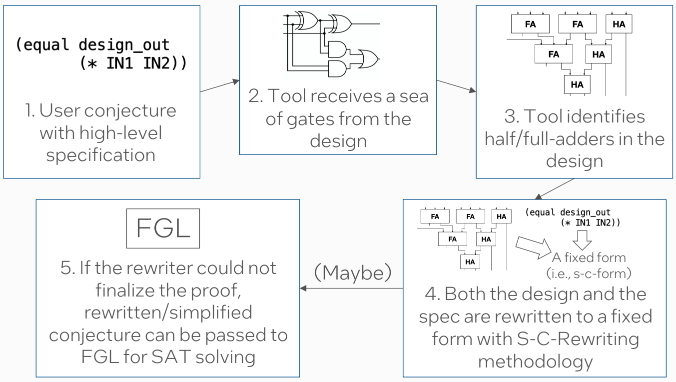
Formal verification of multipliers, especially industrial designs, is difficult. We use the **S-C-Rewriting** method to efficiently verify a variety of multiplier-centric hardware designs. This work presents a custom tool, **VeSCMul**, that packs this method and other tools for easy verification of RTL multipliers. VeSCMul is fully verified itself, very fast, and compatible with industrial designs.

What is S-C-Rewriting?

- A custom term-rewriting method for multipliers: a set of rewrite rules convert both the RTL expressions and high-level specification to the same final form.
- Developed for industrial designs: method supports many configurations such as shifted, truncated, saturated outputs; multiply, multiply-add, dot product…
- Very fast & scales well: 64x64-bit multipliers are verified in seconds, 1024x1024-bit in minutes (much faster than any other method).
- Reliable verification results: soundness proofs are done through ACL2 theorem prover and programming language.
- Caveat: requires separation of multiplier's adder components from the rest of the circuit design components.

What is VeSCMul?

VeSCMul is a tool that implements S-C-Rewriting, and an adder detection program for full automation. It works with other utilities to support verification of complex Verilog designs.



1. Based on target design, user states the conjecture to prove.
2. Included tools (ACL2's SV/SVTV) parses Verilog code and creates flattened symbolic simulation vectors.
3. As S-C-Rewriting depends on adder separation, the tool automatically finds and marks adders.
4. S-C-Rewriting is employed to rewrite both the design and spec to the same form.
5. If rewriting does not finalize the correctness proof, rewritten form may be passed to another tool (FGL) for finalizing the proof or counterexample generation.

VeSCMul Demo

VeSCMul is open-source and distributed with public ACL2 (interactive theorem prover). Events to verify a 64x64-bit multiplier:

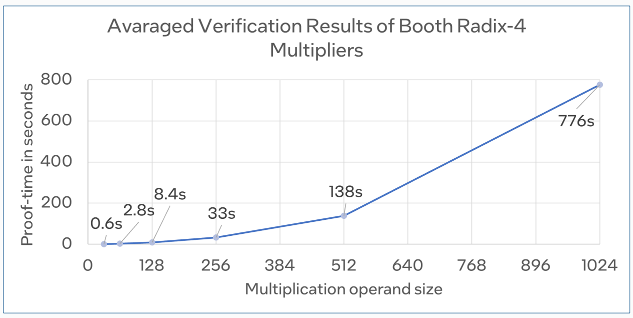
```
(include-book "projects/vescmul/top" :dir :system)

(vescmul-parse
  :name my-multiplier-example
  :file "DT_SB4_HC_64_64_multgen.sv"
  :topmodule "DT_SB4_HC_64_64")

(vescmul-verify
  :name my-multiplier-example
  :concl (equal RESULT
                (loghead 128 (* (logext 64 IN1)
                                (logext 64 IN2)))))
```

- `include-book` event loads VeSCMul and required libraries.
- `vescmul-parse` event parses the target design.
- `vescmul-verify` event attempts to verify the conjecture. `RESULT` is 128-bit wide design output and should be signed multiplication of 64-bit wide inputs `IN1` and `IN2`. `logext` sign-extends, `*` multiplies, `loghead` truncates values. This proof event takes 1-2 seconds and runs fully automatically.

Noteworthy Features

- Ability to state custom conjectures, supporting multiplier variants such as multiply-add, shifted/truncated outputs (vital for industrial designs)
- Fully automatic, only a fraction of target designs requiring manual intervention
- Integration into other verification flows, helpful during more complex tasks such as verification of floating-point designs
- The program itself is fully verified, delivering soundness guarantees of its results

Results



- Tested with 1000s of different design configurations.
- Also got successful results in industrial designs, including verification flow of FP fused multiply-add. Tool helped notably cut down on verification time for new designs.
- Future work includes more testing and further improvements as needed.

Provable Preimage Under-Approximation for Neural Networks

Xiyue Zhang, Benjie Wang and Marta Kwiatkowska
Department of Computer Science, University of Oxford

*An **anytime, scalable** and **flexible** method for preimage approximation of neural networks, with application to **quantitative verification.***

Background

Characterizing the preimage symbolically allows us to perform more complex analysis for **a wider class of properties** beyond local robustness, such as computing the proportion of inputs satisfying a property (quantitative verification) even if standard robustness verification fails.

Methods

Preimage approximation with provable guarantees:

1. **Efficient** input bounding plane generation
2. **Refinement** algorithm with novel input-split and ReLU-split methods
3. **Optimization** of convex bounding functions for tighter preimage approximation

1

Symbolic lower/upper bounding functions from output to input: $\underline{b} - \underline{A}x \leq f(x) \leq \overline{b} - \overline{A}x$

- **under-approximation** in the form of polytope: $\{x \mid \underline{b} - \underline{A}x \geq 0\} \longrightarrow \{x \mid f(x) \geq 0\}$

2

Refinement via splitting plane

- split the domain into subdomains to derive **tighter** preimage polytope over the subdomain
- the preimage is the **union** of the polytopes $\bigcup_{k\in[1,N]} \{x: b_k - A_k x \geq 0\}$

Refinement via naïve splitting is infeasible

Q1. How to **prioritize** which leaf **subregion** to split?

Region search strategy: $vol(\mathcal{C}_1) - vol(\underline{\mathcal{C}_1}) > vol(\mathcal{C}_2) - vol(\underline{\mathcal{C}_2})$ ✓

Q2. How to identify the best **splitting plane**?

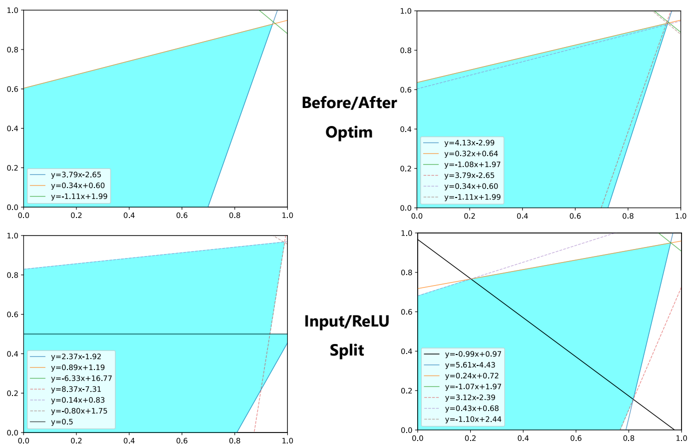
Greedy method: $vol(T(\mathcal{C}_1)) + vol(T(\mathcal{C}_2)) > vol(T(\mathcal{C}_1')) + vol(T(\mathcal{C}_2'))$ ✓

3

Optimize polytope volume via gradient descent

- The optimization problem over $\boldsymbol{\alpha}$ for K specifications

$$\max_{0\leq\boldsymbol{\alpha}\leq 1} \int_{x\in\mathcal{C}} \mathbb{1}_{\min_{i\in[1,K]} \underline{f_i}(x,\boldsymbol{\alpha}_i)\geq 0}\, dx$$



Result 1: Comparison with SOTA methods

Models	Exact		Invprop		Our		
	#Poly	Time(s)	Time(s)	Cov(%)	#Poly	Time(s)	Cov(%)
Vehicle Parking	10	3110.979	2.642	92.1	4	1.175	95.7
VCAS (avg.)	131	6363.272	-	-	12	11.281	91.0

L_∞ attack	#Poly	Cov(%)	Time(s)	Patch attack	#Poly	Cov(%)	Time(s)
0.05	2	100.0	3.107	3×3(center)	1	100.0	2.611
0.07	247	75.2	121.661	4×4(center)	678	38.2	455.988
0.08	522	75.1	305.867	6×6(corner)	2	100.0	9.065
0.09	733	16.5	507.116	7×7(corner)	7	84.2	10.128

- Orders-of-magnitude improvement in **efficiency**
- Preimage in the form of **disjoint polytope union**
- **Splitting method** designed for preimage abstraction
- **Scalability** to high-dimensional inputs

Result 2: Comparison with robustness verifiers

Task	α,β-CROWN		Our		
	Result	Time(s)	Cov(%)	#Poly	Time(s)
Cartpole ($\dot\theta \in [-1.642, -1.546]$)	yes	3.349	100.0	1	1.137
Cartpole ($\dot\theta \in [-1.642, 0]$)	no	6.927	94.9	2	3.632
MNIST (L_∞ 0.026)	yes	3.415	100.0	1	2.649
MNIST (L_∞ 0.04)	unknown	267.139	100.0	2	3.019

- Provide **quantitative** results when the safety property does not hold.

Scan me for full text

fun2model

erc European Research Council Established by the European Commission

elsa European Lighthouse on Secure and Safe AI

UNIVERSITY OF OXFORD

Dissipative quadratizations of polynomial ODE systems

Yubo Cai [1] Gleb Pogudin [2]

[1]École Polytechnique [2]LIX, CNRS, École Polytechnique

Quadratization: What?

