ETAPS Poster Book

Collection of posters presented at ETAPS 2026

Turin, April 11–16, 2026

ESOP 2026 Posters

Caesar: A Deductive Verifier for Probabilistic Programs

caesarverifier.org

Contact: Philipp Schröer (phisch@cs.rwth-aachen.de)

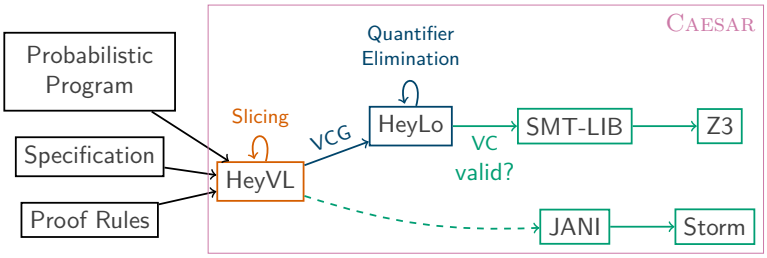
RWTH AACHEN UNIVERSITY

Motivation

- **Probabilistic programs**: modeling randomized algorithms, protocols, and security — and are notoriously hard to get right.
- **Quantitative specifications**: termination probability, expected runtime, resource usage.
- **Proof Rules**: loop invariants, martingales, probabilistic termination.

⇒ **Caesar** aims to be the unifying probabilistic analogue of Boogie/Dafny.

Verification Infrastructure [1]



HeyVL: A Probabilistic Verification Language

Pre- and postconditions are *expectations* – functions from states to $\mathbb{R}^{\infty}_{\geq 0}$. A `proc` with pre f and post g *verifies* iff $f \leq \mathrm{wp}[\![S]\!](g)$, i.e. f is a *lower bound* on the expected value of g after S. Dually, a `coproc` checks $f \geq \mathrm{wp}[\![S]\!](g)$: f *upper-bounds* the expected value.

```
// Verify: E[runtime] <= 2 for a geometric loop
coproc geo_expected_runtime() -> (done: Bool)
    pre  2            // claimed upper bound
    post 0
{
    done = false
    @invariant([!done] * 2)   // user-provided
    while !done {
        reward 1              // count one step
        done = flip(0.5)      // fair coin
    }
}
```

Caesar verifies that the **expected number of iterations is at most 2**, checking that $[\neg\mathrm{done}] \cdot 2$ is a valid loop invariant — automatically.

Weakest Pre-Expectation Reasoning

- **Weakest pre-expectations** generalize Dijkstra's wp-calculus: $\mathrm{wp}[\![S]\!](f)$ is the expected value of f after S.
- **HeyLo** is our $\mathbb{R}^{\infty}_{\geq 0}$-valued assertion logic — *expectations* replace Boolean predicates as pre- and post-conditions.
- **HeyVL** is our quantitative IVL: proof rules are encoded in programs, supporting lower- and upper-bound reasoning.
- **Backends**: Z3 (SMT solver) for verification conditions; Storm (via JANI) for model checking.
- **Soundness**: all verified bounds carry formal guarantees [2].

Advanced Features

- **Extensible**: Uninterpreted functions and user-defined datatypes.
- **Lazy wp unfolding**: optimizations based on forwards-moving symbolic execution for branch pruning.
- **Quantitative verification stmts.**: `assume 0.5*x` to assume the expected value $\geq 0.5 \cdot x$. Also: `assert`, `havoc`, `coassume`, `coassert`, …

Latest Developments

- **Caesar Application [3]:** New proof rule for almost-sure termination under weak fairness for distributed consensus.
- **Slicing [4]:** Error diagnostics, verification witnesses, and hints.
- **Beyond Expected Values [5]:** A programmatic reward transformation using classical wp-reasoning to obtain higher moments, tail probabilities, …
- **Continuous Distributions [6]** like uniform, Irwin-Hall by Riemann sum approximation with sound over- and under-approximations.

[1] Schröer, Batz, Dural, Haase, Kaminski, Katoen, Matheja. *Caesar: A Deductive Verifier for Probabilistic Programs.* Submitted '26.
[2] Schröer, Batz, Kaminski, Katoen, Matheja. *A Deductive Verification Infrastructure for Probabilistic Programs.* OOPSLA '23.
[3] Enea, Majumdar, Motwani, Sathiyanarayana. *Verifying Almost-Sure Termination for Randomized Distributed Algorithms.* POPL '26.
[4] Schröer, Haase, Katoen. *Error Localization, Certificates, and Hints for Probabilistic Program Verification via Slicing.* ESOP '26.
[5] Schröer, Katoen. *Highly Incremental: A Simple Programmatic Approach for Many Objectives.* FM '26.
[6] Batz, Katoen, Randone, Winkler. *Foundations for Deductive Verification of Continuous Probabilistic Programs.* OOPSLA '25.

Causal-Broadcast Memory

Amir Karniel Ori Lahav
Tel Aviv University

DistMem: Shared memory based on causal broadcast

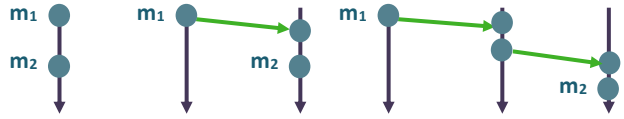
Every process p has a local mapping M_p from locations to values. The code for a process p is:

```
method write(x,v)
  M_p := M_p[x ↦ v]
  causal_broadcast(Wxv)
  return()

method read(x)
  return(M_p(x))

when Wxv is received
from another process:
  M_p := M_p[x ↦ v]
```

- A message m_1 is a cause of a message m_2 when the send of m_1 comes before the send of m_2 in Lamport's happens-before order:



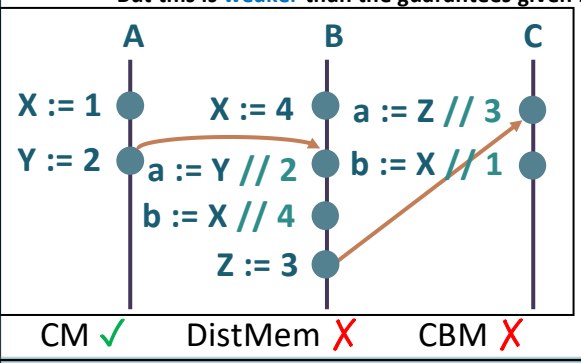
- Causal-Broadcast requires that if some message m_1 is a cause of a later message m_2, then all processes will receive m_1 before m_2.
- The above is a classical algorithm given in "Causal memory: Definitions, implementation, and programming".

But what *is* causal memory?

We ask: given an abstract execution (*no* propagation edges, only sequences of memory operations), what makes it "causal-memory-consistent"?

- The above paper suggests the following model, termed CM:
  - There exists a (partial) causal order *co* extending the program order
  - Every process p has a serialization S_p of all events, justifying its own read events by their *co*-past
  - Each serialization agrees with *co*

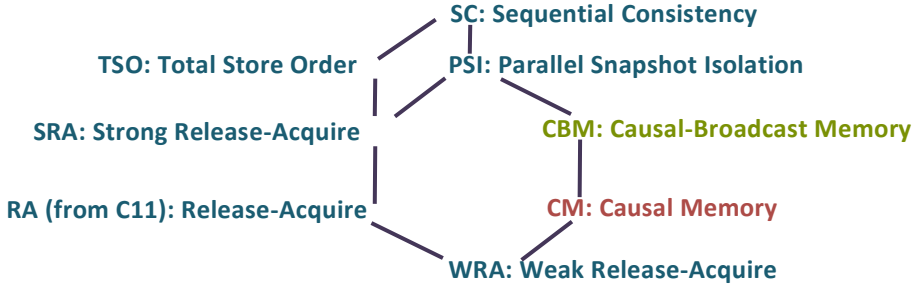
But this is weaker than the guarantees given by the actual implementation!



- CM misses the necessary *co* edge from **X := 1** to **X := 4**.
- We fix this gap by adding the following to the definition of CM, resulting in our novel model **CBM**:
  - If S_p places an event before an event of p, there is *co* between them
- **CBM captures precisely the behaviors allowed by DistMem.**

Additional results

- **Testing**: Checking CBM-consistency of an execution graph can be done in polynomial time.
- **Verifying**: Checking state reachability of client programs operating on top of DistMem is undecidable.
- Placing CBM among other memory models (see below) and proving a write-write-race freedom result.
- Generalizing CBM and DistMem to shared objects beyond memory.



Patrycja Balik, Szymon Jędras, and Piotr Polesiuk

Fram

— a Tale of —

fram-lang.org

Theory-Driven Development,

Effect Inference

In the presence of higher-rank polymorphism!

```
let foo (f : Int ->[IO] Int) : [DB] Int = ...
let bar (g : {E : effect} -> _) = g (foo g)
```

Question: What is the type of bar?

- ({E} -> Int ->[E] Int) ->[DB] Int,
- ({E} -> Int ->[IO] Int) ->[DB,IO] Int,
- something else?

Answer: ({E} -> Int ->[E?ρ,α] Int) ->[DB,α] Int, assuming $\alpha <: \mathrm{IO}$

$\varepsilon ::= \alpha \mid \mathrm{pure} \mid \varepsilon \cdot \varepsilon \mid \varepsilon?\varphi$ — Propositional formula representing delayed decision

```
tr_type(∀α^κ. T) =
  let (P; Δ; r) = tr_type(T) in
  fresh P_α = {p_γ | β ∈ Δ} in
  fresh Δ' = {p_γ | β ∈ Δ} in
  (P ⊎ P_α; Δ'; ∀α^κ.{∀ ξ ∈ Δ ↦ γ_ξ : α[p_γ]}^κ r)

subtype(∀α^κ. r; ∀α^κ. r') =
  let (Ω, φ) = subtype(r, r') in
  (Ω[α ↦ pure] ⊓ ⊤, ρ ∧ Δ[α])
```

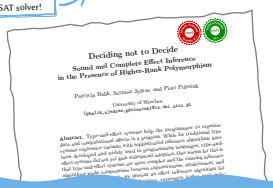
Solved later by a SAT solver!

$$(\alpha)[\alpha] \triangleq \top \qquad (\mathrm{pure})[\alpha] \triangleq \bot \qquad (\varepsilon?\varphi)[\alpha] \triangleq \varepsilon[\alpha] \wedge \varphi$$
$$(\beta)[\alpha] \triangleq \bot \quad (\alpha \neq \beta) \qquad (\varepsilon_1 \cdot \varepsilon_2)[\alpha] \triangleq \varepsilon_1[\alpha] \vee \varepsilon_2[\alpha]$$
$$\Omega[\alpha] \triangleq \bigwedge_{(\varepsilon_1 <: \varepsilon_2) \in \Omega} (\varepsilon_1[\alpha] \Rightarrow \varepsilon_2[\alpha])$$

Collected constraints are stored in algebraic type schemes (extension of ML polymorphism).

$\forall \overline{\alpha}.\ [\Omega]\ \tau$

Inference is still decidable for higher-rank polymorphism without constraints in schemes!



Relational Models

Actual Core language from the implementation verified in Rocq!

$\mathcal{L}\varepsilon\ @\ \tau_d/\varepsilon_d$ — Interpretation of Computational Effects

$$\langle E[\mathcal{S}^\ell k.\, e]\rangle^\ell \longrightarrow \{k \mapsto \lambda x.\, \langle E[x]\rangle^\ell\}^* e$$

$$\frac{\Gamma \vdash v : \mathcal{L}\varepsilon\ @\ \tau_d/\varepsilon_d \qquad \Gamma, k : \tau \to_{\varepsilon_d} \tau_d \vdash e : \tau_d/\varepsilon_d}{\Gamma \vdash \mathcal{S}^\ell k.\, e : \tau/\varepsilon}$$

$$\frac{\Gamma \vdash v : \mathcal{L}\varepsilon\ @\ \tau_d/\varepsilon_d \qquad \Gamma \vdash e : \tau_d/\varepsilon \cdot \varepsilon_d}{\Gamma \vdash \langle e\rangle^\ell : \tau_d/\varepsilon_d} \qquad \frac{\alpha - \text{fresh} \qquad \Gamma, x : \mathcal{L}\alpha\ @\ \tau_d/\varepsilon_d \vdash e : \tau/\varepsilon}{\Gamma \vdash \mathrm{label}\ x\ \mathrm{in}\ e : \tau/\varepsilon}$$

$$\ell \in [\![\mathcal{L}\varepsilon_0\ @\ \tau/\varepsilon]\!]_\eta \iff$$
$$[\![\varepsilon_0]\!]_\eta = \{\langle \mathcal{S}^\ell k.\, e, \ell, R\rangle \mid \forall v \in (R \to_{[\![\varepsilon]\!]_\eta} [\![\tau]\!]_\eta).\ \{k \mapsto v\}^* e \in \mathcal{E}([\![\tau]\!]_\eta, [\![\varepsilon]\!]_\eta)\}$$

$\mathbf{Effect} \triangleq \mathbf{UPred}(\mathrm{Exp}_{\mathrm{cl}} \times \mathrm{Lbl} \times \mathbf{UPred}(\mathrm{Val}_0))$
$ERF \triangleq R \cup (\text{makes step}) \cup SRF$
$SRF \triangleq \{E[e] \mid \exists \ell, R_v.\ (e, \ell, R_v) \in F \wedge \ell\text{-free}\ E \wedge E \in R_v \to_\varepsilon R\}$

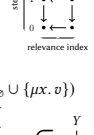
Borrowed ~~Stolen~~ from *Handle with Care*