Consider a system in $\bar{x} = (x_1, \ldots, x_n)$:

$$\begin{cases} x_1' = f_1(\bar{x}), \\ \ldots \\ x_n' = f_n(\bar{x}), \end{cases} \quad \text{where } f_1, \ldots, f_n \in \mathbb{C}[\bar{x}].$$

New variables $y_1 = g_1(\bar{x}), \ldots, y_m = g_m(\bar{x})$ are called quadratization if there exist $h_1, \ldots, h_{m+n} \in \mathbb{C}[\bar{x}, \bar{y}]$, $\deg h_1, \ldots, \deg h_{m+n} \leqslant 2$ such that

$$\begin{cases} x_1' = h_1(\bar{x}, \bar{y}), \\ \ldots \\ x_n' = h_n(\bar{x}, \bar{y}) \end{cases} \quad \text{and} \quad \begin{cases} y_1' = h_{n+1}(\bar{x}, \bar{y}) \\ \ldots \\ y_m' = h_{n+m}(\bar{x}, \bar{y}) \end{cases}$$

Toy example

$$\begin{matrix} x' = x^4 \\ (\text{degree} = 4) \end{matrix} \xrightarrow{\text{introduce } y := x^3} \begin{matrix} \begin{cases} x' = xy \\ y' = 3x'x^2 = 3x^6 \end{cases} \\ (\text{degree} \leqslant 2) \end{matrix}$$

Quadratization: Why?

- Synthesis of chemical reaction networks:

  deg $\leqslant 2 \iff$ bimolecular network
- Reachability analysis: explicit error bounds for Carleman linearization in the quadratic case.
- Moder Order Reduction (MOR)

Research objectives

How to design a quadratization algorithm that preserves the **numerical** properties of the original system and ensures the **computational efficiency** of the algorithm.

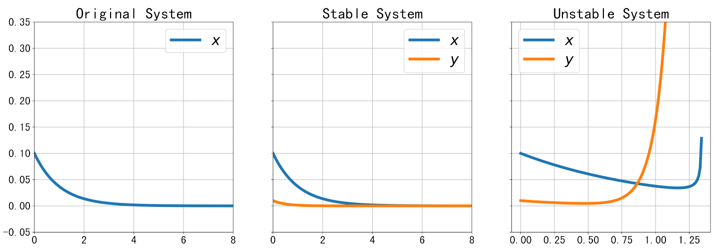


Figure 1. Plot of the following systems with initial condition $\mathcal{X}_0 = [x_0, y_0 = x_0^2] = [0.1, 0.01]$.

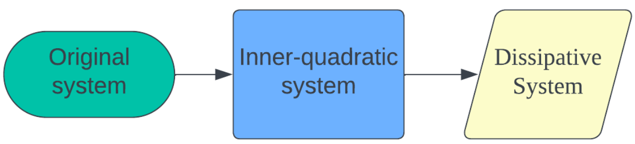
The third system is unstable and diverges in numerical integral!

Original: $x' = -x + x^3 \quad \Leftrightarrow$

Stable: $\begin{cases} x' = -x + xy \\ y' = -2y + 2y^2 \end{cases} \quad \Leftrightarrow$

Unstable: $\begin{cases} x' = -x + xy \\ y' = -2y + 2y^2 + 12\,(y - x^2) = 10y - 12x^2 + 2y^2 \end{cases}$

Our Methodology



We define a system of differential equations

$$\mathbf{x}' = \mathbf{p}(\mathbf{x}), \tag{1}$$

where $\mathbf{x} = \mathbf{x}(t) = (x_1(t), \ldots, x_n(t))$ is a vector of unknown functions and $\mathbf{p} = (p_1, \ldots, p_n)$ is a vector of n-variate polynomials $p_1, \ldots, p_n \in \mathbb{R}[\mathbf{x}]$.

Definition 1 (Equilibrium). For a polynomial ODE system (1), a point $\mathbf{x}^* \in \mathbb{R}^n$ is called an *equilibrium* if $\mathbf{p}(\mathbf{x}^*) = 0$.

Definition 2 (Dissipativity). An ODE system (1) is called *dissipative* at an equilibrium point $\mathbf{x}^*$ if all the eigenvalues of the Jacobian $J(\mathbf{p})|_{\mathbf{x}=\mathbf{x}^*}$ of $\mathbf{p}$ and $\mathbf{x}^*$ have negative real part. It is known that a system which is dissipative at an equilibrium point $\mathbf{x}^*$ is *asymptotically stable* at $\mathbf{x}^*$.

Examples of our methods

Consider the following differential equation:

$$x' = -x(x-1)(x-2)$$

- System's equilibria: $0, 1, 2$
- Dissipative equilibria $x = 0$ and $x = 2$

Inner-quadratic quadratization: introduce $y = x^2$

$$\begin{cases} x' = -xy + 3x^2 - 2x, \\ y' = -2y^2 + 6xy - 4x^2 - \lambda(y - x^2) \end{cases}$$

Dissipative quadratization: append **stabilizer** $h(x, y) = y - x^2$ into the inner-quadratic system with scalar parameter λ

$$\Sigma_\lambda = \begin{cases} x' = -xy + 3x^2 - 2x, \\ y' = -2y^2 + 6xy - 4x^2 - \lambda(y - x^2) \end{cases}$$

Jacobian matrix of the above system:

$$J = \begin{bmatrix} -y + 6x - 2 & -x \\ 6y + 2\lambda x - 8x & -4y - \lambda + 6x \end{bmatrix}$$

For $\lambda = 1, 2, 4, 8, \ldots$ we check the eigenvalues of its Jacobian at points $(0, 0)$ and $(2, 4)$:

λ	at $(0,0)$	at $(2,4)$
1	$-2,\ -1$	$-2,\ 3$
2	$-2,\ -2$	$-2,\ 2$
4	$-2,\ -4$	$-2,\ 0$
8	$-2,\ -8$	$-2,\ -4$

Table 1. Eigenvalues of the Jacobian of Σ_λ

Applications

- Reachability analysis with Carleman linearization.
- Preserving bistability.
- Coupled Duffing oscillators.

More information

- Paper: https://arxiv.org/abs/2311.02508
- Code: https://github.com/yubocai-poly/DQbee

Figure 2. Paper

Figure 3. Code

Z3-Noodler: An Automata-based String Solver

Yu-Fang Chen[1], David Chocholatý[2], Vojtěch Havlena[2], Lukáš Holík[2], Ondřej Lengál[2], and Juraj Síč[2]

[1]Academia Sinica, Taipei, Taiwan [2]Brno University of Technology, Brno, Czech Republic

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

EAPLS Artifacts Available V1.1 — EAPLS Artifacts Evaluated Reusable V1.1

Highlight

- string solver for **quantifier-free theory of strings** (`QF_S`, `QF_SLIA`)
- based on SMT solver Z3 and heavily using **nondeterministic finite automata**
- **stabilization-based procedure** for (dis)equalities with lengths and regular constraints
- support of predicates/functions defined by SMT-LIB
- tailored for **regex-intensive** and **equation-intensive** formulae

Motivation

```
let x = y.substring(1,
        y.length - 1);
let z = y.concat(x);
assert(x === z);
```

$x_0 = substr(y, 1, |y| - 1)$
$\land\ z_0 = y.x_0$
$\land\ x_0 \neq z_0$

Symbolic execution of string programs

```
{action: deactivate,
resource: (a1, a2),
condition:
    {StringLike,
     s3:prefix, home*}
}
```

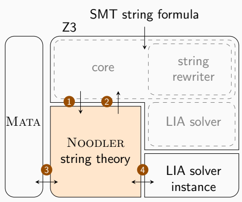
$A = \text{"deactivate"}$
$\land\ (R = \text{"a1"} \lor R = \text{"a2"})$
$\land\ prefix \in home^*$

Amazon cloud access control policies

Architecture

- replacement of Z3's string theory
- **SMT-LIB format** of input formulae
- modified **string theory rewriter** (rules beneficial for the stabilization)

❶ string theory **assignment** (conjunction of (dis)equalities, regular constraints, predicates)
❷ **theory lemma** (including LIA constraints)
❸ **Mata** library for efficient handling of NFAs
❹ internal **LIA solver** for checking lengths constraints



String Theory Core

Axiom saturation

- length-aware string axioms: $|t_1.t_2| = |t_1| + |t_2|$
- axioms for string predicates/functions: $\neg$`contains`$(s, \text{"abc"})$ to $s \notin \Sigma^* abc \Sigma^*$
- different saturation for predicates with concrete values

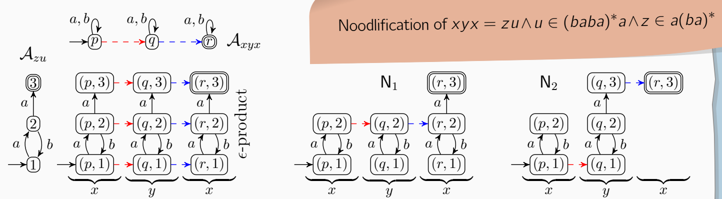
Preprocessing

- transforming the string constraint to a **suitable form**
- tailored for the particular decision procedure
- simple equations converted to regular constraints
- smart **underapproximation**

Decision procedures

- **stabilization-based procedure**
  - iterative **refinement** of variables' languages
  - based on **noodlification** of NFAs representing variable languages
  - efficient **NFA operations in Mata**; eager **simulation-based reduction**
  - generation LIA constraints describing lengths of stable solutions
  - **lazy generation** of stable solutions
  - complete for **chain-free fragment**

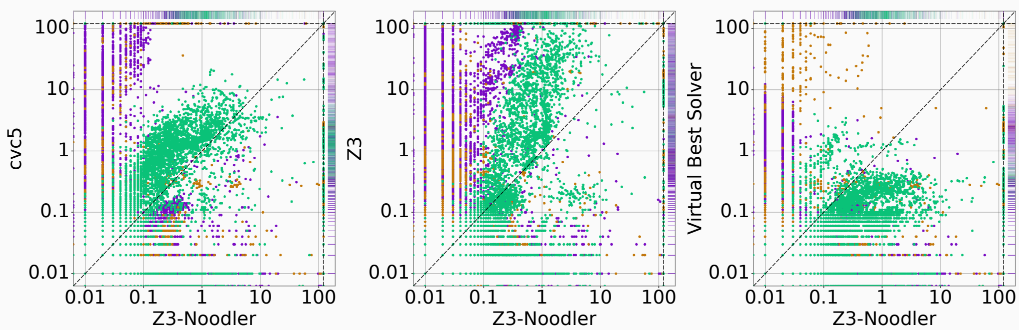
Noodlification of $xyx = zu \land u \in (baba)^* a \land z \in a(ba)^*$



- **Nielsen transformation**
  - **Nielsen graph** construction ⇝ **counter automaton** generation
  - transition saturation of the counter automaton
  - iterative generation of LIA formulae describing paths
  - complete for **quadratic constraints** (no lengths and regular constraints)

Experimental Evaluation

- benchmarks from **SMT-LIB** (`QF_S`, `QF_SLIA`)
- comparison with **SOTA solvers**
- Z3-Noodler v1.1 (TACAS'24 paper was v1.0)
- timeout 120 s, memory limit 8 GiB
  - • **Regex**, • **Equations**, and • **Predicates-small**
- Z3-Noodler **outperforms** other tools on **Regex** and **Equations**
- often **complementary** to other solvers
- great in a solver portfolio
- **extensions**
  - supports **string conversions** (v1.1)
  - support for `replace_all` is in making



Detailed Results

Unsolved cases (smaller is better)

	Regex					Equations								Predicates-small				
	Aut	Den	StrFuzz	Syg	Σ	Kal	Kep	Norn	Slent	Slog	Web	Woo	Σ	StrInt	Leet	StrSm	Σ	PyEx
Included	15995	999	11618	343	28955	19432	587	1027	1128	1976	365	809	25324	16968	2652	1880	21500	23845
Unsupported	0	0	0	0	0	0	0	0	0	0	316	0	316	0	0	0	0	0
Z3-Noodler	60	**0**	**2**	**0**	**62**	270	**3**	**0**	**1**	**0**	**8**	59	**341**	264	4	137	405	94
cvc5	93	18	703	**0**	814	**1**	240	84	24	**0**	47	54	450	**5**	**0**	**19**	**24**	**19**
Z3	125	116	537	**0**	778	284	309	124	73	31	104	**27**	952	239	**0**	59	298	987
Z3str4	60	4	30	**0**	94	174	254	73	73	16	121	78	789	1102	4	60	1166	570
OSTRICH	**48**	6	218	**0**	272	288	387	**0**	126	6	74	53	934	1059	27	173	1259	12833
Z3str3RE	66	27	185	1	279	144	311	133	87	55	192	118	1040	3231	192	259	3682	17764
Z3-Noodler^{pr}	86	1	1982	**0**	2069	508	575	**0**	6	**0**	45	256	1390	1627	29	692	2348	13362

Tool

Available at GitHub

https://github.com/VeriFIT/z3-noodler

SV-COMP and Test-Comp Posters

SoSy-Lab
Software Systems

13th Competition on Software Verification

Dirk Beyer

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Participants

Table 1: Competition candidates with tool references and representing jury members; new for first-time participants, ∅ for hors-concours participation

Participant	Jury member	Affiliation
2ls	V. Malík	BUT, Czechia
aise new	Z. Chen	NUDT, China
BRICK	L. Bu	Nanjing U., China
Bubaak	M. Chalupa	ISTA, Austria
Bubaak-SpLit new	M. Chalupa	ISTA, Austria
CBMC∅	(h. c.)	–
COASTAL∅	(h. c.)	–
CoVeriTeam-AlgSel∅	(h. c.)	–
CoVeriTeam-ParPort∅	(h. c.)	–
CPAchecker	D. Baier	LMU Munich, Germany
CPALockator∅	(h. c.)	–
CPA-BAM-BnB∅	(h. c.)	–
CPA-BAM-SMG∅	(h. c.)	–
CPV new	P.-C. Chien	LMU Munich, Germany
Crux∅	(h. c.)	–
CSeq∅	(h. c.)	–
Dartagnan	H. Ponce de León	Huawei Dresden, Germany
Deagle	F. He	Tsinghua U., China
DIVINE∅	(h. c.)	–
EBF	F. Aljaafari	U. of Manchester, UK
EmergenTheta new	L. Bajczi	BME Budapest, Hungary
ESBMC-incr∅	(h. c.)	–
ESBMC-kind	F. Brauße	U. Manchester, UK
Frama-C-SV	M. Spiessl	LMU Munich, Germany
Gazer-Theta∅	(h. c.)	–
GDart	F. Howar	TU Dortmund, Germany
GDart-LLVM∅	(h. c.)	–
Goblint	S. Saan	U. Tartu, Estonia
Graves-CPA∅	(h. c.)	–
Graves-Par∅	(h. c.)	–
Infer∅	(h. c.)	–
Java-Ranger∅	(h. c.)	–
JayHorn	H. Mousavi	U. Tehran, TIAS, Iran
JBMC	P. Schrammel	U. Sussex / Diffblue, UK
JDart∅	(h. c.)	–
Korn	G. Ernst	LMU Munich, Germany
Lazy-CSeq∅	(h. c.)	–
LF-checker∅	(h. c.)	–
Locksmith∅	(h. c.)	–
MLB	L. Bu	Nanjing U., China
Mopsa	R. Monat	Inria and U. Lille, France
PeSCo-CPA∅	(h. c.)	–
PIChecker∅	(h. c.)	–
Pinaka∅	(h. c.)	–
PredatorHP	V. Šoková	BUT, Czechia
Proton new	R. Metta	TCS, India
SPF∅	(h. c.)	–
sv-sanitizers new	S. Saan	U. of Tartu, Estonia
SWAT new	N. Loose	U. of Luebeck, Germany
Symbiotic	M. Jonáš	Masaryk U., Czechia
Theta	L. Bajczi	BME Budapest, Hungary
UAutomizer	M. Heizmann	U. Freiburg, Germany
UGemCutter	D. Klumpp	U. Freiburg, Germany
UKojak	F. Schüssele	U. Freiburg, Germany
UTaipan	D. Dietsch	U. Freiburg, Germany
VeriAbs	P. Darke	TCS, India
VeriAbsL	P. Darke	TCS, India
VeriOover∅	(h. c.)	–

Features

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

Verifier	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
2ls				✓	✓			✓	✓		✓						✓			
aise new			✓																	
BRICK	✓		✓	✓				✓								✓				
Bubaak			✓								✓					✓	✓			✓
Bubaak-SpLit new			✓		✓						✓				✓	✓	✓		✓	✓
CBMC∅				✓							✓					✓				
COASTAL∅			✓																	
CVT-AlgoSel∅	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		✓	✓
CVT-ParPort∅	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
CPAchecker	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		✓	✓
CPALockator∅	✓	✓					✓				✓	✓	✓	✓		✓				
CPA-BAM-BnB∅	✓	✓					✓				✓	✓	✓	✓						
CPA-BAM-SMG∅																				
CPV new				✓	✓	✓								✓						
Crux∅			✓																	
CSeq∅				✓							✓					✓				
Dartagnan				✓							✓					✓				
Deagle				✓												✓				
DIVINE∅			✓				✓				✓					✓			✓	✓
EBF				✓																
EmergenTheta new				✓	✓						✓	✓		✓					✓	✓
ESBMC-incr∅				✓	✓						✓					✓				
ESBMC-kind				✓	✓		✓	✓			✓					✓				
Frama-C-SV								✓												
Gazer-Theta∅	✓	✓		✓			✓				✓	✓	✓	✓						✓
GDart			✓								✓									✓
GDart-LLVM∅			✓								✓									
Goblint								✓								✓			✓	
Graves-CPA∅	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
Graves-Par∅																				
Infer∅								✓	✓	✓										✓
Java-Ranger∅			✓								✓									
JayHorn	✓	✓				✓		✓					✓	✓						
JBMC				✓							✓					✓				
JDart∅			✓								✓									✓
Korn		✓	✓				✓													✓
Lazy-CSeq∅				✓							✓					✓				
LF-checker∅																				
Locksmith∅																✓				
MLB			✓								✓									✓
Mopsa								✓												
PeSCo-CPA∅	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
PIChecker∅	✓	✓									✓	✓		✓		✓			✓	
Pinaka∅			✓	✓							✓									
PredatorHP									✓											
Proton new				✓																
SPF∅			✓						✓							✓				
sv-sanitizers new																✓				
SWAT new			✓																	
Symbiotic			✓		✓			✓	✓		✓					✓				✓
Theta	✓	✓					✓				✓	✓		✓		✓			✓	✓
UAutomizer	✓	✓									✓		✓	✓	✓	✓	✓		✓	✓
UGemCutter	✓	✓									✓		✓	✓	✓	✓			✓	✓
UKojak	✓	✓									✓		✓	✓						
UTaipan	✓	✓					✓	✓			✓		✓	✓	✓	✓			✓	✓
VeriAbs	✓			✓	✓		✓	✓										✓	✓	✓
VeriAbsL	✓			✓	✓		✓	✓										✓	✓	✓
VeriOover∅																				

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

Verifier	CPAchecker	CProver	Esbmc	Jpf	Ultimate	JavaSMT	MathSAT	Cvc4	SMTInterpol	z3	MiniSAT	Apron
2ls		✓									✓	
aise new												
BRICK										✓	✓	
Bubaak										✓		
Bubaak-SpLit new												
CBMC∅		✓									✓	
COASTAL∅				✓								
CVT-AlgoSel∅	✓	✓	✓		✓	✓	✓				✓	
CVT-ParPort∅	✓	✓	✓		✓	✓	✓				✓	
CPAchecker	✓					✓	✓					✓
CPALockator∅	✓					✓	✓					
CPA-BAM-BnB∅	✓					✓	✓					
CPA-BAM-SMG∅	✓					✓	✓					
Crux∅										✓		
CSeq∅		✓									✓	
Dartagnan						✓						
Deagle											✓	
DIVINE∅												
EBF			✓				✓					
EmergenTheta new												
ESBMC-incr∅			✓				✓					
ESBMC-kind			✓				✓					
Frama-C-SV												
Gazer-Theta∅												
GDart								✓		✓		
GDart-LLVM∅										✓		
Goblint												✓
Graves-CPA∅	✓					✓	✓					
Graves-Par∅												
Infer∅												
Java-Ranger∅				✓								
JayHorn												
JBMC		✓									✓	
JDart∅				✓				✓		✓		
Korn										✓		
Lazy-CSeq∅		✓									✓	
LF-checker∅												
Locksmith∅												
MLB												
Mopsa												✓
PeSCo-CPA∅	✓					✓	✓					
PIChecker∅	✓					✓	✓		✓			
Pinaka∅												
PredatorHP												
Proton new												
SPF∅				✓								
sv-sanitizers new												
SWAT new												
Symbiotic										✓		
Theta												
UAutomizer					✓		✓	✓	✓	✓		
UGemCutter					✓		✓	✓	✓	✓		
UKojak					✓				✓	✓		
UTaipan					✓		✓	✓	✓	✓		
VeriAbs	✓	✓								✓	✓	
VeriAbsL	✓	✓								✓	✓	
VeriOover∅												