$\mathcal{D}_r\tau_1 \simeq \tau_2$ — Recursive Datatypes and Irrelevant Values

$$\frac{\Gamma \vdash_{\mathrm{irr}} v : \mathcal{D}_r\tau_1 = \tau_2 \qquad \Gamma \vdash e : \tau_2/\varepsilon}{\Gamma \vdash \mathrm{roll}_0\ e : \tau_1/\varepsilon} \qquad \frac{\Gamma \vdash_{\mathrm{irr}} v : \mathcal{D}_r\tau_1 = \tau_2 \qquad \Gamma \vdash e : \tau_1/\varepsilon}{\Gamma \vdash \mathrm{unroll}_0\ e : \tau_2/\mathrm{NTerm}^?r \cdot \varepsilon}$$

$$\frac{a - \text{fresh} \qquad a \text{ occurs positively in } \tau_2 \text{ when } r = + \qquad \Gamma, x :_{\mathrm{irr}} \mathcal{D}_r\alpha \simeq \tau_2 \vdash e : \tau/\varepsilon}{\Gamma \vdash \mathrm{label}\ x\ \mathrm{in}\ e : \tau/\varepsilon}$$

$$v \in [\![\mathcal{D}_r\tau_1 \simeq \tau_2]\!]_\eta \iff$$
$$\triangleright_{\mathrm{rel}} \bot \wedge$$
$$\square_{\mathrm{rel}}([\![\tau_2]\!]_\eta \subseteq [\![\tau_1]\!]_\eta) \wedge$$
$$\square_{\mathrm{rel}}([\![\tau_1]\!]_\eta \subseteq \triangleright_\gamma [\![\tau_2]\!]_\eta)$$



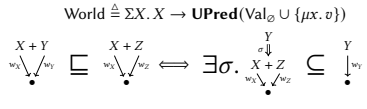
$\Gamma, x : o\tau$ — Productivity of General Recursion

$$\frac{\Gamma, x : o\tau \vdash e_1 : \tau/\mathrm{pure} \qquad \Gamma, x : \tau \vdash e_2 : \tau'/\varepsilon}{\Gamma \vdash \mathrm{letrec}\ x = e_1\ \mathrm{in}\ e_2 : \tau'/\varepsilon}$$

$$\frac{\Gamma, \Gamma' \vdash e : \tau/\varepsilon}{\Gamma, o\Gamma' \vdash o e : \tau/\mathrm{NTerm}\cdot\varepsilon}$$

$$\frac{e_1 \longrightarrow e_1'}{\mathrm{letrec}\ x = e_1\ \mathrm{in}\ e_2 \longrightarrow \mathrm{letrec}\ x = e_1'\ \mathrm{in}\ e_2}$$

$$\mathrm{World} \triangleq \Sigma X.\, X \to \mathbf{UPred}(\mathrm{Val}_\varnothing \cup \{\mu x.\, v\})$$



Coherence

Fram has implicit parameters...

```
let foo bar =
  let ~a = 13 in bar ()
let ~a = 42 in
foo (fn () -> ~a)
```

What should the above code evaluate to?

- 42?
- 13?

...and parameter sections.

```
parameter ~n
parameter ~m

let baz x = ...
```

Which parameters should `baz` generalize?

- ~n?
- ~a?
- Both?

Source of ambiguity

Semantics is given by a translation of the typing derivation.

$$\frac{\Gamma; \overline{\sim n}, \sim m : \tau \vdash e : \sigma \rightsquigarrow e \qquad \Gamma \setminus \{\overline{\sim n}\}, \sim m : \tau \vdash e : \sigma \rightsquigarrow e}{\Gamma; \overline{\sim n} \vdash \mathrm{parameter}\ \sim m\ \vec{d} \rightsquigarrow \vec{d} \Rightarrow \Gamma' \qquad \Gamma, x : \{\overline{\sim m} : \tau\} \to \sigma; \overline{\sim n} \vdash \vec{d} \rightsquigarrow \vec{d} \Rightarrow \Gamma'}$$
$$\frac{}{\Gamma; \overline{\sim n} \vdash \mathrm{fun}\ x = e\ \vec{d} \rightsquigarrow \mathrm{fun}\ x \overline{m} = e\ \vec{d} \Rightarrow \Gamma'}$$

$$\frac{\Gamma(x) = \{\overline{\sim m} : \tau'\} \to \sigma \qquad \Gamma \vdash e : \tau \rightsquigarrow e \qquad \Gamma, \overline{\sim n} : ? \vdash \sim m : \tau' \rightsquigarrow \sim m}{\Gamma \vdash x\{\overline{\sim n = e}\} : \sigma \rightsquigarrow \mathrm{let}\ \overline{\sim n = e}\ \mathrm{in}\ x\ \overline{\sim m}}$$

Problem:

$$\left.\begin{array}{l}\Gamma \vdash e : \tau \rightsquigarrow e_1 \\ \Gamma \vdash e : \tau \rightsquigarrow e_2\end{array}\right\} \overset{?}{\Rightarrow} e_1 \equiv e_2$$

Solution: Equivalent untyped semantics
which does not depend on a particular type derivation.

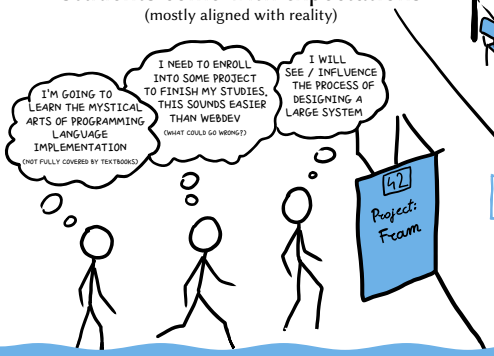
$$\frac{\rho; \overline{\sim n} \vdash \sim m \Downarrow \rho' \qquad \rho; \overline{\sim n} \vdash \vec{d} \Downarrow \rho''}{\rho; \overline{\sim n} \vdash \mathrm{parameter}\ \sim m\ \vec{d} \Downarrow \rho'} \qquad \frac{\rho[x \mapsto \langle \rho \setminus \{\overline{\sim n}\}, \lambda\{\overline{\sim n}\}.\, e\rangle]; \overline{\sim n} \vdash \vec{d} \Downarrow \rho'}{\rho; \overline{\sim n} \vdash \mathrm{fun}\ x = e\ \vec{d} \Downarrow \rho'}$$

$$\frac{\rho(x) = \langle \rho', \lambda\{\overline{\sim m}\}.\, e'\rangle \qquad \rho \vdash e \Downarrow v \qquad \rho' \uplus (\rho|_{\overline{\sim m}} \vdash e' \Downarrow v')}{\rho \vdash x\{\overline{\sim m = e}\} \Downarrow v'}$$

Student Involvement, and

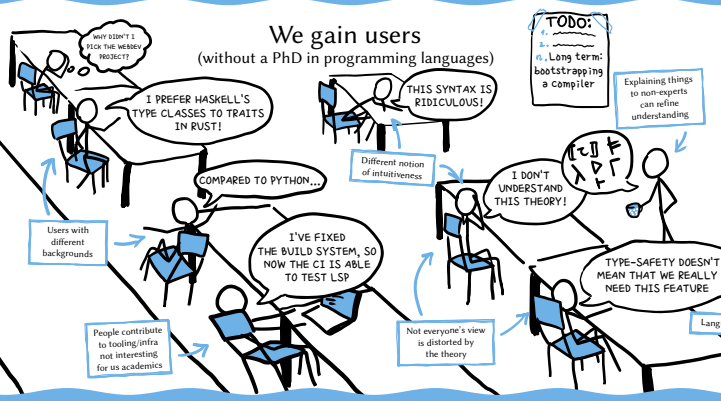
Students come with expectations

(mostly aligned with reality)



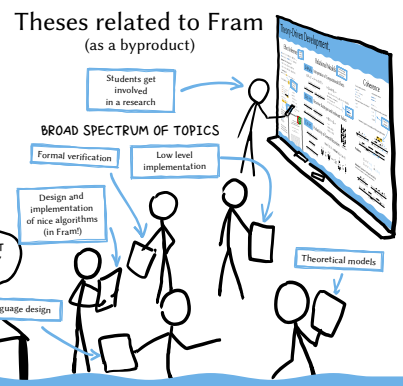
We gain users

(without a PhD in programming languages)



Theses related to Fram

(as a byproduct)



Minimal, but Expressive Design

Named Parameters

Fram supports Rank-N types.

```
let foo (id : {X} -> X -> X) =
  (id True, id {X = Int})
```

What else can we put in the curly {braces}?

Named type parameters

Optional and regular named parameters

Implicit parameters

```
parameter ~say

let greeting {~say} name =
  ~say ("Hello " + name + "!")

let farewell name =
  ~say ("Bye " + name + "!")

let conversation {name, ?msg} () =
  greeting name; msg.iter ~say; farewell name

let _ =
  let ~say = printStrLn
  let msg = "Isn't the weather nice?"
  in
  conversation { msg, name = "Fram" } ()
```

Sections — Declared parameters are abstracted implicitly when used

Method constraints

Packed/unpacked modules

Nice Features for Free!

Records

```
data Vec = { x : Int, y : Int }
```

is a syntactic sugar for

```
data Vec = Vec of { x : Int, y : Int }
method x (Vec { x }) = x
method y (Vec { y }) = y
```

so we can write

```
method add (v1 : Vec) (v2 : Vec) =
  Vec { x = v1.x + v2.x, y = v1.y + v2.y }
```

Functors

```
data MapS Key = Map of
  { T : type -> type
  , empty : {Val} -> T Val
  , method add : {Val} -> T Val -> Key -> Val ->[] T Val
  , ...
  }

make : { Key, method lt : Key -> Key ->[] Bool } -> MapS Key

let Map { module IntMap } = make { Key = Int }
```

Existential Types

```
data Stream X =
  | Stream of {St}, St, (St ->[] Pair X St)

let map1 f (Stream {St=OldSt} st next) =
  Stream {St=Pair Int OldSt}
    (0, st)
    (fn (i, st) =>
      let (x, st') = next st in
      (f i x, (i+1, st')))
```

Effect Handlers

```
data BT E =
  { flip : Unit ->[E] Bool
  , fail : {X} -> Unit ->[E] Bool
  }

method rec select (bt : BT _) a b =
  if a >= b then bt.fail ()
  else if bt.flip () then a
  else bt.select (a + 1) b
```

Using datatypes for effect capabilities

```
let triples (bt : BT _) n =
  let a = bt.select 1 n
  let b = bt.select a n
  let c = bt.select b n
  in
  if a * a + b * b == c * c then (a, b, c)
  else bt.fail ()
```

Lexically scoped effect handlers!

```
let printTriples n =
  handle bt = BT
    { effect flip () = resume True; resume False
    , effect fail () = ()
    }
  in
  printStrLn (triples bt n).show
```

Handler defines an effect capability and introduces a new effect

Effect system guarantees that effects doesn't escape their scope

Don't tell anyone that this is the shift$_0$ delimited control operator

Almost invisible to the programmer thanks to the effect inference

An Example: Prolog

```
parameter ~bt : BT _
parameter ~st : VarState _
parameter ~kb : KnowledgeBase _
parameter ~fr : Fresh _

let rec unify (t1 : Term) (t2 : Term) =
  match t1.view, t2.view with
  ...
  | Var x, t ->
    t.occursCheck x;
    ~st.set x t
  | Fun f ts1, Fun g ts2 ->
    if f == g then List.iter2 {~onError = ~bt.fail} unify ts1 ts2
    else ~bt.fail ()
  end

let rec eval (t : Term) =
  let Clause t' ts = ~bt.choose ~kb.ask >.refresh in
  unify t t';
  List.iter eval ts
```

```
let ↑Any =
  handler BT
    { effect flip () = resume True >.orElse (fn () => resume False)
    , effect fail () = None
    }
  return x => Some x
  end

let anyTriple n =
  handle bt with ↑Any in triples bt n
```

First-class effect handlers for greater modularity

A Formally Verified Procedure for Width Inference in FIRRTL

Keyin Wang[1,2,3], Xiaomu Shi[1,2,3], Jiaxiang Liu[1,2,3], Zhilin Wu[1,2,3], Fu Song[1,2,3,4], Taolue Chen[5], David N. Jansen[1,2,3]

[1]Key Laboratory of System Software (CAS), Beijing, China · [2]Institute of Software, CAS · [3]Univ. of Chinese Academy of Sciences
[4]Nanjing Institute of Software Technology · [5]Birkbeck, Univ. of London

ESOP 2026 · ETAPS ·Turin, Italy

ESOP Tuesday 17:10

Long-term goal: formally verified FIRRTL compiler

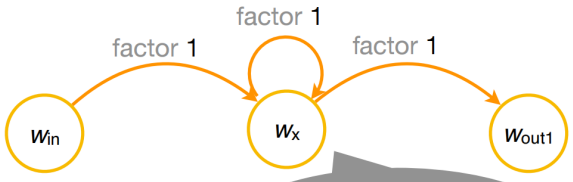
Width Inference Problem: In FIRRTL programs, bit widths of many components are not specified explicitly and must be inferred during compilation. The InferWidths compilation pass in official compilers like **firtool** may fail even for simple FIRRTL programs, especially those with circular dependencies between components.

Solution: Solve the problem through a linear inequation system.

Key takeaways:

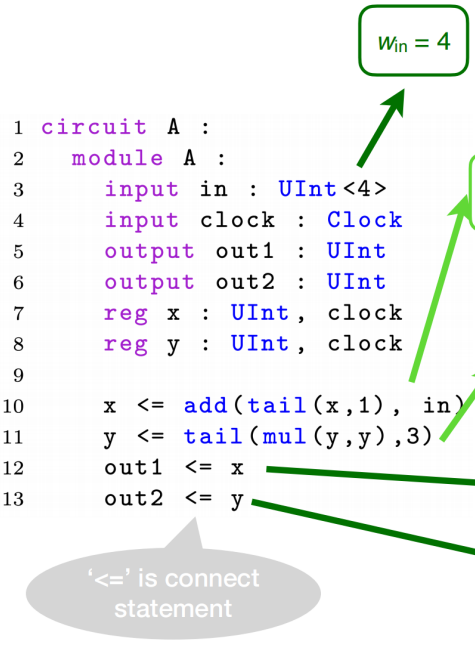
- 1st formally verified width inference
- complete algorithm for all FIRRTL programs
- proven correct with Rocq
- outperforms standard compiler firtool

Dependency graph



Solve inequations in topological order of dependencies

Circular dependencies with factor 1: easy to solve if there is no additive circle > 0



$w_{in} = 4$

$w_x \geq \max(w_{tail(x,1)}, w_{in}) + 1 = \max(w_x - 1, w_{in}) + 1$

$w_x \geq w_x - 1 + 1 \wedge w_x \geq w_{in} + 1$

$w_y \geq w_{mul(y,y)} - 3 = w_y + w_y - 3$

$w_y \geq 2w_y - 3$

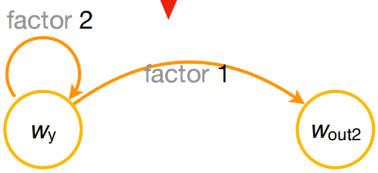
$w_{out1} \geq w_x$

$w_{out2} \geq w_y$

'<=' is connect statement

Even trivial circular dependencies cannot be handled by firtool

Circular dependencies with factor > 1: upper bound on width



Impact of Incomplete Width Inference:

- Widths too large - wastes area/power, degrades timing performance
- Widths too small - arithmetic overflow, data loss, incorrect hardware behavior
- Errors propagate - violate functional guarantees in subsequent compilation passes

Motivation

FIRRTL (Flexible Intermediate Representation for RTL) is an intermediate representation for hardware designs, analogous to **LLVM IR** for C/C++ programs. It is the core dialect in the CIRCT project and used extensively in RISC-V processor designs including **RocketChip, BOOM, NutShell, and XiangShan.**

Why Width Inference Matters: The width of a component determines the number of bits used to represent it in hardware. Correct width inference is essential for:

- **Hardware correctness:** Prevents arithmetic overflow and data loss
- **Resource optimization:** Minimizes area and power consumption
- **Compilation safety:** Errors at this stage propagate to subsequent passes

Formal Verification in Rocq

- **Sound:** If the algorithm returns a solution, it satisfies all constraints
- **Complete:** If a solution exists, the algorithm finds it
- **Optimal:** The returned solution is the least solution

Verification Effort: ~7,000 lines of Rocq code with machine-checked proofs.