More Information

https://sv-comp.sosy-lab.org/2024/

Reference
D. Beyer. State of the art in software verification and witness validation: SV-COMP 2024. In *Proc. TACAS*, LNCS . Springer, 2024

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

Participant	ReachSafety 17746 points 11346 tasks	MemSafety 3216 points 2135 tasks	ConcurrencySafety 5672 points 3259 tasks	NoOverflows 13044 points 8188 tasks	Termination 4900 points 2354 tasks	SoftwareSystems 5251 points 3313 tasks	FalsificationOverall 8317 points 28700 tasks	Overall 49097 points 31054 tasks	JavaOverall 828 points 587 tasks
2ls	6000	224	0	5976	**1584**	10	1311	10564	
aise new									
BRICK									
Bubaak	3788	1890	11	6465	1481	-1082	-617	12206	
Bubaak-SpLit new	4692	1312	7	-41374	661	**872**	1959	-18177	
CPAchecker	**10084**	1897	2029	**8603**	1195	**784**	**4812**	**21568**	
CPV new	6330								
Dartagnan			**3547**						
Deagle									
EBF			636						
EmergenTheta new	1178								
ESBMC-kind	8364	2077	1853	8272	1048	-1063	2394	17896	
Frama-C-SV				1098					
GDart									**616**
Goblint	2289	1304	2583	7059	890	536		15458	
JayHorn									325
JBMC									**618**
Korn									
MLB									**676**
Mopsa	2241	1516		8063		**2197**			
PredatorHP		**2321**							
Proton new					**3526**				
sv-sanitizers new		290							
SWAT new									566
Symbiotic	7052	**2156**	238	7370	1258	687	**4050**	17192	
Theta	2119		2354						
UAutomizer	6320	**2110**	**3079**	**9497**	**3248**	261	3139	**26396**	
UGemCutter			**3189**						
UKojak	4869	1400	0	7363	0	233	2291	10593	
UTaipan	5751	2014	2655	**9231**	0	351	**3157**	**18042**	
VeriAbs	**10541**								
VeriAbsL	**10735**								
CBMC∅	1269	1330	1229	5771	1125	-2569	-3764	8391	
COASTAL∅									-2752
CVT-AlgoSel∅	2635		41						
CVT-ParPort∅	-6152	1655	911	-17812	1289	-1297	-9118	-7545	
CPA-BAM-BnB∅						-2439			
CPA-BAM-SMG∅		2039				-2804			
CPALockator∅			-4924						
Crux∅	2066			490					
CSeq∅			-12478						
DIVINE∅	4655	298	390	0	0	76	256	3576	
ESBMC-incr∅			542						
Gazer-Theta∅									
GDart-LLVM∅									
Graves-CPA∅	3831					-322	-1538	5470	
Graves-Par∅	876	1627	53	-17650	1256	-2037	-9024	-6731	
Infer∅	-99128		-8289	-73312		-24917			
Java-Ranger∅									398
JDart∅									382
Lazy-CSeq∅			-15024						
LF-checker∅			772						
Locksmith∅									
PeSCo-CPA∅	5814					-76	3247	17315	
PIChecker∅			521						
Pinaka∅	2418			1337	855				
SPF∅									182
VeriOover∅									

Ranking

Table 5: Overview of the top-three verifiers for each category; new for first-time participants, measurements for CPU time and energy rounded to two significant digits.

Rank	Verifier	Score	CPU Time (in h)	CPU Energy (in kWh)	Solved Tasks	Unconf. Tasks	False Alarms	Wrong Proofs
ReachSafety								
1	**VeriAbsL**	**10735**	190	7075	1138		**2**	
2	VeriAbs	10541	190	6720	1032		**1**	
3	CPAchecker	10084	200	6468	286	2		
MemSafety								
1	**PredatorHP**	**2321**	1.2	1823	3	3		
2	Symbiotic	2156	0.77	1855	0		**5**	
3	UAutomizer	2110	62	1637	4			
ConcurrencySafety								
1	**Dartagnan**	**3547**	14	2086	0		**5**	
2	UGemCutter	3189	32	1851	4	1		
3	UAutomizer	3079	28	1791	3		**1**	
NoOverflows								
1	**UAutomizer**	**9497**	62	4532	2			
2	UTaipan	9231	66	4420	11		**1**	
3	CPAchecker	8603	18	5596	192			
Termination								
1	**Proton** new	**3526**	19	1888	126	1		
2	UAutomizer	3248	18	1631	11			
3	2ls	1584	4.2	1167	201			
SoftwareSystems								
1	**Mopsa**	**2197**	15	2030	0			
2	Bubaak-SpLit new	872	0.42	480	163	8		
3	CPAchecker	784	43	1756	71			
FalsificationOverall								
1	**CPAchecker**	**4812**	91	4920	218	10		
2	Symbiotic	4050	27	4281	191	11		
3	UTaipan	3157	33	1602	34	1		
Overall								
1	**UAutomizer**	**26396**	290	13617	114	3	**7**	
2	CPAchecker	21568	320	17968	698	16	**1**	
3	UTaipan	18042	240	11524	71	1	**13**	
JavaOverall								
1	**MLB**	**676**	0.93	484	34			
2	JBMC	618	0.44	424	80			
3	GDart	616	2.6	453	9			

Score Schema

Table 6: Scoring schema for SV-COMP 2024 (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)

Cummulative Score

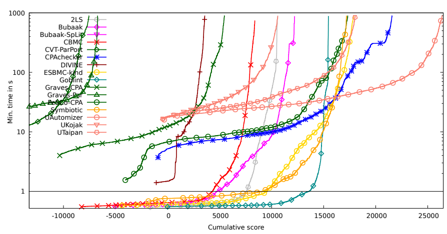


Figure 1: Quantile functions for category *C-Overall*.

SoSy-Lab
Software Systems

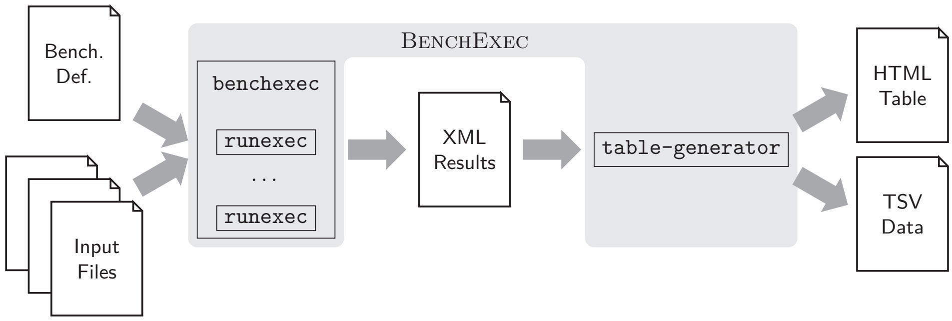
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

BenchExec

Reliable Benchmarking: Requirements and Solutions

Dirk Beyer, Stefan Löwe, and Philipp Wendler

Overview



BenchExec: **A Framework for Reliable Benchmarking and Resource Measurement**

Benchmarking Requirements

1. Measure and Limit Resources Accurately
2. Terminate Processes Reliably
3. Assign Cores Deliberately
4. Respect Nonuniform Memory Access
5. Avoid Swapping
6. Isolate Individual Runs

Scope

- Linux systems
- CPU-bound tool (negligible I/O)
- No use of other resources such as GPUs
- No networking / distributed execution
- No user interaction
- No malicious intent

⇒ **Great for solvers, verifiers, etc.!**

Use Cases

- Low-level command for isolated, limited, and measured execution of a tool
- Integration in other benchmarking frameworks via command line and Python API (**used by StarExec**)
- Benchmarking with large number of runs
- Competition execution (**used e.g. by SV-COMP since 2016**)
- Regression testing

Techniques and Features

Benchmarking containers implemented with Linux features such as

- **Control groups (cgroups)** for resource limitation and measurements (compatible with cgroups v1 and v2)
- **Namespaces** for isolation
- **Overlay filesystem (overlayfs)** for intercepting file writes

(same techniques as used by Docker, etc.)

- Parallel execution of tools
- Automatic calculation of distribution of cores and memory regions
- Knows about NUMA and hyper threading
- Configurable file-system layout in container (hide directories, allow write access, etc.)

Execution

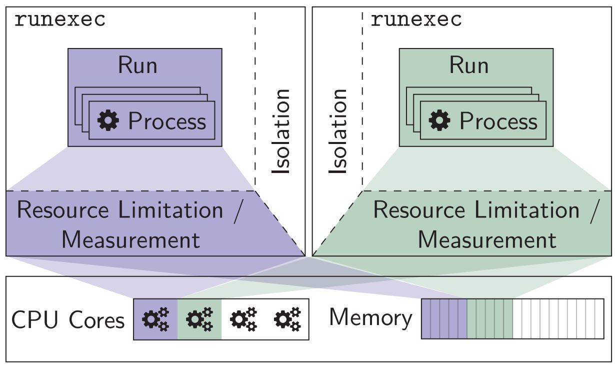
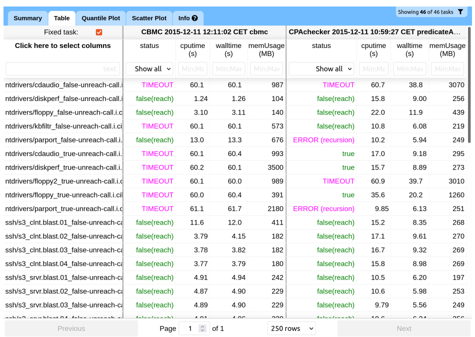
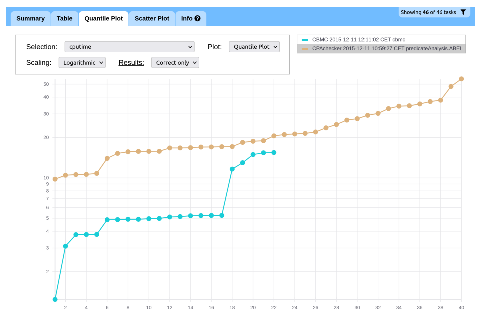


table-generator

- Combine results from several executions
- Define table layout
- Select and filter results
- Compute statistics
- Export raw data as TSV
- Generate interactive tables as stand-alone HTML files
- Quantile and scatter plots
- Live analysis of data

Interactive online example





Paper

- STTT 2017
- Open Access
- DOI 10.1007/s10009-017-0469-y
- Important aspects for benchmarking, hardware influence, how to present results, ...

Thanks to all Contributors!

Aditya Arora, Laura Bschor, Thomas Bunk, Montgomery Carter, Saransh Chopra, Andreas Donig, Karlheinz Friedberger, Peter Häring, Florian Heck, Hugo van Kemenade, George Karpenkov, Mike Kazantsev, Michael Lachner, Thomas Lemberger, Sebastian Ott, Stephan Lukasczyk, Alexander von Rhein, Alexander Schremmer, Dennis Simon, Andreas Stahlbauer, Thomas Stieglmaier, Martin Yankov, Ilja Zakharov, and more (100 in total)!

Tool BenchExec

- License Apache 2.0
- No root access required for benchmarking
- Available on PyPI and `github.com/sosy-lab/benchexec`

CPAchecker

A Tool for Configurable Program Analysis

SoSy-Lab Software Systems

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

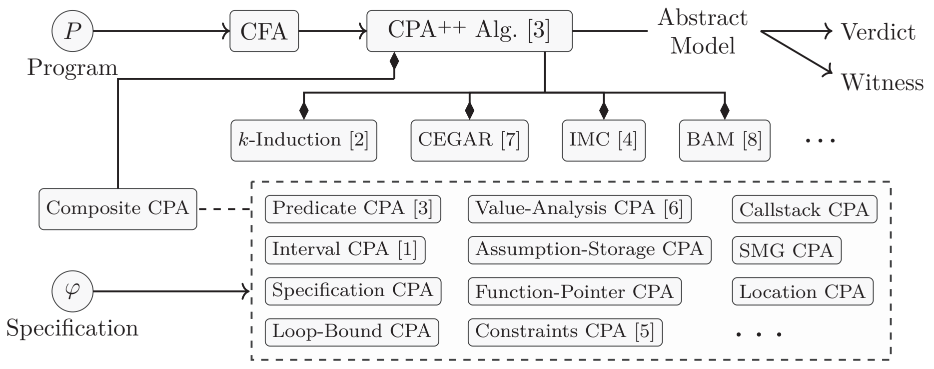
Daniel Baier, Dirk Beyer, Po-Chun Chien, Marek Jankola, Matthias Kettl, Nian-Ze Lee,
Thomas Lemberger, Marian Lingsch-Rosenfeld, Martin Spiessl, Henrik Wachowitz, and Philipp Wendler

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



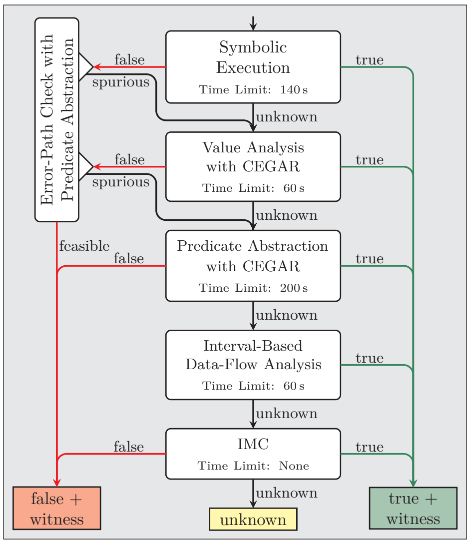
Competition Contribution

"CPAchecker 2.3 with Strategy Selection" is our latest paper describing new developments and configurations used in SV-COMP 2024.

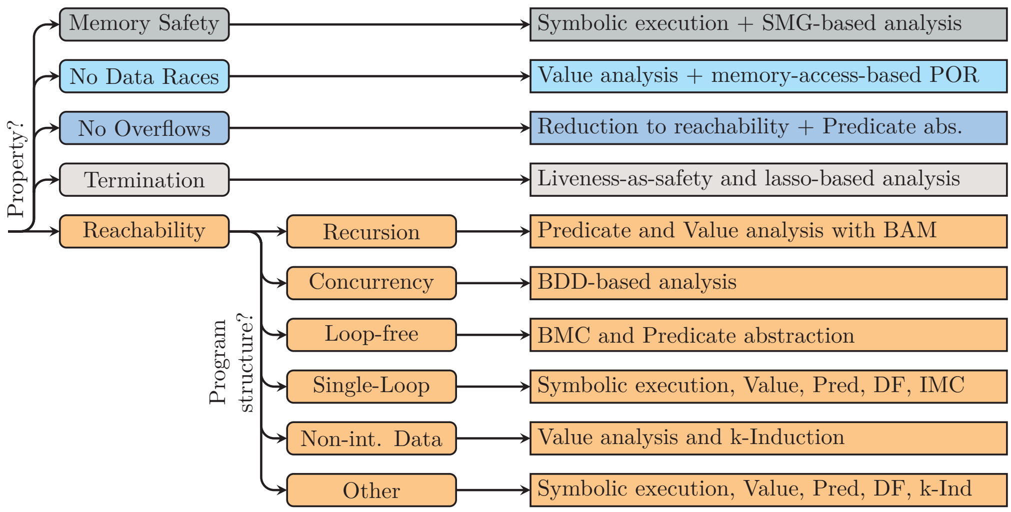
- Utilize strategy selection to predict a sequential portfolio of analyses
- Support all properties and categories of C programs
- 1st place in category *FalsificationOverall*
- 2nd place in category *Overall*
- 3rd place in category *ReachSafety*
- 17 968 validated results in total (the most among all participants)
- Only 17 wrong results (0.06 % of all tasks)
- New and improved analyses for:
  - Reachability
  - Memory safety
  - Termination
  - Overflows
  - Data races

Paper available here

Config for Reachability Single-Loop

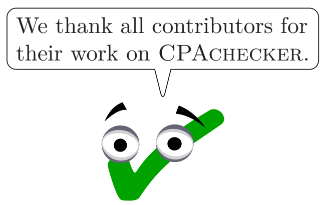


Verification Strategy for SV-COMP 2024



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] Beyer, D., Chien, P.C., Lee, N.Z.: CPA-DF: A tool for configurable interval analysis to boost program verification. In: Proc. ASE. pp. 2050–2053. IEEE (2023). https://doi.org/10.1109/ASE56229.2023.00213

[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. arXiv/CoRR **2208**(05046) (July 2022). https://doi.org/10.48550/arXiv.2208.05046

[5] Beyer, D., Lemberger, T.: Symbolic execution with CEGAR. In: Proc. ISoLA. pp. 195–211. LNCS 9952, Springer (2016). https://doi.org/10.1007/978-3-319-47166-2_14

[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

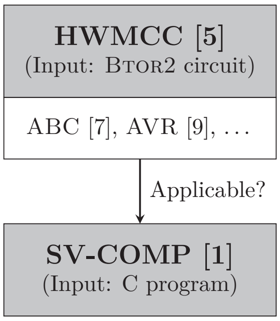
[7] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643

[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

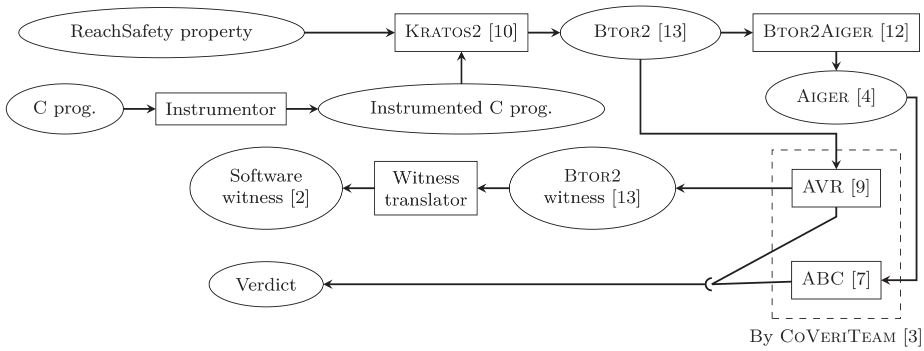
CPV: A Circuit-Based Program Verifier

Po-Chun Chien and Nian-Ze Lee
{po-chun.chien, nian-ze.lee}@sosy.ifi.lmu.de

Motivation



Software Architecture

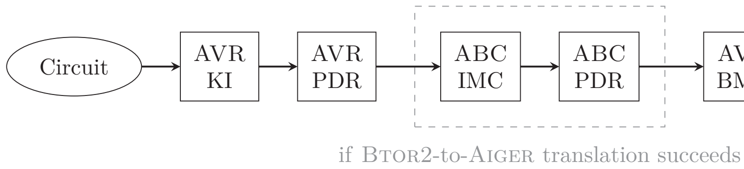


Try CPV!