Extraction: From the Rocq implementation, we extract an executable OCaml implementation. This is the first formally verified InferWidths pass for FIRRTL.

Benchmark		Avg. #cpnts	Time (ms) per instance		
			firtool	Gurobi	BFWInferWidths
MANUAL (72)		119	7.49	12.07	**1.00**
REALWORLD	NutShell	7,152	190.70	194.55	**158.31**
	Rocket Chip	4,882	127.90	120.64	**22.24**
	RISC-V BOOM	205,608	8,338.30	**3,326.94**	3,467.80

- Handles **100%** benchmarks
- More than **7** (resp, **10**) times faster than firtool (resp, Gurobi) On MANUAL.
- More efficient than firtool, comparable with Gurobi on REALWORLD.

FASE 2026 Posters

REVISITING THE ROLE OF NATURAL LANGUAGE CODE COMMENTS IN CODE TRANSLATION

High-quality comments can bridge semantic gaps

RQ1: Usefulness

- Comments INCREASE success rates
- Indiscriminate use can ALSO DEGRADE results

RQ2: Intent

- Short comments are useful
- Comment should clearly describe what the corresponding code does

RQ3: Density and Language

- Arbitrary restrictions on comment density not useful
- English is the GOLD standard

RQ4: Location

- Inline comments directly next to or above the relevant code block are best
- Provides immediate context for the LLM during translation

Datasets

1,100+ Code Samples (Uncommented) | 80,000+ Translations

Benchmarks - AVATAR and CodeNet

5 PLs - 20 (Source-PL, Target-PL) Pairs

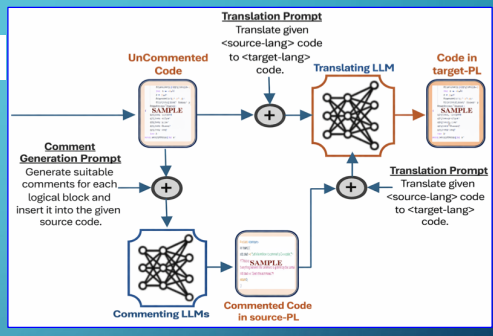
C | C++ | JAVA | Go | PYTHON

Translating Models

Code Llama 13B | DeepSeek-Coder-V2 | GPT-4o-mini | Granite-8B | StarCoder-1

Commenting Models

DeepSeek-Coder-V2 | GPT-4o-mini | Mistral 7B



COMMENTRA TRANSLATION FRAMEWORK

Step 0
Baseline Translation
Attempt translation without any added comments.
If Fails → Go to Step 1

Step 1 Iteration
Inject Basic Comments
Add short, descriptive comments using cost-effective model (e.g., DeepSeek).
If Fails → Go to Next Step

.....

Step n Iteration
Refine Comments
Use stronger model (GPT-4) to refine comments.

Safeguards
Uses comments only when required to explain complex program logic.

Seamless Integration
Plug-and-play compatible with existing translation frameworks.

Cost-Effective
Only invokes commenting models when basic translation fails, minimizing token usage.

Contact : monika.gupta.0878@gmail.com

Don’t go MAD with Anomalies! Design-time Microservice Anomaly Detection in Migration to Microservices

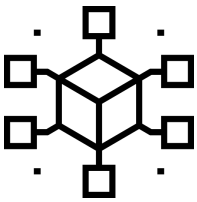
Valentim Romão, Rafael Soares, Luís Rodrigues, Vasco Manquinho

Motivation

Monolith

\+ Strong Consistency
\- Low Scalability

Migration →

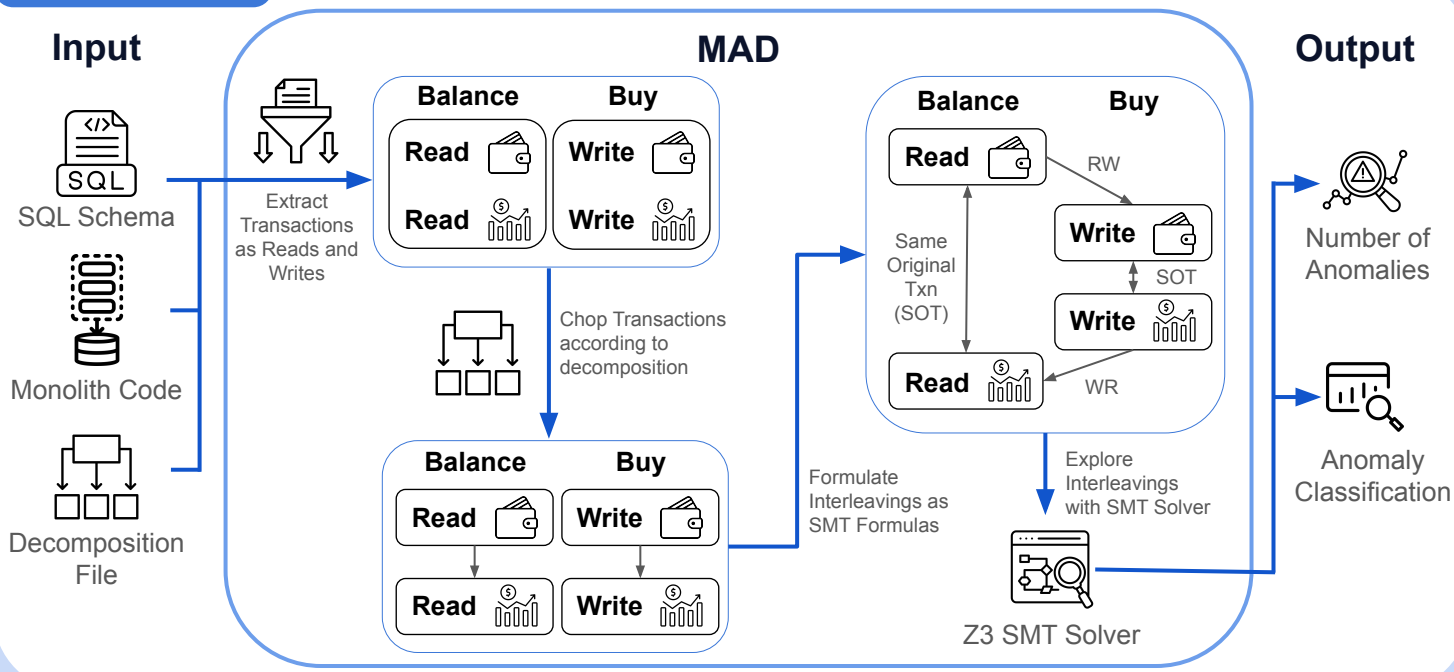


Microservices

\+ High Scalability
\- Loss of Isolation
\- **Data Anomalies**

RQ: How can developers predict data anomalies before the migration?

Solution



Evaluation

Benchmarks

TPC-C
Jpabook
Jpetstore
Myweb
React

Decompositions

Mono
“Best”
Full

MAD Execution Time(s)

	Mono	“Best”	Full
TPC-C	23	9	306
Jpabook	48	136	222
Jpetstore	454	7,239	10,249
Myweb	309	358	514
React	545	996	1,517

Anomalies Found

	Mono	“Best”	Full
TPC-C	0	0	28
Jpabook	0	22	25
Jpetstore	0	40	85
Myweb	0	0	7
React	0	13	18

Anomaly Types

Benchmark	Dirty Write	Lost Updates	Read Skew
Jpetstore	4	12	24

Read the paper:

This work was supported by national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT) under projects UID/50021/2025 (DOI: doi.org/10.54499/UID/50021/2025), UID/PRR/50021/2025 (DOI: doi.org/10.54499/UID/PRR/50021/2025), PTDC/CCI-COM/2156/2021 (DOI: doi.org/10.54499/PTDC/CCI-COM/2156/2021), 2023.14280.PEX (DOI: doi.org/10.54499/2023.14280.PEX), and LISBOA2030-FEDER-00771200 (DOI: doi.org/10.54499/2023.18452.ICDT).

Towards Decentralised Dynamic Reconfiguration of Software Systems

Mina Yavari (m.mina@lancaster.ac.uk) and Damian Arellanes (damian.arellanes@lancaster.ac.uk)

School of Computing and Communications, Lancaster University, United Kingdom

Introduction

- Modern software is used in constantly changing environments so focusing on making software reconfigurable is significant.
- Dynamic reconfiguration approaches can generally be classified into two categories of exogenous (centralised) reconfiguration and endogenous (decentralised) reconfiguration.
- Exogenous reconfiguration: the system configuration is modified by an external controller.
- Endogenous reconfiguration: the system configuration is adapted collaboratively without relying on a central controller.
- Decentralised systems are particularly suitable for modern technologies (the Internet of Things (IoT), Edge computing and cyberphysical systems).

Contributions

We introduce a formal framework for decentralised architectural reconfiguration. Our main contributions are:

- Software Architecture Generator (SAG) for organising alternative architectural components.
- Hierarchical Control Structure (HCS) enabling distributed decision-making.
- Three decentralised protocols supporting runtime reconfiguration.

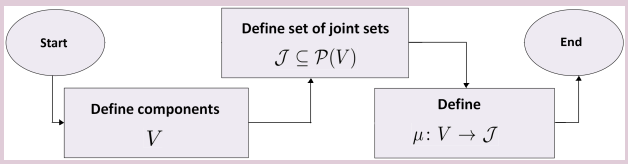
The approach removes the need for a global controller while maintaining coordinated architectural selection.

Software Architecture Generator (SAG)

A SAG is a quintuple $(V, \mathcal{J}, \mu, \Omega, n)$ where:

- V is a non-empty finite set $\{V_i\}_{i\in\mathbb{Z}^+}$ of n-dimensional components, equipped with the order relation $<$ given by $v_i < v_j \iff i \leq j$ for all $v_i, v_j \in V$,
- $\mathcal{J} \subseteq \mathcal{P}(V)$ is a set where $\mathcal{P}$ denotes power set and for each $\gamma \in \mathcal{J}$, called a joint set, we have $\sum_{k=\min I}^{\max I} k = \frac{(\max I+\min I)(\max I-\min I+1)}{2} = \sum_{k\in I} k$ where $I = \{i \in \mathbb{Z}^+ | v_i \in \gamma\}$,
- $\mu\colon V \to \mathcal{J}$ is a non-surjective total function where $\exists!\ v \in V$ with $\mu(v) = \emptyset$, $\exists!\ \gamma \in \mathcal{J}$ with $\gamma \notin \mu(V)$ and, if R is the relation given by μ, $\forall (v,\gamma) \in R,\ \nexists (v',\gamma') \in R, v \in \gamma'$ and
- $\Omega\colon \mathcal{J} \to \mathbb{Z}^+$ is an id function for joint sets given by $\Omega(\gamma) = i \times 10^{|\gamma|-1} + (i+1) \times 10^{|\gamma|-2} + \cdots + (i+|\gamma|-1)) \times 10^0$ where $\gamma \in \mathcal{J}$ with $i = \min\{i \in \mathbb{Z}^+ | v_i \in \gamma\}$.

Fig 1: Three-staged Methodology for Defining SAG Instances



Hierarchical Control Structure (HCS)

- After constructing a SAG, a HCS is formed to define dependencies between joint set identifiers.
- A HCS establishes a master-slave relationship between controllers using a master and a slave function.
- Each joint set identifier is associated with a controller that manages the components belonging to that joint set.
- Controllers store information about:
  - the components in the joint set,
  - master and slave controller relationships,
  - the weights associated with each component.
- Controllers may be deployed on a single machine or distributed across multiple machines.

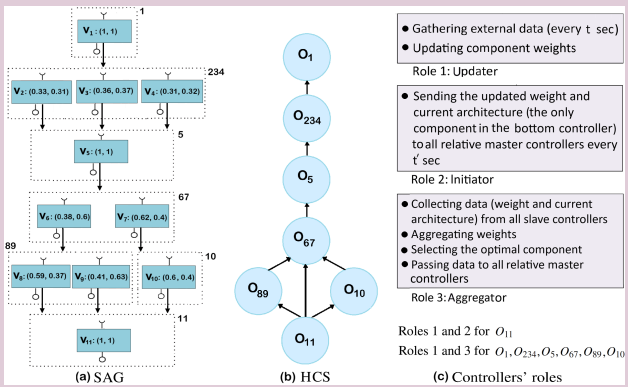
Decentralised Reconfiguration

Each controller executes three protocols that collectively enable decentralised architectural selection.

- **Update protocol.** This protocol periodically gathers environmental data and updates the weights of components.
- **Initiation protocol.** Executed by the bottom controller, this protocol determines the optimal component locally and sends the result to its master controllers.
- **Aggregation protocol.** Master controllers collect results from their slave controllers, compute compound weights and select the optimal component among their alternatives.