Artifact DOI: 10.5281/zenodo.10063681

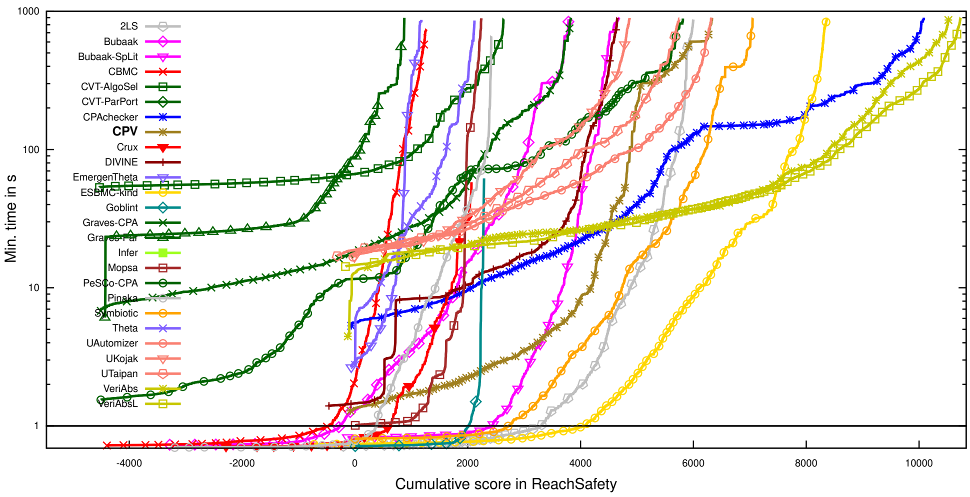
Strategy for SV-COMP 2024

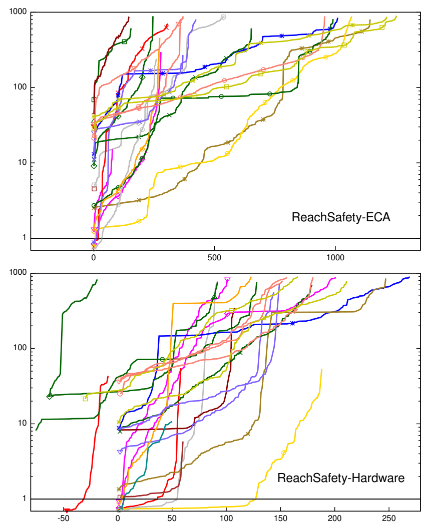
CPV runs a sequential portfolio consisting of property-directed reachability (PDR) [8], interpolation-based model checking (IMC) [11], k-induction (KI) [14], and bounded model checking (BMC) [6].



Evaluation Results at SV-COMP 2024

6th, 3rd, and 2nd place in *ReachSafety*, *ReachSafety-ECA*, *ReachSafety-Hardware*, respectively





Summary

- It is feasible to utilize sequential circuits as intermediate representations for software verification
- CPV can employ different hardware verifiers as the backend
- CPV competed well against other mature verifiers in SV-COMP
- Future work:
  - Support more verification properties (e.g., no-overflow and termination)
  - Export correctness witnesses
  - Incorporate more backend verifiers
  - Apply circuit optimization to improve the performance of verification

References

[1] Beyer, D.: State of the art in software verification and witness validation: SV-COMP 2024. In: Proc. TACAS (2024)

[2] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Lemberger, T., Tautschnig, M.: Verification witnesses. ACM Trans. Softw. Eng. Methodol. **31**(4), 57:1–57:69 (2022)

[3] Beyer, D., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. In: Proc. TACAS. pp. 561–579. LNCS 13243 (2022)

[4] Biere, A.: The AIGER And-Inverter Graph (AIG) format version 20071012. Tech. Rep. 07/1, Institute for Formal Models and Verification, Johannes Kepler University (2007)

[5] Biere, A., Froleyks, N., Preiner, M.: 11th Hardware Model Checking Competition (HWMCC 2020). `http://fmv.jku.at/hwmcc20/`, accessed: 2023-01-29

[6] Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers **58**, 117–148 (2003)

[7] Brayton, R., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: Proc. CAV. pp. 24–40. LNCS 6174 (2010)

[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.: AVR: Abstractly verifying reachability. In: Proc. TACAS. pp. 413–422. LNCS 12078 (2020)

[10] Griggio, A., Jonáš, M.: Kratos2: An SMT-based model checker for imperative programs. In: Proc. CAV. pp. 423–436 (2023)

[11] McMillan, K.L.: Interpolation and SAT-based model checking. In: Proc. CAV. pp. 1–13. LNCS 2725 (2003)

[12] Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Source-code repository of Btor2, BtorMC, and Boolector 3.0. `https://github.com/Boolector/btor2tools`, accessed: 2023-01-29

[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)

ULTIMATE Automizer

ultimate.informatik.uni-freiburg.de

github.com/ultimate-pa/ultimate

Matthias Heizmann, Manuel Bentele, Daniel Dietsch, Xinyu Jiang, Dominik Klumpp, Frank Schüssele, Andreas Podelski

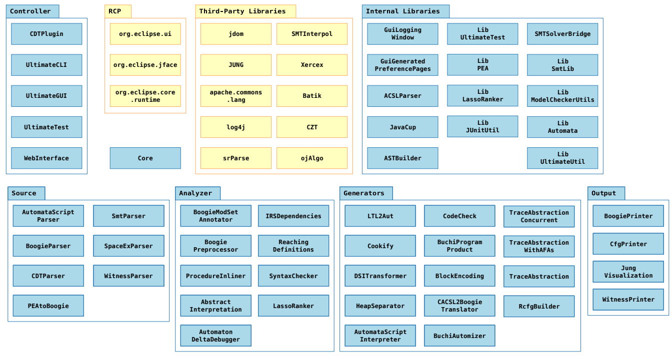
Features

- Memory safety analysis
- Overflow detection
- Termination analysis using Büchi automata
- Nontermination analysis using geometric nontermination arguments
- LTL software model checking
- Bitprecise analysis
- IEEE 754 floating point analysis
- Error witnesses
- Correctness witnesses
- Error localization

Techniques

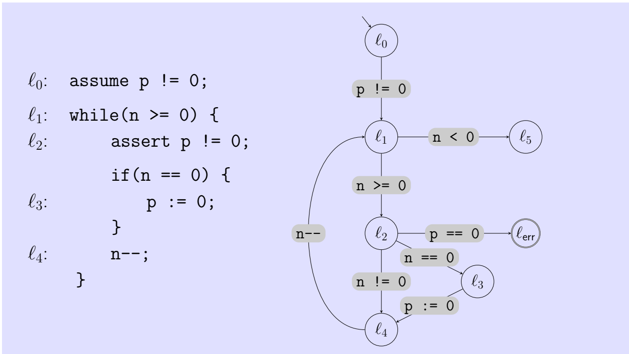
- On-demand trace-based decomposition
- Interprocedural analysis via nested word automata
- Theory-independent interpolation
- Refinement selection
- Configurable block encodings
- Multi SMT solver support
- Synthesis of ranking functions
- Efficient complementation of semi-deterministic Büchi automata
- (Nested word) automata minimization

ULTIMATE program analysis framework



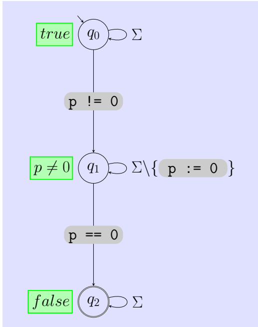
Automata-theoretic proof of program correctness

Program $\mathcal{P}$ is correct because each error trace is infeasible, i.e. the inclusion $\mathcal{P} \subseteq \mathcal{A}_1 \cup \mathcal{A}_2$ holds.



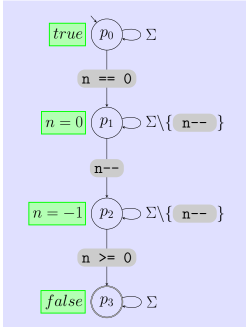
Program / automaton $\mathcal{P}$ whose language is the set of error traces.

$\subseteq$



Automaton $\mathcal{A}_1$ whose language is a set of infeasible traces.

$\cup$



Automaton $\mathcal{A}_2$ whose language is a set of infeasible traces.

- Alphabet: set of program statements $\Sigma = \{$ `p != 0`, `n < 0`, `n >= 0`, `p == 0`, `n == 0`, `n != 0`, `p := 0`, `n --` $\}$
- The language of $\mathcal{P}$ is the set of error traces.
- In the first iteration, we analyze feasibility of the error trace $\pi_1 =$ `p != 0` `n >= 0` `p == 0`. π_1 is infeasible. Via interpolation, we obtain the following Hoare triples.

  $\{true\}$ `p != 0` $\{p \neq 0\}$
  $\{p \neq 0\}$ `n >= 0` $\{p \neq 0\}$
  $\{p \neq 0\}$ `p == 0` $\{false\}$

  We construct the automaton $\mathcal{A}_1$ such that its language is the set of all traces whose infeasibility can be shown using the predicates $true$, $p \neq 0$, and $false$.
- Analogously, in the second iteration the automaton $\mathcal{A}_2$ is constructed.
- We check the inclusion $\mathcal{P} \subseteq \mathcal{A}_1 \cup \mathcal{A}_2$ and conclude that each error trace is infeasible and hence $\mathcal{P}$ is correct.

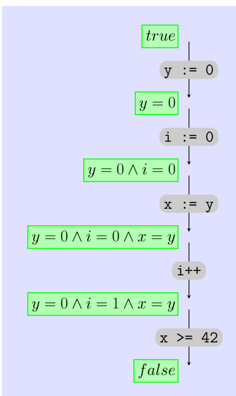
Definition Given an automaton $\mathcal{A} = (Q, \delta, q_{\mathsf{init}}, Q_{\mathsf{final}})$ over the alphabet of program statements, we call a mapping that assigns to each state $q \in Q$ a predicate φ_q a *Floyd-Hoare annotation for automaton $\mathcal{A}$* if the following implications hold.

$$(q, st, q') \in \delta \implies \{\varphi_q\} st \{\varphi_{q'}\} \text{ is a valid Hoare triple}$$
$$q = q_{\mathsf{init}} \implies \varphi_q = true$$
$$q \in Q_{\mathsf{final}} \implies \varphi_q = false$$

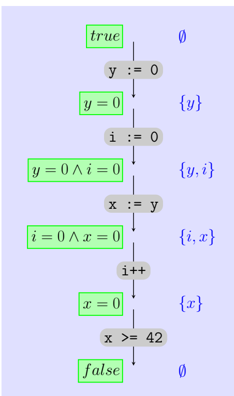
Theorem If an automaton $\mathcal{A}$ has a Floyd-Hoare annotation, then $\mathcal{A}$ recognizes a set of infeasible traces.