The selected component is appended to the current architecture and the result is propagated upwards in the hierarchy until the top controller determines the final architecture.

Fig 2: SAG vs HCS and Controller Roles



Conclusions

- This work presented a decentralised framework for runtime architectural selection.
- The framework enables systems to dynamically adapt their architecture without relying on a central controller.
- This approach is particularly suitable for dynamic environments such as cloud systems and IoT platforms.
- Future work will focus on:
  - empirical evaluation of execution time
  - scalability analysis
  - validation in real-world distributed systems

Search-based Software Testing for Drone Applications: An Experience with the Simulink Environment

Annalisa Sergi[1], Yousef Ahmed Abdel Rahman Shoeib[1], Andrea Bombarda[1], Nunzio Marco Bisceglia[1] and Claudio Menghi[1,2]

[1]University of Bergamo, Bergamo, Italy
[2]McMaster University, Hamilton, Canada

a.sergi2@studenti.unibg.it, y.shoeib@studenti.unibg.it, andrea.bombarda@unibg.it, nunziomarco.bisceglia@gssi.it, claudio.menghi@unibg.it

Introduction

Drones are cyber-physical systems used in various applications, from aerial inspection to search-and-rescue missions.
Autonomous flying is a desirable feature for drones, as they are frequently used in monitoring and inspection of large and isolated areas. In such situations, the GPS signal is not guaranteed; hence, other methods of local position feedback are required.
We used Search-Based Software Testing (SBST), a popular approach for testing CPS models, to find a requirement violation in the control logic.

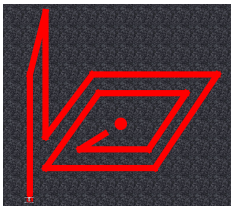
Drone Study Subject

Controlled system

The controlled system is the Parrot SA Mambo Fly Minidrone, produced by Parrot SA and used in the MathWorks Minidrone Competition.

Requirements

The competition requires participants to develop a line-following navigation algorithm on a track like the figure shown. The rules specify a minimum distance between segments of 20 cm and minimum angle of 15 degrees, also intersections are not allowed.

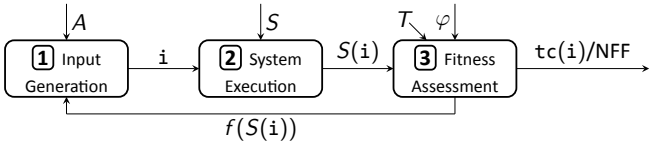


Controller

We developed three versions of the drone controller logic.

ID	Description and supported tracks
V1	Checks for the line in all four directions: forward, back, left, and right. Designed for long segments and 90-degree angles aligned with the x and y axes.
V2	Follows the line until the segment ends, then rotates the view to align the next segment. Supports medium segments and angles above 60-degree.
V3	Detects segment ends and boundaries to choose the turn, then rotates until the next segment is aligned. Works with advanced tracks.

Testing the Controller with SBST



Algorithm 1 Input Generation Component.

```
1: function GENERATETRACK(inpRanges,prevSample)
2:     curSample=GETNEWSAMPLE(inpRanges,prevSample);
3:     while CHECKTRACK(curSample) == 0
4:         curSample =GETNEWSAMPLE(inpRanges,prevSample);
5:     end
6:     track=INTERPOLATE(curSample)
7:     return track
8: end function
```

Algorithm 2 Track Interpolation.

```
1: function INTERPOLATE(curSample)
2:     points = ARRTOPOINTSMATRIX(curSample);
3:     [tiles,land] = LINEPATCHES(points);
4:     LINES2WRL(tiles,land);
5: end function
```

Evaluation

Experimental setup

$\psi := \Box(timeOutsideLine \leq 250) \wedge \Diamond(completed = 1) \wedge \Box(distance \leq 25)$.

Parameter	Value
Search algorithms	UR, SA
Number of runs	10
Maximum number of iterations per run	1500
Time budget (T)	300 s

Results

	UR			SA		
Controller	**FR**	$\widehat{S}$	$\widehat{S}$	**FR**	$\widehat{S}$	$\widehat{S}$
Version 1	10/10	1.0	1.2	10/10	1.0	3.5
Version 2	10/10	1.0	1.4	10/10	1.0	1.5
Version 3	10/10	45.0	69.1	10/10	122.0	132.6

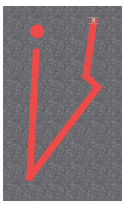
RQ1: Effectiveness

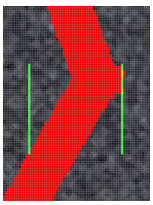
Our results confirm SBST's effectiveness in generating failure revealing tests. They also show that UR outperforms SA for controller version V3, while for V1 and V2 their effectiveness is comparable.

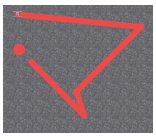
RQ2: Usefulness

The manual inspection confirmed the usefulness of the failure-revealing test cases: all the behaviors identified by the framework were violating the requirements and helped identify weaknesses in the controller. SBST even found violations in version V3, previously considered correct.


Failure case 1


Failure case 2


Failure case 3


Failure case 4

Conclusions

We adapted the SBST tool STaLiRo **for track generation** and for evaluating three software controllers against the requirements.
Using Uniform Random and Simulated Annealing search strategies, **SBST consistently uncovered failure-revealing test cases**, including subtle violations in the final competition–ready controller (V3) **that manual validation had missed**. The generated tests exposed weaknesses on short segments with acute angles, alignment/turning edge cases, and artifacts from track construction. **These results highlight the need for domain-tailored input generation to apply SBST across CPS domains** and show that **integrating SBST throughout development helps prevent overconfidence in manual testing**.

Tool Availability

GitHub
Source Code

Zenodo
Replication package

FoSSaCS 2026 Posters

Interaction Improvement

Adrienne Lancelot, Giulio Manzonetto, Guy McCusker and Gabriele Vanoni

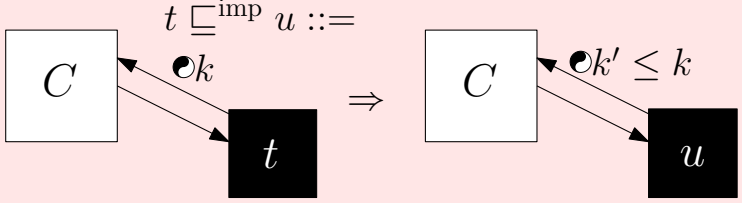
Studying the contextual preorder in a cost-aware manner:
counting interactions between the program and its execution environment

ignoring internal computation steps

syntactic difference between internal and interactive computations (of [1])

Interaction Improvement

$t \sqsubseteq^{\text{imp}} u ::=$



Formally:

$$\forall C \in \mathcal{C}_{\circ\bullet}, k \in \mathbb{N},\ C\langle \overline{t}^{\bullet} \rangle \Downarrow^{\bullet k}_{\mathtt{h}_{\circ\bullet}} \Rightarrow \exists k' \leq k,\ C\langle \overline{u}^{\bullet} \rangle \Downarrow^{\bullet k'}_{\mathtt{h}_{\circ\bullet}}$$

The Checkers λ-Calculus

playing with λ's and @'s

Colors $c ::= \bullet \mid \circ$

$\Lambda_{\bullet\circ} \ni t, u ::= x \mid \lambda_c x.t \mid t \cdot^c u$

Silent Steps:

$$(\lambda_\bullet x.t) \bullet u \to_\tau t\{x \leftarrow u\}$$
$$(\lambda_\circ x.t) \circ u \to_\tau t\{x \leftarrow u\}$$

Interaction Steps:

$$(\lambda_\bullet x.t) \circ u \to_\bullet t\{x \leftarrow u\}$$
$$(\lambda_\circ x.t) \bullet u \to_\bullet t\{x \leftarrow u\}$$

Notation: $t \Downarrow^{\bullet k}_{\mathtt{h}_{\circ\bullet}}$ if t terminates for head reduction in k interaction steps.

A model for interaction improvement, via *non-idempotent intersection types*

Relational Checkers Type Semantics

Linear types $\mathsf{L}, \mathsf{L}' ::= \mathsf{A} \mid \mathsf{M} \xrightarrow{\mathsf{c}} \mathsf{L}$ where, $\mathsf{c} \in \{\circ, \bullet\}$

Multi types $\mathsf{M}, \mathsf{N} ::= [\mathsf{L}_1, \dots, \mathsf{L}_n]$ $n \geq 0$

$$\frac{}{x : [\mathsf{L}] \vdash^0_\bullet x : \mathsf{L}}\ \mathsf{ax} \qquad \frac{(\Gamma_i \vdash^{k_i}_\bullet t : \mathsf{L}_i)_{i \in I} \quad I \text{ finite}}{\uplus_{i \in I} \Gamma_i \vdash^{\sum_{i \in I} k_i}_\bullet t : [\mathsf{L}_i]_{i \in I}}\ \mathsf{many} \qquad \frac{\Gamma, x : \mathsf{M} \vdash^k_\bullet t : \mathsf{L}}{\Gamma \vdash^k_\bullet \lambda_{\mathsf{c}} x.t : \mathsf{M} \xrightarrow{\mathsf{c}} \mathsf{L}}\ \lambda$$

$$\frac{\Gamma \vdash^{k_1}_\bullet t : \mathsf{M} \xrightarrow{\mathsf{c}} \mathsf{L} \quad \Delta \vdash^{k_2}_\bullet u : \mathsf{M}}{\Gamma \uplus \Delta \vdash^{k_1+k_2}_\bullet t \cdot^{\mathsf{c}} u : \mathsf{L}}\ @_\tau \qquad \frac{\Gamma \vdash^{k_1}_\bullet t : \mathsf{M} \xrightarrow{\mathsf{c}} \mathsf{L} \quad \Delta \vdash^{k_2}_\bullet u : \mathsf{M}}{\Gamma \uplus \Delta \vdash^{k_1+k_2+1}_\bullet t \cdot^{\mathsf{c}^\perp} u : \mathsf{L}}\ @_\bullet$$

Mismatch between Improvement and Type Semantics:

$\lambda_\bullet y.x \bullet y \sqsubseteq^{\text{imp}}_\bullet x$ but $\exists(\Gamma, \mathsf{L})$ such that $\Gamma \vdash_\bullet \lambda_\bullet y.x \bullet y : \mathsf{L}$ and $\Gamma \not\vdash_\bullet x : \mathsf{L}$

Repainting Arrow Types

$$\frac{\mathsf{p} \in \{+, -\}}{\mathsf{A} \leq^{\mathsf{p}}_0 \mathsf{A}} \qquad \frac{\mathsf{M}' \leq^-_{k_1} \mathsf{M} \quad \mathsf{L}' \leq^+_{k_2} \mathsf{L}}{\mathsf{M}' \xrightarrow{\circ} \mathsf{L}' \leq^+_{k_1+k_2+1} \mathsf{M} \xrightarrow{\bullet} \mathsf{L}} \qquad \frac{\mathsf{M}' \leq^{\neg \mathsf{p}}_{k_1} \mathsf{M} \quad \mathsf{L}' \leq^{\mathsf{p}}_{k_2} \mathsf{L}}{\mathsf{M}' \xrightarrow{\mathsf{c}} \mathsf{L}' \leq^{\mathsf{p}}_{k_1+k_2} \mathsf{M} \xrightarrow{\mathsf{c}} \mathsf{L}} \qquad \frac{\mathsf{L}'_1 \leq^{\mathsf{p}}_{k_1} \mathsf{L}_1 \quad \cdots \quad \mathsf{L}'_n \leq^{\mathsf{p}}_{k_n} \mathsf{L}_n}{[\mathsf{L}'_1, \dots, \mathsf{L}'_n] \leq^{\mathsf{p}}_{k_1+\dots+k_n} [\mathsf{L}_1, \dots, \mathsf{L}_n]}$$

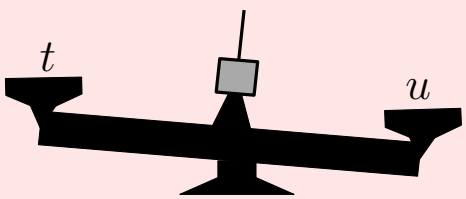
Polarized Whitercheaper Improvement

$t \sqsubseteq^{\text{pwc}}_\bullet u$ if $\forall(\Gamma, \mathsf{L}, k)$ such that $\Gamma \vdash^k_\bullet t : \mathsf{L} \Rightarrow \exists(\Gamma', \mathsf{L}', k')$ such that $\Gamma \vdash^{k'}_\bullet u : \mathsf{L}$, $\Gamma' \vdash \mathsf{L}' \leq^+_d \Gamma \vdash \mathsf{L}$ and $k \geq k' + d$

Example:

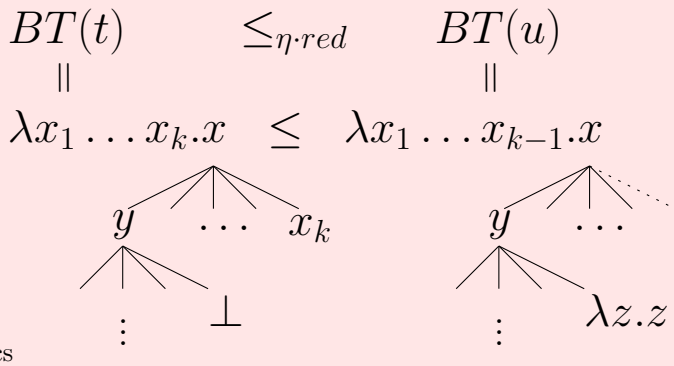
$x : [\mathbf{0} \xrightarrow{\circ} \mathsf{L}] \vdash^1_\bullet \lambda_\bullet y.x \bullet y : \mathbf{0} \xrightarrow{\bullet} \mathsf{L}$ and $x : [\mathbf{0} \xrightarrow{\circ} \mathsf{L}] \vdash^{k'}_\bullet x : \mathbf{0} \xrightarrow{\circ} \mathsf{L}$ such that $x : [\mathbf{0} \xrightarrow{\circ} \mathsf{L}] \vdash \mathbf{0} \xrightarrow{\circ} \mathsf{L} \leq^+_1 x : [\mathbf{0} \xrightarrow{\circ} \mathsf{L}] \vdash \mathbf{0} \xrightarrow{\bullet} \mathsf{L}$