Interpolation with unsatisfiable cores

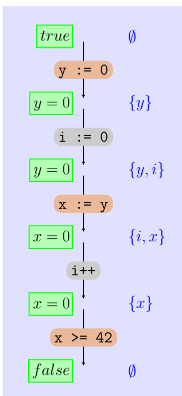
Level 1: "interpolation" via
- strongest post



Level 2: interpolation via
- strongest post
- live variable analysis



Level 3: interpolation via
- strongest post
- live variable analysis
- unsatisfiable cores



Algorithm (for level 3)

- Input: infeasible trace $st_1, \dots, st_n$ and unsatisfiable core $\mathsf{UC} \subseteq \{st_1, \dots, st_n\}$.
- Replace each statement that does not occur in UC by a skip statement or a havoc statement.

  assume statement ψ $\rightsquigarrow$ `skip`
  assignment statement `x:=t` $\rightsquigarrow$ `havoc x`
- Compute sequence of predicates $\varphi_0, \dots, \varphi_n$ iteratively using the strongest post predicate transformer sp.

$$\varphi_0 := true$$
$$\varphi_{i+1} := sp(\varphi_i, st_{i+1})$$

- Eliminate each variable from predicate φ_i that is not live at position i of the trace.
- Output: sequence of predicates $\varphi_0, \dots, \varphi_n$ which is a sequence of interpolants for the infeasible trace $st_1, \dots, st_n$.

Ultimate GemCutter: Commutativity in Concurrent Program Verification

Dominik Klumpp, Daniel Dietsch, Matthias Heizmann, Frank Schüssele, Azadeh Farzan, Andreas Podelski

ultimate-pa.org

github.com/ultimate-pa/ultimate

Commutativity Simplifies Proofs of Concurrent Programs

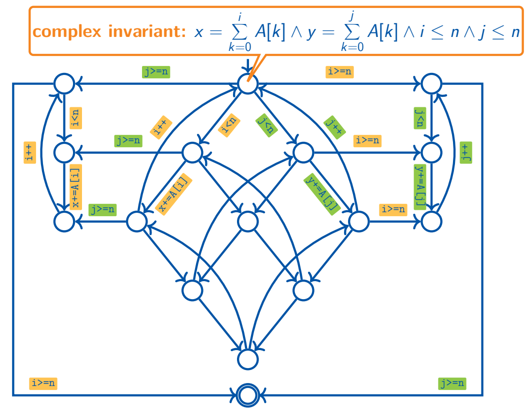
Concurrent Program

$\{ x = y = i = j = 0 \}$

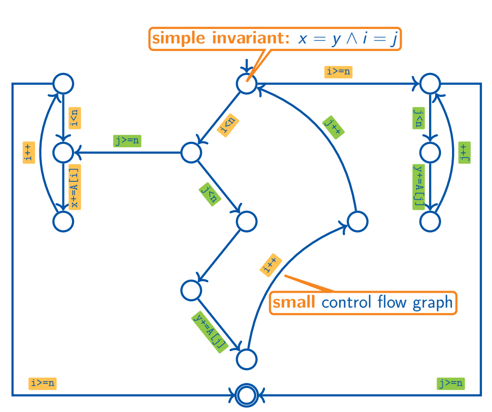
```
while (i < n) {          while (j < n) {
    x += A[i];      ||       y += A[j];
    i++;                     j++;
}                        }
```

$\{ x = y \}$

All Interleavings



A Sound Reduction



Commutativity

Many pairs of statements **commute**:
i.e., order of execution does not matter

Example: `x+=A[i]` `y+=A[j]` $\sim$ `y+=A[j]` `x+=A[i]`

Extension: **proof-sensitive commutativity**

Example: `*x = 0` `*y = 1` $\sim$ `*y = 1` `*x = 0`
if we have proven that $x \neq y$

swapping adjacent commuting statements
$\rightsquigarrow$ **equivalent** traces

Reduction

representative subset of program traces: at least one representative per equivalence class

Soundness:

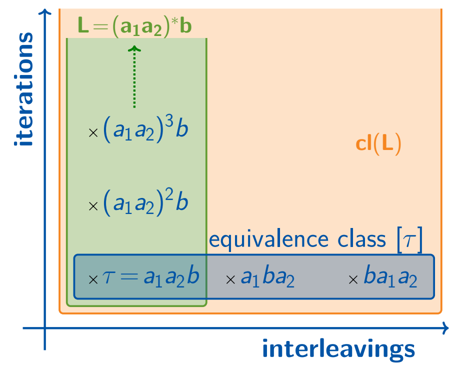
one trace correct $\Rightarrow$ all equivalent traces correct
correctness of reduction $\Rightarrow$ correctness of program

Verification Principle

GemCutter **generalizes** from spurious counterexamples τ to larger sets of correct traces:

trace abstraction generalizes across loop iterations to a set of traces L

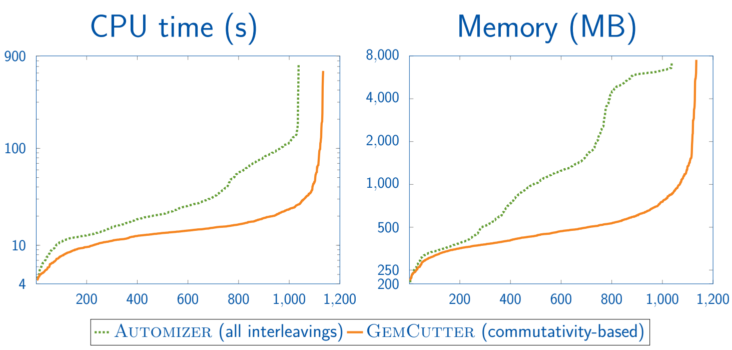
commutativity allows for generalization across interleavings to the set $cl(L)$ of all equivalent traces



If $cl(L)$ contains all program traces, the program is correct.
Equivalently: If L contains all traces *of a reduction*, then the program is correct.

Performance

Evaluation shows significant advantages over a state-of-the-art verifier (Ultimate Automizer):



Competitions:

- **SV-COMP'24:** **2nd place** in *ConcurrencySafety*
- **SV-COMP'23:** **3rd place** in *ConcurrencySafety*
- **SV-COMP'22:** **3rd place** in *ConcurrencySafety*, **1st place** in *NoDataRace (demo)*

Commutativity & Verification

choice of representatives affects proof simplicity
- **challenge:** select suitable representatives

choice of proof affects possible commutativity
- **challenge:** find useful *abstract* commutativity

partial order reduction algorithms speed up verification
- **challenge:** adapt classical POR algorithms

commutativity reasoning is widely applicable
- **challenge:** extend to more programs & properties

[SV-COMP'22] *Ultimate GemCutter and the Axes of Generalization,* Klumpp, Dietsch, Heizmann, Schüssele, Ebbinghaus, Farzan and Podelski, 2022

[PLDI'22] *Sound Sequentialization for Concurrent Program Verification,* Farzan, Klumpp and Podelski, 2022

[POPL'23] *Stratified Commutativity in Verification Algorithms for Concurrent Programs,* Farzan, Klumpp and Podelski, 2023

[POPL'24] *Commutativity Simplifies Proofs of Parameterized Programs,* Farzan, Klumpp and Podelski, 2024

ULTIMATE KOJAK

ultimate.informatik.uni-freiburg.de github.com/ultimate-pa/ultimate

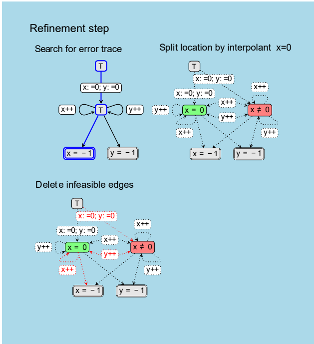
Daniel Dietsch, Marius Greitschus, Matthias Heizmann, Jochen Hoenicke, Alexander Nutz, Christian Schilling, Tanja Schindler

Features

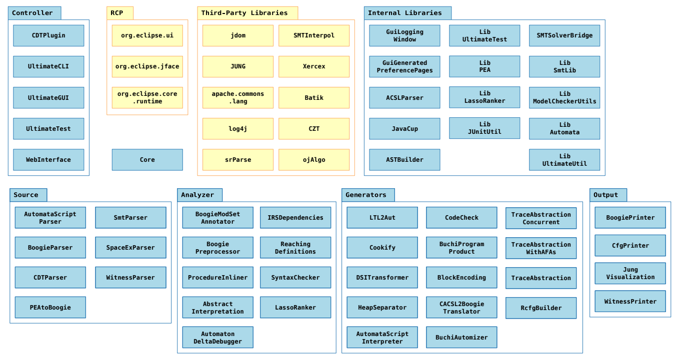
- Reachability analysis
- Memory safety analysis
- Bitprecise analysis
- IEEE 754 floating point analysis
- Error witnesses
- Correctness witnesses

Techniques

- Abstraction refinement
- Configurable block encodings
- Multi SMT solver support
- Newton-style interpolation



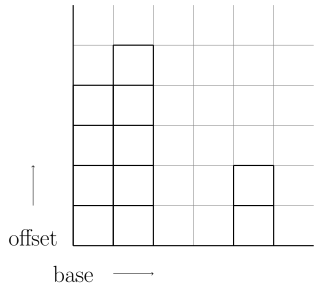
ULTIMATE program analysis framework



C memory model

Models dynamically allocated memory through Boogie arrays:

- `memory-[int|pointer|bitvector8|...]`: store memory contents
  - one array per used Boogie data type
  - two dimensional, a memory address has components "base" and "offset"
  - models disjointness of memory areas allocated by different malloc calls
- `valid`: store which base addresses are allocated
- `length`: store maximal offset at each base address
- "`*p` is a valid pointer dereference" $\iff$ `valid`[`p.base`] $\wedge$ `p.offset` $\leq$ `length`[`p.base`]
- "Program has no memory leaks" $\iff$ `valid` $=$ `old(valid)` at the end of `main`



valid	T	T	F	F	T	F
length	4	5	-1	-1	2	-1

SMT solver integration

Hoare triple checks

"Is $\{P\}\, s\, \{Q\}$ a Hoare triple?"