Characterizing the Costs at Play in Relational Semantics



Interaction Improvement

via $\sqsubseteq^{\text{pwc}}_\bullet$



[2]

Relational Semantics
Uncolored Intersection Type Semantics

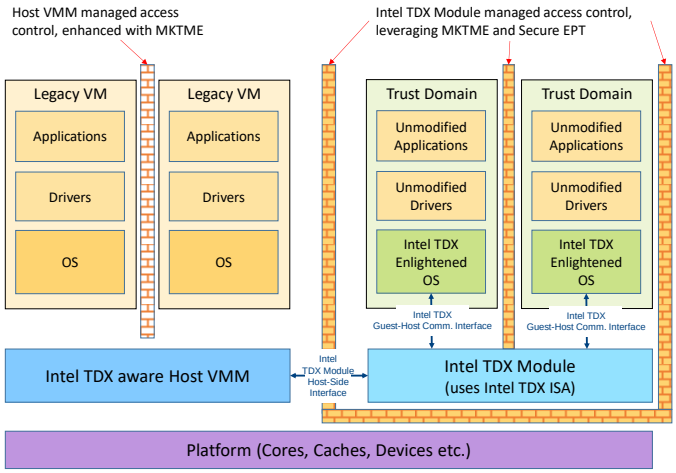
[1] Accattoli, Lancelot, Manzonetto, Vanoni. **Interaction equivalence**, 2025. *Proc. ACM Program. Lang. 9(POPL).*
[2] Breuvart, Manzonetto, Ruoppolo. **Relational graph models at work**, 2018. *Log. Methods Comput. Sci. 14(3).*

TACAS 2026 Posters

A Case Study in Firmware Verification
Applying Formal Methods to Intel® TDX Module

Dirk Beyer[1], Po-Chun Chien[1], Bo-Yuan Huang[2],
Nian-Ze Lee[3,1], and Thomas Lemberger[1]
[1]LMU Munich [2]Intel INT31 [3]National Taiwan University

Intel Trust Domain Extensions



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

Example ABI Specification

Excerpted specification of `TDH.MNG.KEY.CONFIG`

- **Input Operands**
  RAX: SEAMCALL instruction leaf number and version
  RCX: The physical address of a TDR page (HKID bits must be 0)
- **Output Operands**
  RAX: SEAMCALL instruction return code
  Other registers: Unmodified
- **State-Operand Information**
  TDR page: Explicitly accessed, read/write permission, 4KB-alignment
  Key Encryption Tables: Implicitly accessed
- **Completion-Status Codes**
  `TDX_OPERAND_INVALID`: An input operand of the ABI is invalid
  `TDX_LIFECYCLE_STATE_INCORRECT`: In an incorrect lifecycle state
  `TDX_KEY_GENERATION_FAILED`: Failed to generate a random key

Challenges for C Verifiers

- Extensive use of nested type punning (`union` types)
  – Many tools fail already at frontend parsing
- Initialization and preconditions for complex structures
  – Native havocking primitives greatly improves tool effectiveness
  – Constraints over array elements need standardized annotations (e.g., ∀ quantifier) for better effectiveness in automated tools.
- Handling of compiler attributes for memory-layout control
  – E.g., `aligned(n)` and `__packed__`
- Handling of inline assembly code
- Need for standardized annotation scheme for standard library functions (e.g., `memcpy`) and system calls
- Limited effectiveness for negative-space safety properties
  – Current tools lack semantic slicing of unreachable core logic

Contributions

- Adapted code-level methodology for firmware
- Developed proof harnesses and assembled verification tasks for TDX Module v1.5.05 using HarnessForge [2]
- Released harnesses and tasks publicly [3]

More Information

Proof Harness

- Initialize global data and assume preconditions
  – Havoc memory region byte by byte
  – Havoc object field by field
  – Use verifier builtins (e.g., `__CPROVER_havoc_object` in CBMC)
- Mock access to externally defined data and model inline assembly
- Assert postconditions

```
1  // proof harness components
2  void hmkc__setup() {
3    fv_setup_module_state();
4    fv_setup_tdr();
5    fv_setup_tdcs();
6  }
7  void hmkc__teardown() {
8    fv_teardown_tdcs();
9    fv_teardown_tdr();
10   fv_teardown_module_state();
11 }
12 void hmkc__invalid_input_rcx__precond() {
13   ASSUME(
14     !is_valid_hmkc_input_rcx() &&
15     is_valid_hmkc_state_metadata() &&
16     is_valid_hmkc_state_lifecycle());
17 }
18 void hmkc__invalid_input_rcx__postcond() {
19   ASSERT(
20     get_local_data()->vmm_regs.rax ==
21     api_error_with_operand_id(
22       TDX_OPERAND_INVALID,OPERAND_ID_RCX));
23 }
24 // auto-generated task entry point
25 void main() {
26   hmkc__setup();
27   hmkc__invalid_input_rcx__precond();
28   hmkc__function_call();
29   hmkc__invalid_input_rcx__postcond();
30   hmkc__teardown();
31 }
```

Verification Results (#solved)

	Interface function	#tasks	Mem. havoc.	Obj. havoc.	Tool-spec. havoc.
Guest-side TDG	MR.REPORT	10	0	0	5
	SERVTD.WR	10	0	0	7
	SYS.RD	10	0	0	8
	VM.WR	17	0	0	14
	VP.ENTER	18	0	0	12
	VP.VMCALL	10	0	0	8
Host-side TDH	EXPORT.RESTORE	10	3	0	7
	IMPORT.ABORT	9	2	0	7
	MNG.ADDCX	33	5	2	11
	MNG.CREATE	18	3	0	17
	MNG.INIT	33	5	0	11
	MNG.KEY.CONFIG	18	4	0	16
	MNG.KEY.FREEID	17	2	0	16
	MNG.VPFLUSHDONE	21	4	0	19
	MR.FINALIZE	29	3	0	7
	PHYMEM.PAGE.RECLAIM	25	0	0	5
	SYS.CONFIG	29	0	0	8
	SYS.INIT	17	0	0	13
	SYS.KEY.CONFIG	13	10	0	13
	SYS.SHUTDOWN	17	0	0	13
	SYS.UPDATE	25	0	0	19
	VP.ENTER	29	0	0	9
	Overall	418	41	2	245

References

[1] Intel Trust Domain Extensions, https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html
[2] Beyer, D., Chien, P.C., Huang, B.Y., Lee, N.Z., Lemberger, T.: Harnessforge: Automated extraction of verification tasks from industry-scale software projects (2026), (manuscript available upon request)
[3] Beyer, D., Chien, P.C., Huang, B., Lee, N.Z., Lemberger, T.: The Intel TDX Module benchmark set. Zenodo (2025)

DeGAS: Gradient-Based Optimization of Probabilistic Programs without Sampling

Francesca Randone[1], Romina Doz[2], Mirco Tribastone[3], Luca Bortolussi[2]
[1]TU Wien, [2]University of Trieste, [3]IMT School for Advanced Studies Lucca
francesca.randone@tuwien.ac.at, romina.doz@phd.units.it

Why not just sampling?

Probabilistic programs expose tunable **parameters** — thresholds, noise levels, control gains — that must be optimized to meet objectives such as maximizing data likelihood or satisfying reachability constraints.

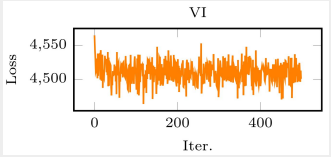
Current practices rely on **sampling methods** (Monte Carlo Markov Chain, Variational Inference, etc.).

These work well in smooth models, but fail when programs contain **continuous-variable branches** and **hard observations**: *gradients become discontinuous or zero almost everywhere, and samplers can get stuck or oscillate*.

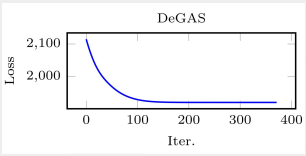
In the following program we want to optimize μ_1 and μ_2 for data likelihood

$$v \sim gauss(\mu_1, 5)$$
$$\textbf{if } v > 0 \textbf{ then:}$$
$$\quad y \sim gauss(\mu_2, 1)$$
$$\textbf{else:}$$
$$\quad y \sim gauss(-2, 1)$$

Loss plot with Variational Inference (discontinuous gradient)



Loss plot with DeGAS (smoothing)



DeGAS: Differentiable Gaussian Approximate Semantics

Instead of sampling, DeGAS approximates the program's posterior using a **differentiable Gaussian-Mixture (GM) semantics**, built on the moment-matching strategy used by SOGA[(1)].

Every program instruction — deterministic and random assignments, conditional statements and conditioning — transforms a distribution, approximated by a GM, into a new GM, **maintaining differentiability with respect to the parameters**.

Key insight: Non-differentiable transformations are ***smoothed*** by a parameter ϵ.

Before smoothing (non differentiable)	`y = x + 1`	`x = c`	`x = flip(p)` `x = gm([0.5, 0.5], [0, 1], [0,0])`	`observe(y == 1.)`

After smoothing (differentiable)	`y = x + 1 + gauss(0, ϵ)`	`x = gauss(c, ϵ)`	`x = gm([0.5, 0.5], [0, 1], [ϵ, ϵ])`	`sm-observe(y == 1.)`
				`observe(y==1.)`
				`y = gauss(1., ϵ)`

Formal Guarantees (Theorems 1 & 2)

Differentiability: For any valid program P(θ), the posterior distribution computed by DeGAS is differentiable in θ.

Convergence: If smoothing is applied *consistently*, the error introduced by the smoothing vanishes.

Using DeGAS in Python

1. Write a SOGA program

`thermostat.soga`

```
...
if isOn > 0 {
  if newT > _tOff {
    isOn = -1;
  } else {
    skip;
  } end if;
} else {
  if newT < _tOn {
    isOn = 1;
  } else {
    skip;
  } end if;
} end if;
...
```

2. Compile it to a smooth control flow graph

```
compiledFile = compile2SOGA('thermostat.soga')
cfg = produce_cfg(compiledFile)
smooth_cfg(cfg)
```

3. Initialize parameters

```
init_params = {'tOn':  15., 'tOff':22.}
params_dict = initialize_params(init_params)
```

4. Define a custom loss *(function of a distribution)*

```
loss = lambda dist : neg_log_lik(traj, dist)
```

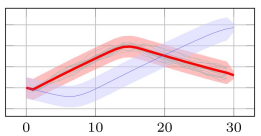
Key insight: DeGAS can optimize ***any differentiable loss***, not just likelihoods (e.g., reachability probabilities, probabilistic targets, etc.)

4. Run the optimization loop *(default: torch.optim.Adam)*

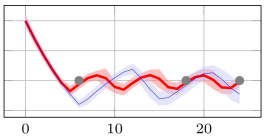
```
loss_list, time_opt, n_iters =
  optimize(cfg, params_dict,
    loss, n_steps=60, lr=0.1)
```

5. Inspect results *(blue: initial, red:optimized)*

Likelihood optimization



Probabilistic target (reach points)



[(1)]Randone, Francesca, et al. "Inference of probabilistic programs with moment-matching gaussian mixtures." *Proceedings of the ACM on Programming Languages* 8.POPL (2024): 1882-1912.

TACAS 2026

Pre-print: Artifact: DeGAS GitHub:

ReCheck: Automated Contextual Improvement Verification for Functional Calculi across User-Defined Operational Semantics

Makoto Hamana | Kento Emoto
Kyushu Institute of Technology, Japan

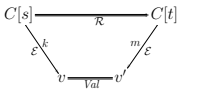
Problem and Goal

- Program optimizations and rewrite rules should preserve **efficiency**, not only extensional equivalence.
- Sands' *contextual improvement* requires that replacing s by t never increases evaluation cost in any surrounding program context.
- Existing proofs are often manual and tied to one specific operational semantics.
- We target **user-defined operational semantics**: syntax classes, evaluation contexts, evaluation rules, and refinement rules.

Goal: verify contextual improvement *automatically* by critical-pair analysis.

Contextual Improvement

A set of refinement rules $\mathcal{R}$ is a contextual improvement with respect to evaluation rules $\mathcal{E}$ if



closed terms

with $k \geq m$.

same observable value, with no extra evaluation cost

Key difficulty. The definition quantifies over *all contexts* C.

TERS: Term Evaluation and Refinement Systems

A TERS specifies:

- a signature,
- syntax classes (values, computations, answers, handlers, ...),
- evaluation contexts $\mathcal{E}ctx$,
- evaluation rules $\mathcal{E}$,
- refinement rules $\mathcal{R}$.

$$\frac{(l \to r) \in \mathcal{E} \quad E \in \mathcal{E}ctx}{E[l\theta] \to_{\mathcal{E}} E[r\theta]} \qquad \frac{(l \Rightarrow r) \in \mathcal{R} \quad C \in \mathcal{C}}{C[l\theta] \to_{\mathcal{R}} C[r\theta]}$$

This cleanly separates **deterministic evaluation** from **global refinement**.

Main Theorem

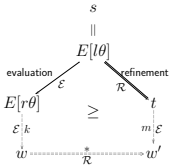
(Muroya,Hamana 2024) If

1. $\mathcal{E}$ is deterministic,
2. $\mathcal{R}$ is value-invariant,
3. the TERS $(\mathcal{E}, \mathcal{R})$ is **locally coherent**,

then $\mathcal{R}$ is a **contextual improvement** with respect to $\mathcal{E}$.

This reduces a universal semantic property to a **local overlap condition** between evaluation and refinement.

Local Coherence Criterion



with $1 + k \geq m$.

Lemma. For a **well-behaved** TERS, local coherence holds iff every critical pair between $\mathcal{E}$ and $\mathcal{R}$ is joinable.

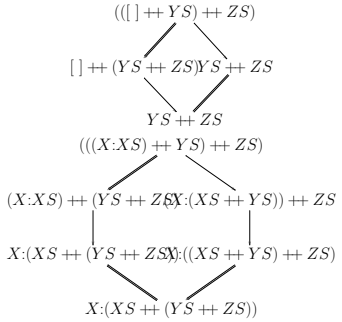
Consequence: contextual improvement is checked by finite overlap analysis.

Running Example: Append Fusion

Values $V ::= [\,] \mid V : V$
Eval. ctxts $E ::= \mid E +\!\!+ t \mid V +\!\!+ E \mid E : t \mid V : E$
Eval. rules $[\,] +\!\!+ ys \to ys$
$(x : xs) +\!\!+ ys \to x : (xs +\!\!+ ys)$
Refinement $(xs +\!\!+ ys) +\!\!+ zs \Rightarrow xs +\!\!+ (ys +\!\!+ zs)$

Associativity of append is treated as an *efficiency-improving* rewrite under deterministic evaluation.

Critical Pairs for Append Fusion



Both critical pairs are joinable.

Tool Support: ReCheck

- ReCheck extends SOL and automates critical-pair analysis for contextual improvement.
- Input: syntax classes, evaluation contexts, evaluation rules, and refinement rules.
- Output: overlaps, joinability checks, and non-joinable counterexamples.

Specification → critical pairs → joinability → certificate

Case Studies and Scope

Untyped call-by-value λ-calculus

Values $V ::= x \mid \lambda x.M$
Eval. ctxts $E ::= \mid E\,M \mid V\,E$
Eval. rule $(\lambda x.M)\,V \to M[x := V]$

The same framework also supports second-order TERS presentations and refinements such as

$$\lambda(x.M[x])@V \Rightarrow M[V], \qquad \lambda(x.V@x) \Rightarrow V.$$

Map/map fusion

```
map f []     = []
map f (x:xs) = f x : map f xs
(.) f g x    = f (g x)
RULE "map/map":
map f (map g xs) => map (f . g) xs
```

Lazy program optimization

```
(rep1)  replicate(z)      => []
(rep2)  replicate(s(N))   => s(z) : replicate(N)
(take1) take(z, XS)       => []
(take2) take(s(N), [])    => []
(take3) take(s(N), X:XS)  => X : take(N,XS)
(ones)  ones              => s(z) : ones
(conj)  take(N, ones)     => replicate(N)
```

ReCheck reports 2 critical pairs, with 1 non-joinable; after adding

$$\texttt{take}(N, s(z){:}\texttt{ones}) \Rightarrow \texttt{replicate}(N),$$

all critical pairs become joinable.

Take-Home Message

- Contextual improvement can be reduced to **joinability of critical pairs**.
- This yields a **semantics-parametric** and **automation-friendly** verification method.
- ReCheck is effective for user-defined TERS, including ADT-heavy examples where SMT-centered tools are often less suitable.

Tools	Typical strength
Mochi / RCaml	safety or refinement-type verification with SMT
Timbuk	TRS + tree-automata completion
ReCheck	contextual improvement for user-defined TERS, strong on ADTs

Keywords: contextual improvement, operational semantics, evaluation contexts, program optimization, critical pairs, ReCheck

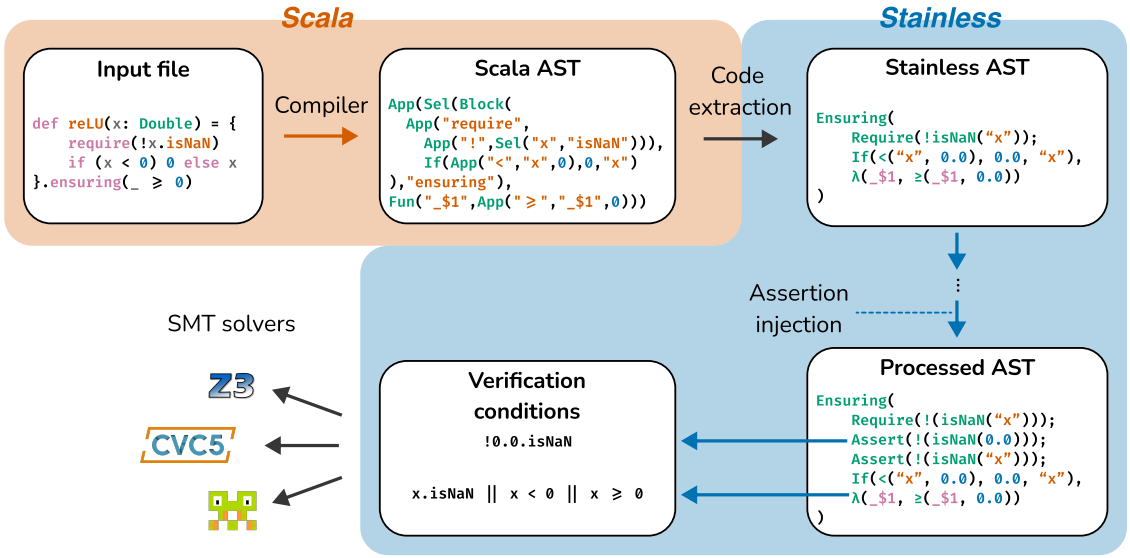
Verifying Floating-Point Programs in Stainless

Andrea Gilot, Axel Bergström, Eva Darulova

UPPSALA UNIVERSITET

We add support for bit-precise floating-point reasoning to the Stainless verifier for Scala, enabling sound reasoning about numerical programs.

Motivation. Floating-point numbers are unintuitive: `NaN` ≠ `NaN` and rounding errors accumulate, introducing bugs in the code.



Polymorphic equality

Problem: Reflexivity of polymorphic equality is unsound when mixed with floating points.

T = T ⇒ NaN = NaN ✗

Solution: We extend Stainless with a **@noeq** annotation to treat polymorphic equality as an uninterpreted symbol.

```
@noeq
def id[T](t: T) = {t}
.ensuring(res ⇒ res == t)
// postcondition fails ...

id[Float];
// ... but instantiation
succeeds
```

Case Study: Verified Math Standard library

Problem: SMT-solvers do not support transcendental functions.

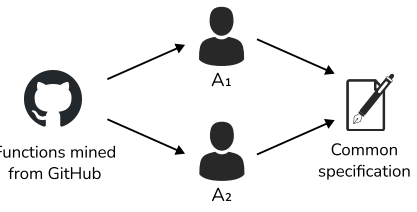
Solution: We verify properties of FdLibm implementations of transcendental functions.

```
def cos(x: Double): Double = {
  // FdLibm implementation
}.ensuring(res =>
  (x.isNaN || x.isInfinity) == res.isNaN
  && (res.isNaN || -1.0d <= res && res <= 1.0d)
)
```

Previous work **axiomatised** these properties, we **verify** them.

User benchmarks

We evaluate Stainless on **user code** by mining floating-point functions on GitHub.



We **independently** add contracts to mined functions, then converge on a common specification.

Bitwuzla

We add Bitwuzla support to Stainless

On our benchmark set Bitwuzla is **significantly faster** than other solvers but **lacks support** for many features of the language.

Proportion of VCs solved on FdLibm benchmarks

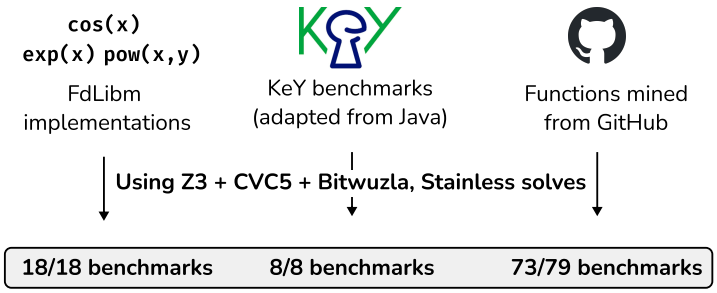
Solver	%VCs solved
Z3	78.1%
CVC5	88.7%
Bitwuzla	84.8%

10.6% of VCs are solved **by Bitwuzla only**.

Assertion injection

Non-handling of NaN is a frequent source of bugs. Stainless **automatically** checks that NaN values are not involved in comparisons, equalities and type casts.

Evaluation



Funded by the European Union
European Research Council

Read the paper!

Robustness Verification of Graph Neural Networks Via Lightweight Satisfiability Testing

Chia-Hsuan Lu [1] Tony Tan [2] Michael Benedikt [1]

[1]University of Oxford [2]University of Liverpool

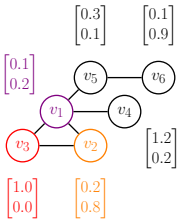
Graph Neural Networks

GNNs are defined as a sequence of layers.

Input. A labeled graph, where each node v is initialized with a feature vector from its label embedding.

Layer computation. At each layer, GNNs update node features $\xi^{(\ell)}(v)$ by:

1. Aggregate: collect previous-layer features from neighbors;
2. Affine transformation: combine aggregated and own previous-layer features;
3. Activation: apply non-linearity (ReLU).



Output. After L layers, each node is assigned a class:

$$\hat{c}(\mathcal{G}, v) := \arg\max_{1 \le i \le C} \left(\xi^{(L)}(v)\right)[i].$$

For example, $\xi^{(\ell+1)}(v_3) = \mathsf{ReLU}\left(\mathbf{C}^{(\ell)}\begin{bmatrix}1.0\\0.0\end{bmatrix} + \mathbf{A}^{(\ell)}\left(\begin{bmatrix}0.1\\0.2\end{bmatrix} + \begin{bmatrix}0.2\\0.8\end{bmatrix}\right) + \mathbf{b}^{(\ell)}\right)$.

Adversarial Robustness of GNNs

Goal. Does the model's prediction remain unchanged under small input perturbations?

Unlike standard feedforward neural networks, GNN predictions may change under:

- feature perturbations: modifying node features;
- structural perturbations: inserting or deleting edges. (In this work, we focus only on this.)

Setting

- a labeled graph $\mathcal{G}$;
- a fragile-edge set F: a set of edges that can switch from being edges to non-edges and vice versa;
- a budget Δ: the maximum number of switching allowed.

The admissible perturbation space consists of all graphs obtainable from $\mathcal{G}$ by changing at most Δ edges in F.

A GNN is adversarially robust for node v if, for every perturbed graph $\mathcal{G}'$ in this space,

$$\hat{c}(\mathcal{G}', v) = \hat{c}(\mathcal{G}, v).$$

Graphs with Unknown Edges

An incomplete graph is a labeled graph $\mathcal{H} = \langle V, E, E^{\mathsf{Unk}}, E^{\mathsf{Non}}, X\rangle$, where E, E^{Unk}, and E^{Non} form a partition of $V \times V$, and X denotes the labels of the nodes.

Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be incomplete graphs with the same V and X. $\mathcal{H}_2$ refines $\mathcal{H}_1$ if it can be obtained from $\mathcal{H}_1$ by repeatedly converting an unknown edge into either a normal edge or a non-edge.

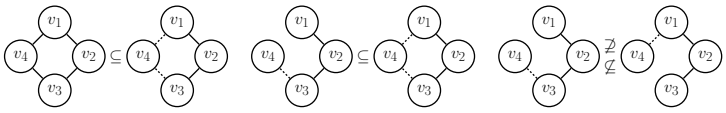
Observation: refinement defines a partial order on incomplete graphs with the same V and X.

Observation: there is a natural distance function on incomplete graphs with the same V and X

$$\mathsf{dist}(\mathcal{H}_1, \mathcal{H}_2) := |\{(v,u) \in V \times V \mid (v,u) \text{ is inconsistent between } \mathcal{H}_1 \text{ and } \mathcal{H}_2\}|.$$

Here, (v,u) is inconsistent if either

$$(v,u) \in E_1 \text{ and } (v,u) \in E_2^{\mathsf{Non}}, \quad \text{or} \quad (v,u) \in E_1^{\mathsf{Non}} \text{ and } (v,u) \in E_2.$$



d-radius Satisfaction Problem

Given an node property $\mathcal{Q}$, a normal graph $\mathcal{G}$, an incomplete graph $\mathcal{H}$, and a node v, we say $\mathcal{Q}$ is satisfied within radius d of $\mathcal{G}$ with respect to $\mathcal{H}$ at v if there is a normal graph $\mathcal{G}' \subseteq \mathcal{H}$, with $\mathsf{dist}(\mathcal{G}', \mathcal{G}) \le \mathsf{dist}(\mathcal{H}, \mathcal{G}) + d$ such that $\langle \mathcal{G}', v\rangle$ satisfies $\mathcal{Q}$.

The non-adversarial robustness of GNNs can be reduced to d-radius satisfaction:

Does there exist $\mathcal{G}' \subseteq \mathcal{H}$ with $\mathsf{dist}(\mathcal{G}', \mathcal{G}) \le \Delta$ such that $\langle \mathcal{G}', v\rangle$ satisfies $\mathcal{Q}$?

where

- $\mathcal{H}$ is the incomplete graph obtained by converting edges in F to unknown edges in $\mathcal{G}$;
- $\mathcal{Q}$ is the node property that holds on node v in $\mathcal{G}'$ if $\hat{c}(\mathcal{G}', v) \neq \hat{c}(\mathcal{G}, v)$.

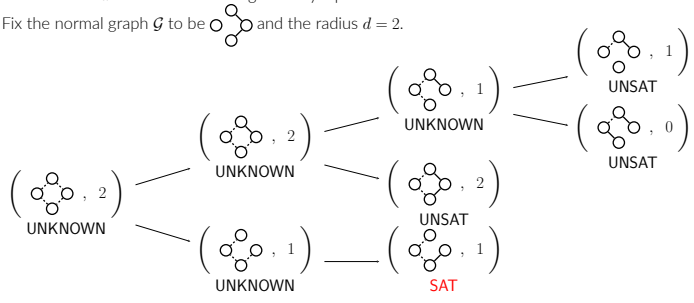
Partial Oracle

A partial oracle $\mathcal{O}$ is a function that receives an incomplete graph $\mathcal{H}$ and a budget d as input, and which outputs either SAT, UNSAT, or UNKNOWN, and satisfy the following conditions:

1. (Correctness for SAT) If $\mathcal{O}(\mathcal{H}, d)$ returns SAT, then there is a normal graph $\mathcal{G}' \subseteq \mathcal{H}$ with $\mathsf{dist}(\mathcal{G}', \mathcal{G}) \le \mathsf{dist}(\mathcal{H}, \mathcal{G}) + d$ and $\langle \mathcal{G}', v\rangle$ satisfies $\mathcal{Q}$.
2. (Correctness for UNSAT) If $\mathcal{O}(\mathcal{H}, d)$ returns UNSAT, then for each normal graph $\mathcal{G}' \subseteq \mathcal{H}$ with $\mathsf{dist}(\mathcal{G}', \mathcal{G}) \le \mathsf{dist}(\mathcal{H}, \mathcal{G}) + d$, $\langle \mathcal{G}', v\rangle$ does not satisfy $\mathcal{Q}$.
3. (Soundness on normal graphs or zero budget) If $\mathcal{H}$ is normal or $d = 0$, then $\mathcal{O}(\mathcal{H}, d)$ never returns UNKNOWN.