Features:

- Simplify check if $(\mathit{variables}(P) \cup \mathit{variables}(st)) \cap \mathit{variables}(Q) = \emptyset$.
  - often blocked because P, st and Q access the same array (but perhaps at different positions)
  - attempt to partition arrays via "alias analysis" (work in progress)
- Avoid checks with intricate predicates.
- Use incremental (push/pop) solver queries when possible, e.g., group checks that share the same precondition P.
- Abstract interpretation-based:
  Check if $\mathit{post}^{\#}(P^{\#}, st) \sqsubseteq Q^{\#}$ holds in some abstract domain.
- Unify equivalent predicates.
- Cache Hoare triples and implication between predicates.

Tree interpolation

- Interpolating solvers used by Ultimate: SMTInterpol, Z3
- Tree interpolation syntax example (procedures `foo`, `bar`):

```
(assert (! (..) :named foo-stm1))
(assert (! (..) :named foo-stm2))
(assert (! (..) :named bar-stm1))
(assert (! (..) :named bar-stm2))
(assert (! (..) :named foo-stm3))
(check-sat)
(get-interpolants (foo-stm1 foo-stm2 (bar-stm1 bar-stm2) foo-stm3))
```

Interface

- Java interface (currently only SMTInterpol)
- SMTLib2 interface
- Solvers in use at SV-COMP 2018: SMTInterpol, Z3, MathSat, CVC4 as many as we can get!

Newton-style interpolation

- Input: infeasible trace $st_1, \ldots, st_n$, unsatisfiable core $\mathsf{UC} \subseteq \{st_1, \ldots, st_n\}$
- Replace statements not in UC:
  - assume statement `ψ` ⇝ `skip`
  - assignment statement `x:=t` ⇝ `havoc x`
- Compute sequence of predicates $\varphi_0, \ldots, \varphi_n$ iteratively using strongest post operator post

$$\varphi_0 := \mathit{true}$$
$$\varphi_{i+1} := \mathit{post}(\varphi_i, st_{i+1})$$

- Eliminate each variable from predicate φ_i that is not live at position i of the trace.
- Output: sequence of predicates $\varphi_0, \ldots, \varphi_n$ which is a sequence of interpolants for the infeasible trace $st_1, \ldots, st_n$

	trace τ	state assertions for τ	interpolating trace $\tau^{\#}$	state assertions for $\tau^{\#}$
φ_0		true		true
st_1	`b:=a`		`b:=a`	
φ_1		$a = b$		$a = b$
st_2	`x:=0`		`havoc x`	
φ_2		$a = b \wedge x = 0$		$a = b$
st_3	`havoc p`		`havoc p`	
φ_3		$a = b \wedge x = 0$		$a = b$
st_4	`!a[p]`		`!a[p]`	
φ_4		$a = b \wedge x = 0 \wedge a[p] = \mathit{false}$		$a = b \wedge a[p] = \mathit{false}$
st_5	`a[p]:=true`		`a[p]:=true`	
φ_5		$a = b[p := \mathit{true}] \wedge x = 0 \wedge a[p] = \mathit{true}$		$a = b[p := \mathit{true}] \wedge a[p] = \mathit{true}$
st_6	`x:=x+1`		`havoc x`	
φ_6		$a = b[p := \mathit{true}] \wedge x = 1 \wedge a[p] = \mathit{true}$		$a = b[p := \mathit{true}] \wedge a[p] = \mathit{true}$
st_7	`a[p]:=false`		`a[p]:=false`	
φ_7		$a = b \wedge x = 1 \wedge a[p] = \mathit{false}$		$a = b \wedge a[p] = \mathit{false}$
st_8	`a!=b`		`a!=b`	
φ_8		false		false

SoSy-Lab
Software Systems

6th Competition on Software Testing

(TEST-COMP '24)

Dirk Beyer

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Features

Table 2: Technologies and features that the test generators used

Tester	Bounded Model Checking	CEGAR	Evolutionary Algorithms	Explicit-Value Analysis	Floating-Point Arithmetics	Guidance by Coverage Measures	Predicate Abstraction	Random Execution	Symbolic Execution	Targeted Input Generation	Algorithm Selection	Portfolio
cetfuzz new			✓								✓	
CoVeriTest		✓		✓	✓		✓					✓
ESBMC-kind ∅	✓			✓	✓							
FDSE new					✓	✓		✓	✓			
Fizzer new												
FuSeBMC	✓				✓	✓				✓		✓
FuSeBMC-AI	✓				✓	✓				✓		✓
HybridTiger ∅		✓		✓	✓		✓					
KLEE ∅					✓				✓	✓		
KLEEF new					✓	✓			✓	✓		
Legion ∅				✓	✓	✓		✓	✓	✓		
Legion/SymCC ∅				✓	✓	✓		✓	✓	✓		
Owi new					✓			✓	✓	✓		
PRTest					✓			✓				
Rizzer new									✓			
Symbiotic					✓	✓			✓	✓		✓
TracerX	✓				✓				✓	✓		
TracerX-WP new												
UTestGen new		✓										✓
WASP-C ∅					✓			✓	✓			

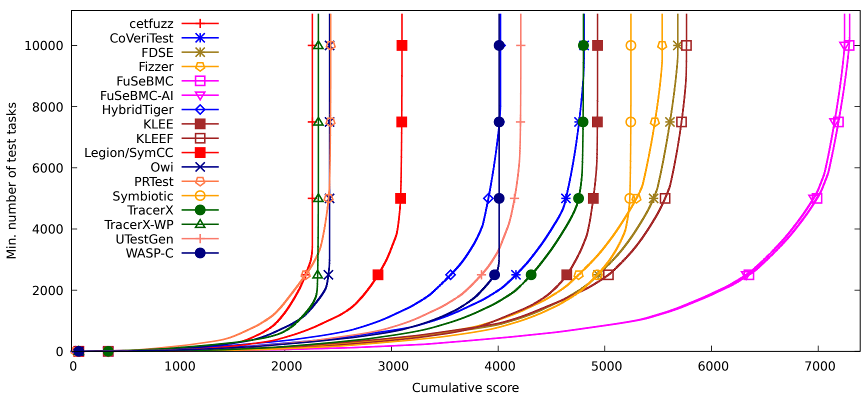
Participants

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

Tester	Jury member	Affiliation
cetfuzz new	Sumesh Divakaran	College of Eng. Trivandrum, India
CoVeriTest	Marie-Christine Jakobs	LMU Munich, Germany
ESBMC-kind ∅	(hors concours)	–
FDSE new	Zhenbang Chen	National U. of Defense Techn., China
Fizzer new	Marek Trtík	Masaryk U., Brno, Czechia
FuSeBMC	Kaled Alshmrany	U. of Manchester, UK
FuSeBMC-AI	Mohannad Aldughaim	U. of Manchester, UK
HybridTiger ∅	(hors concours)	–
KLEE ∅	(hors concours)	–
KLEEF new	Yurii Kostyukov	Huawei, China
Legion ∅	(hors concours)	–
Legion/SymCC ∅	(hors concours)	–
Owi new	Léo Andrès	OCamlPro / LMF, France
PRTest	Thomas Lemberger	LMU Munich, Germany
Rizzer new	Adam Štafa	Masaryk U., Brno, Czechia
Symbiotic	Martin Jonáš	Masaryk U., Brno, Czechia
TracerX	Joxan Jaffar	National U. of Singapore, Singapore
TracerX-WP new	Joxan Jaffar	National U. of Singapore, Singapore
UTestGen new	Max Barth	LMU Munich, Germany
WASP-C ∅	(hors concours)	–

Final Score

Figure 1: Quantile functions for category *Overall*.



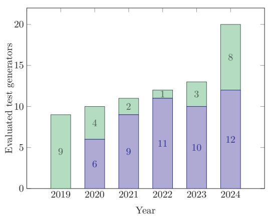
Results

Table 3: Quantitative overview over all results

Tester	Cover-Error 1173 tasks	Cover-Branches 2933 tasks	Overall 4106 tasks
cetfuzz new	226	2197	2258
CoVeriTest	462	4826	4806
ESBMC-kind ∅	195		
FDSE new	617	5132	5684
Fizzer new	583	**5146**	5538
FuSeBMC	**930**	**5478**	**7295**
FuSeBMC-AI	**926**	**5418**	**7248**
HybridTiger ∅	393	3987	4022
KLEE ∅	713	3023	4932
KLEEF new	655	4975	**5766**
Legion ∅		2896	
Legion/SymCC ∅	264	3381	3098
Owi new	256	2241	2420
PRTest	167	2980	2431
Rizzer new	555		
Symbiotic	**666**	3957	5245
TracerX	509	4435	4799
TracerX-WP new	322	1521	2315
UTestGen new	409	4195	4212
WASP-C ∅	532	2838	4009

Participation

Top: New participants



Ranking

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

Rank	Tester	Score	CPU Time (in h)
Cover-Error			
1	FuSeBMC	**930**	76
2	FuSeBMC-AI	926	68
3	Symbiotic	666	5.2
Cover-Branches			
1	FuSeBMC	**5478**	2400
2	FuSeBMC-AI	5418	2300
3	Fizzer new	5146	1700
Overall			
1	FuSeBMC	**7295**	2500
2	FuSeBMC-AI	7248	2400
3	KLEEF new	5766	1700

Report

https://test-comp.sosy-lab.org/2024/

References

Reference
D. Beyer. Automatic testing of C programs: Test-Comp 2024. Springer, 2024