We can solve d-radius satisfaction guided by a partial oracle.

Fix the normal graph $\mathcal{G}$ to be  and the radius $d = 2$.



Algorithm for the d-radius Satisfaction

```
procedure Check(H, d)
    OracleResult ← O(H, d)
    if OracleResult is SAT then
        return SAT
    else if OracleResult is UNSAT then
        return UNSAT
    else
        Pick e ∈ E^Unk
        if e is an edge in G then
            H1 ← H by converting e into a normal edge.
            H2 ← H by converting e into a non-edge.
        else
            H1 ← H by converting e into a non-edge.
            H2 ← H by converting e into a normal edge.
        end if
        if (Check(H1, d) return SAT) or (Check(H2, d − 1) return SAT) then
            return SAT
        else
            return UNSAT
        end if
    end if
end procedure
```

There is a trade-off between oracle quality and cost:

- an exact oracle checks exponentially many normal graphs, but reduces the search to a single recursive call;
- a trivial oracle simply returns UNKNOWN for every non-normal graph, so the full recursion tree must be explored.

Lightweight Partial Oracle for the Robustness of GNNs

Our tool RobLight uses a polynomial-time partial oracle with two components:

- (Non-robustness tester) If $\langle \mathcal{G}', v\rangle$ satisfies $\mathcal{Q}$, returns SAT; otherwise, UNKNOWN.
  $\mathcal{G}'$ is obtained from $\mathcal{H}$ by converting each unknown edge e into either a normal edge if e is an edge in $\mathcal{G}$, or a non-edge otherwise.
- (Bound propagator) It computes over-approximate bounds in a bottom-up manner such that, for every normal graph $\mathcal{G}' \subseteq \mathcal{H}$,

  $$\underline{\xi}^{(\ell)}_{\mathcal{H}}(v) \le \underline{\xi}^{(\ell)}_{\mathcal{G}'}(v) \le \overline{\xi}^{(\ell)}_{\mathcal{H}}(v).$$

  After computing these bounds, check whether, for every $1 \le c' \neq c \le d^{(L)}$,

  $$\overline{\xi}^{(L)}_{\mathcal{H}}(v)[c] < \underline{\xi}^{(L)}_{\mathcal{H}}(v)[c'].$$

  If so, it returns UNSAT; otherwise, it returns UNKNOWN.

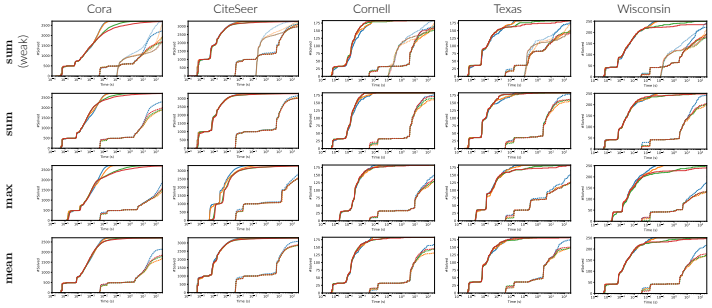
Optimization Strategic: Incremental Computation

Suppose we convert an unknown edge into an edge.

- At each layer ℓ, only the features of the red nodes may change.
- The final-layer feature of v depends only on the features of the blue nodes.

Hence, it suffices to update only the nodes that are both red and blue. The number of updates decreases from 24 to 8.

Some Experimental Results: Number of Solved Instances Versus Runtime



- Solid, dashed, and dotted lines represent RobLight, GNNev [LLK25], and SCIP-MPNN [HZCM24], respectively.
- Blue, orange, green, and red correspond to budgets 1, 2, 5, and 10, respectively.

Conclusion

Our contributions

- We show that combining heuristic search with lightweight solvers can outperform the state of the art, highlighting that current solvers underexploit structure in GNN verification problems.
- We extend the scope of robustness analysis for GNNs. To our knowledge, no prior tool supports exact analysis beyond 4-layer GNNs with hidden dimension 32.

Some future directions

- Abstract interpretation tools for tighter analysis.
- Combining both feature and structural perturbations.
- Developing heuristics for recursion-path selection based on graph-theoretic properties.

GitHub arXiv

Robust Verification of Concurrent Stochastic Games

Angel Y. He David Parker
Department of Computer Science, University of Oxford

Why robust verification?

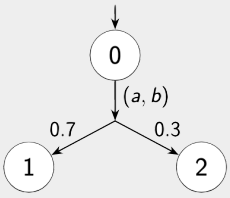
- Many autonomous systems are **concurrent**, **stochastic**, and operate under **uncertain** dynamics.
- Transition probabilities are **learned** from limited or noisy data, or arise from model **abstraction**.
- **Concurrent stochastic games** (CSGs) model multi-agent stochastic systems with simultaneous action choices.
- ⚠ Standard CSG verification assumes **exact transitions** → unrealistic

*How do we model **uncertainty** in CSGs, and perform **robust** verification (and synthesis) on these models?*

Model: Interval Concurrent Stochastic Games

CSG: $\mathcal{G} = (N, S, \bar{s}, A, \Delta, P)$ — zero-sum / nonzero-sum

N players simultaneously choose actions
Joint action determines probabilistic transition



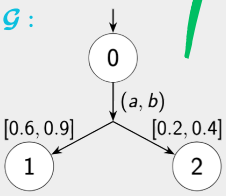
- (Minimax) **value, optimal strategies**
- **Nash Equilibria** (NE)
- **Social-welfare optimal NE (SWNE)**: NE that maximises players' total utility

Robust CSG (RCSG): $\mathcal{G} = (N, S, \bar{s}, A, \Delta, \mathcal{P})$ — uncertain transition function

$\mathcal{P}$ describes a set of possible transition functions

- ***Robust* value, *robust* optimal strategies** — holds under all possibilities / worst-case
- ***Robust* NE (RNE)**
- ***Robust* SWNE (RSWNE)**: NE that maximises worst-case total utility

$\mathcal{G}$:



Interval CSG (ICSG): $\mathcal{G} = (N, S, \bar{s}, A, \Delta, \check{P}, \hat{P})$
RCSG with interval uncertainty

$$\mathcal{P}_{sa} = \left\{ P_{sa} \in \mathcal{D}(S) \mid \forall s'.\, P_{sa}(s') \in [\check{P}_{sas'}, \hat{P}_{sas'}] \right\}$$

🎯 **Objectives**: (in)finite-horizon probabilistic & reward reachability

Robust Bellman Equations

Zero-sum: nature adversarial against one player, e.g., minimises player 1's utility — uncertainty

$$V(s) = \sup_{\sigma_1 \in \mathcal{D}(A_1(s))} \inf_{\sigma_2 \in \mathcal{D}(A_2(s))} \left\{ r(s,\sigma) + \sum_{a \in A(s)} \sigma_{sa} \inf_{P_{sa} \in \mathcal{P}_{sa}} \sum_{s' \in S} P_{sas'} V(s') \right\}$$

$V(s)$: player 1's value; σ_2: mixed strategies for optimality; inner infimum: nature's inner problem

Nonzero-sum: nature minimises total utility

$$V_+(s) = \sup_{\sigma \in \Sigma_{\varepsilon\text{-RNE}}} \left[r_+(s,\sigma) + \sum_{a \in A} \sigma_{sa} \inf_{P_{sa} \in \mathcal{P}_{sa}} \left\{ \sum_{s' \in S} P_{sas'} V_+(s') \right\} \right]$$

$V_+(s)$: players' total value; players optimise their own objectives

Contributions

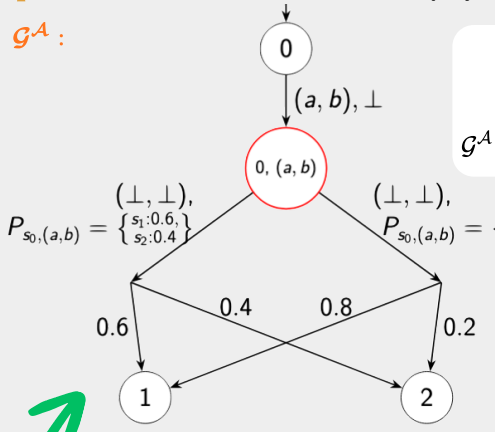
- Introduced **RCSGs** (with focus on ICSGs) + ***robust*** solution concepts
- **Value-preserving** reduction to **non-robust CSG** solving
- **Player-/nature-first** value-equivalence
- **Scalable** ICSG solver implemented in PRISM-games

Adversarial Expansion

*Model **nature as player 3** who chooses transitions adversarially*

- Nature aligns with opponent → **2-player CSG** — *via "deviation MDPs"*
- Nature minimises social welfare → **3-player CSG** + NE filtering

$\mathcal{G}^{\mathcal{A}}$:



$\mathcal{G} : s \xrightarrow{a,P} s'$ ⇝ $\mathcal{G}^{\mathcal{A}} : s \xrightarrow{a} (s,a) \xrightarrow{P_{sa}} s'$

Robust reasoning ⇝ *Standard CSG solving*

Main Theoretical Results

1. Adversarial expansion (ICSG $\mathcal{G}$ → CSG $\mathcal{G}^{\mathcal{A}}$) is **value-preserving**
2. **Player-first = nature-first** semantics: Equivalent robust value regardless of whether players or nature moves first
   - allows **robust dynamic programming**
   - i.e., fix nature's choice of *P* first, then solve induced CSG

➡ Enables algorithm to compute robust value of the ICSG by reusing CSG algorithms (without building $\mathcal{G}^{\mathcal{A}}$ explicitly!)

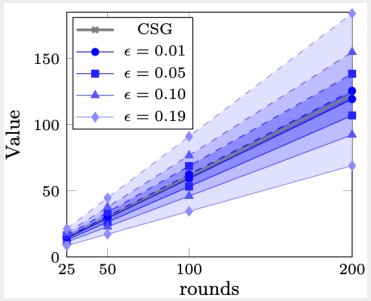
RVI/RBI update for state $s \in S$ in $\mathcal{G} = (N, S, \bar{s}, A, \Delta, \check{P}, \hat{P})$

```
for all a ∈ A(s) do
    P*_sa ← SolveInnerProblem(s, a, V_prev, P̌, P̂)
end for
Z ← ConstructNFG(P*_s, V_prev)
V_next[s] ← SolveNFG(Z)
```

Experimental Results

- Implementation in **PRISM-games**
- CSG benchmarks augmented with transition uncertainty

- **Zero-sum ICSGs**: similar scaling to non-robust CSGs
  - millions of states & transitions possible
  - < 2× overhead over CSGs
- **Nonzero-sum ICSGs:** more expensive but tractable for many instances



ICSG values vs. game size in *Intrusion Detection*

Future work

- **Non-rectangular / non-polytopic** uncertainty
- Richer **temporal** objectives (e.g., LTL)
- **Learning** robust learned controllers

Extended version: https://arxiv.org/pdf/2601.12003 TACAS 2026
angel-he 7angel4 angel.he@balliol.ox.ac.uk

EvolveGen: Algorithmic Level Hardware Model Checking Benchmark Generation through Reinforcement Learning

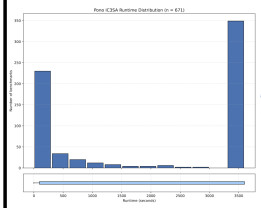
Guangyu Hu[1]*, Xiaofeng Zhou[1]*, Wei Zhang[1], Hongce Zhang[2]

[1]The Hong Kong University of Science and Technology [2]The Hong Kong University of Science and Technology (Guang Zhou)

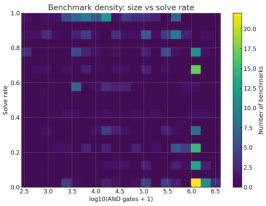
Email: ghuae@connect.ust.hk, xzhoubu@connect.ust.hk, wei.zhang@ust.hk, hongcezh@hkust-gz.edu.cn

I. Motivation

- Existing hardware model checking benchmarks are ***scarce***.
- Current benchmark suites have a ***polarized difficulty distribution***
- Current benchmark suites' ***difficulty is driven by scale***
- Prior fuzzers operate at the AIGER/BTOR2 level, cannot ***producing difficult and meaningful benchmarks***



polarized difficulty distribution

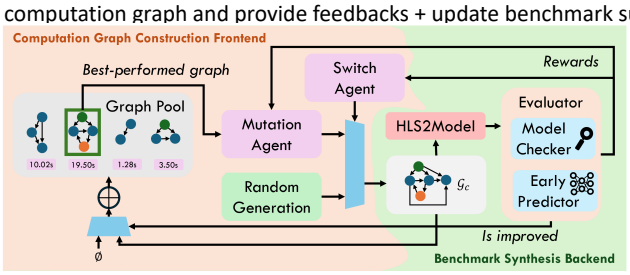


scale-driven difficulty

II. Framework

A hardware model checking benchmark generation framework

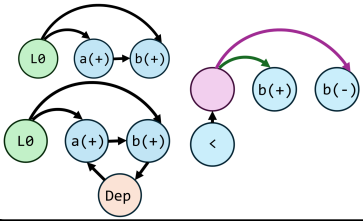
- A Computation Graph Construction Frontend to evolve the graph
- A benchmark synthesis backend to evaluated the generated computation graph and provide feedbacks + update benchmark suite



III. Methodology

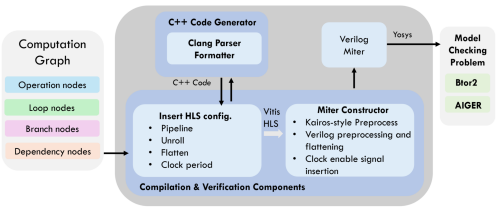
Computation Graph:

- nodes: operations / variables / loops / branches
- edges: data flows / code block membership
- attributes: variable type / variable bitwidth / loop implementation types



HLS2Model

- Input: Computation Graph;
- Output: Model checking problem in BTOR2/AIGER;
- Internal Tools: HLS / Yosys
- Convert computation graph to C -> HLS compiles C to RTL -> Kairos generate miter -> Yosys



Multi-Agent Strategy for Graph Exploration:

- Mutation agent: mutate the current best-performed graph
- Switch agent: switch between the evolved best-performed graph (exploitation) and the randomly generated new graph (exploration)

Graph Pool:

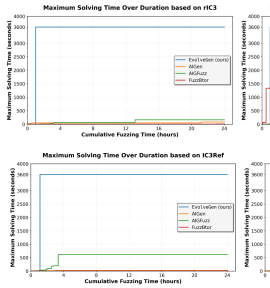
- Maintain the set of promising graphs;
- If a computation graph is observed with improvement, do update

Evaluator:

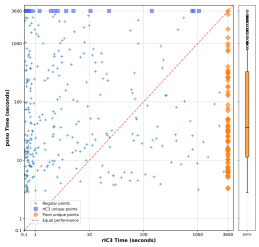
- Either predict the solving time with XGBoost (difficult cases) or use model checker to measure the real runtime (simple cases)

IV. Results

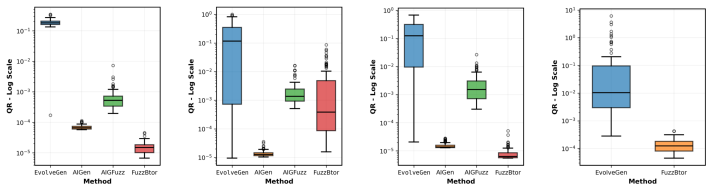
- Fuzzing efficiency



- Ability to Distinguish Performance Differences



- Benchmark qualities



(a) rIC3 (b) ABC-PDR (c) IC3Ref (d) Pono

V. Conclusion

- EvolveGen leverages reinforcement learning to generate hardware model checking benchmarks from a high-level algorithmic abstraction. By constructing computation graphs and using HLS to synthesize functionally equivalent but structurally distinct hardware designs, it creates challenging verification instances.
- Our experiments show that EvolveGen efficiently creates a diverse benchmark set in AIGER and BTOR2, effectively revealing performance bottlenecks in state-of-the-art model checkers.

香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

TACAS

QSOLE: Automatic QBF Equivalence Checking

Peter Pfeiffer[1,2] Mark Peyrer[2] Daniel Große[1] Martina Seidl[2]

[1]Institute for Complex Systems, Johannes Kepler University Linz,
[2]Institute for Symbolic Artificial Intelligence, Johannes Kepler University Linz
peter.pfeiffer@jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Introduction

It is often unclear whether two QBFs that encode the same problem in different ways yield the same solutions. We present QSOLE, the first fully automatic checker for solution-based QBF equivalence. QSOLE decomposes equivalence checks into smaller entailment computations [1] and allows for explicit exclusion of variables from equivalence checks enabling comparison of formulas using different local auxiliary variables.

zenodo.org/records/18590551

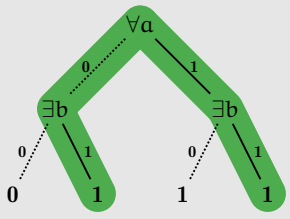
Quantified Boolean Formulas (QBF)

QBF is a 2-player game:

- Players $\forall$ and $\exists$ assign values to their variables (in order).
- The $\exists$-player has to *satisfy* the subformula.
- The $\forall$-player has to make the subformula *unsatisfiable*.
- Solutions ($\mathbb{S}$) are winning strategies.
  - > Models ($\mathbb{S}_\exists$) are winning strategies for the $\exists$-player.
  - > Counter-Model ($\mathbb{S}_\forall$) are winning strategies for the $\forall$-player.

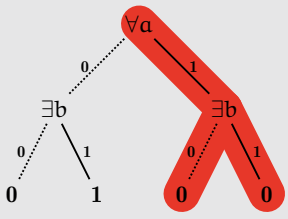
Example: Model

$\forall a \exists b : (a \vee b)$



Example: Counter-Model

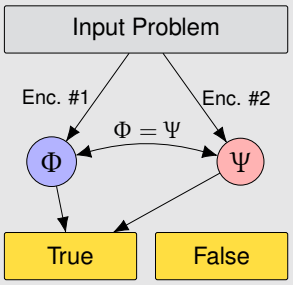
$\forall a \exists b : (a \vee b) \wedge \neg a$



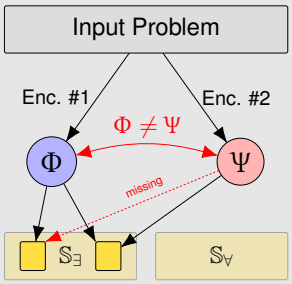
Motivation

Classic QBF equivalence [2] only compares formulas based on evaluation results. Solution-based QBF equivalence [1] ensures that they have the same solutions. It can be used to compare and debug QBF encodings. OSOLE can ensure that solutions match or find counter-examples in case they do not.

Example: Classic QBF Equivalence



Example: Solution-based Equivalence



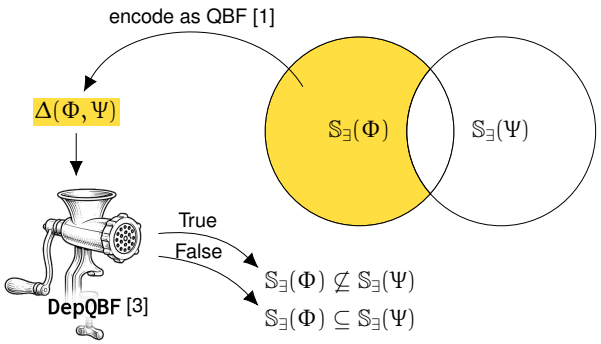
References

[1] P. Pfeiffer, D. Große, M. Seidl: "Refined Notions of QBF Equivalences" (JELIA25)
[2] H. Kleine Büning, X. Zhao: "Equivalence Models for Quantified Boolean Formulas." (SAT04)
[3] F. Lonsing, U. Egly: "Depqbf 6.0: A Search-Based QBF Solver Beyond Traditional QCDCL" (CADE17)

Skolem Entailment Checking in QSOLE

Definition: Skolem Entailment

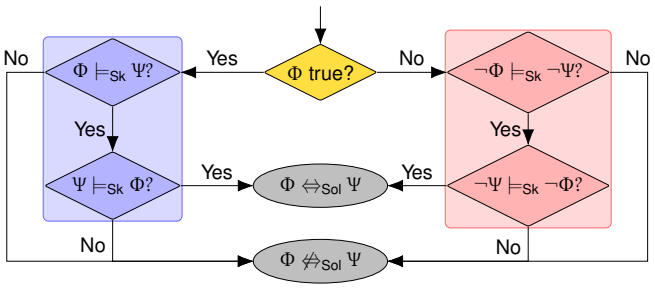
$\Phi \models_{Sk} \Psi$ if and only if $\mathbb{S}_\exists(\Phi) \subseteq \mathbb{S}_\exists(\Psi)$



Solution Equivalence Checking in QSOLE

Definition: Solution Equivalence

$\Phi \Leftrightarrow_{Sol} \Psi$ if and only if $\mathbb{S}(\Phi) = \mathbb{S}(\Psi)$



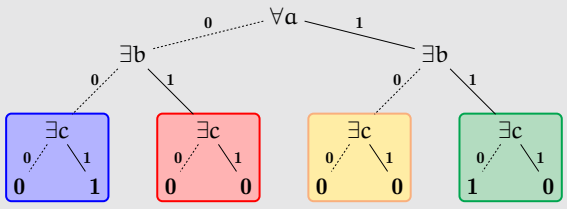
Auxiliary Variables

QSOLE can exclude inner parts of the prefix from comparisons.

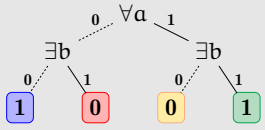
Example: Auxiliary Variables

$\forall a \exists b \exists c : (a \vee \neg b) \wedge (\neg a \vee b) \wedge (a \vee c)$

By default, this formula is treated as:



Comparison on the prefix $\forall a \exists b$ instead treats this formula as:



TACAS 2026
Apr 13-16

Error-Tolerant Quantum State Discrimination: Optimization and Quantum Circuit Synthesis

Chien-Kai Ma, Bo-Hung Chen, Tian-Fu Chen, Dah-Wei Chiou, Jie-Hong Roland Jiang

National Taiwan University

Paper (Arxiv) | Code (Zenodo) | NTU ALCom Lab

Problem

- Unambiguous QSD will not work under noise.
- We want a set of quantum circuits for QSD for further study.

Optimize for POVM with CVXPY

QSD Problem

CrossQSD

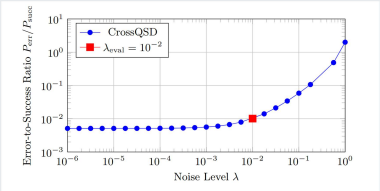


Fig. 1. CrossQSD: Error-to-success ratio $P_{\text{err}}/P_{\text{succ}}$ vs. noise level λ. The predefined states are three coherent states truncated to three qubits; $\alpha_i = \beta_i = 0.01$; $\lambda_{\text{eval}} = 0.01$.

FitQSD

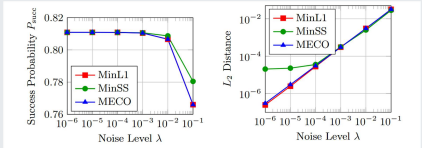


Fig. 2. FitQSD: Success probability P_{succ} (left) and L_2 distance (right) vs. noise level λ. The predefined states are given by Eq. (9). The MOSEK solver precision is set to 10^{-9}.

Hybrid Objective

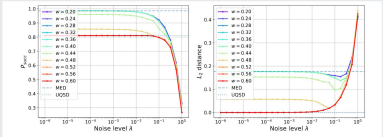


Fig. 3. Hybrid-objective QSD with $\ell = 1$ and the predefined states in Eq. (9): Success probability P_{succ} (left) and L_2 distance (right) vs. noise level λ.

POVM

Circuit synthesis and optimization

1. Translate to rank-1 POVMs with SVD
2. Remove rank-1 ops. with small singular values
3. Generate circuits from **isometries**
4. Further reduction by **circuit approx.**

Ex: Discriminate coherent states with fewer qubits (7 → 4)

Circuit specs	Without Approximation	With Approximation
Ancilla qubits	2	1
Single-qubit gates	150	34
Two-qubit gates	68	15
Circuit depth	127	25
POVM total rank	12	6

#Qubit	State Fidelity	Original		Resynthesis			Approx_deg = 1.00		
		Depth	#2Q	Depth	#2Q	Process Fidelity	Depth	#2Q	Process Fidelity
2	0.97803	84	41	45	20	1.00000	29	14	0.99999
3	0.99998	430	218	225	100	0.99999	83	61	0.99992
4	0.99999	2034	1025	1009	444	0.99999	253	252	0.99989
5	0.99999	8916	4474	4273	1868	0.99999	817	1020	0.99990
6	0.99999	37220	18634	17585	7660	0.99999	2729	4091	0.99984

QCircuit

Conclusion

- We offer alternative solutions to noise-aware QSD problems.
- We can not only synthesize a quantum circuit for QSD but also reduce its size with a modest trade-off.

32nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2026), Turin, Italy

SV-COMP'26 and Test-Comp'26 Posters

15th Competition on Software Verification

Dirk Beyer and Jan Strejček

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

MUNI

Participants

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

Participant	Jury member	Affiliation
AISE	Z. Chen	NUDT, China
APROVE (KOAT+LOAT)	N. Lommen	RWTH Aachen, Germany
BRICK	L. Bu	Nanjing U., China
COOPERACE meta	V. Vojdani	U. Tartu, Estonia
CPACHECKER	M. Jankola	LMU Munich, Germany
CPV	P.-C. Chien	LMU Munich, Germany
CSEQ	O. Inverso	Gran Sasso Sc. I., Italy
DARTAGNAN	H. Ponce de León	Huawei Dresden, Germany
DASA new	F. Mächtle	U. Lübeck, Germany
DEAGLE	F. He	Tsinghua U., China
EMERGENTHETA	L. Bajczi	BME Budapest, Hungary
ESBMC-INCR	X. Li	U. Manchester, UK
ESBMC-KIND	X. Li	U. Manchester, UK
GDART	M. Mues	TU Wuppertal, Germany
GOBLINT	S. Saan	U. Tartu, Estonia
GOBLITCH new	K. Holter	U. Tartu, Estonia
HORNIX	M. Blicha	U. Lugano, Switzerland
IEKKE new	P. Di Biase	Unimol, Italy
JAYHORN ∅	H. Mousavi	U. Tehran and TIAS, Iran
JBMC	P. Schrammel	Diffblue, UK
JLISA new	G. Zanatta	U. Venice, Italy
KORN	G. Ernst	LMU Munich, Germany
MLB	L. Bu	Nanjing U., China
MOPSA	R. Monat	Inria & U. Lille, France
MUVAL new	H. Unno	Tohoku U., Japan
NACPA meta	H. Wachowitz	LMU Munich, Germany
OGCHECKER new	Z. Yang	Xidian U., China
PROTON	R. Metta	TCS, India
PYSVLIB-CHC new	G. Ernst	LMU Munich, Germany
RACERF	T. Dacík	BUT, Czechia
REFUNCTION new	N. Moussaoui Remil	Inria and ÉNS, France
RE3VER new	A. Štěpková	Masaryk U., Czechia
SEAL new	T. Dacík	BUT, Czechia
SVF-SVC	M. Richards	U. New South Wales, AU
SVLIBCHECKER new	M. Lingsch-Rosenfeld	LMU Munich, Germany
SV-SANITIZERS	S. Saan	U. Tartu, Estonia
SWAT	N. Loose	U. Luebeck, Germany
SYMBIOTIC	M. Jonáš	Masaryk U., Czechia
THETA	C. Telbisz	BME Budapest, Hungary
THORN	L. Bajczi	BME Budapest, Hungary
UAUTOMIZER	M. Heizmann	U. Freiburg, Germany
UGEMCUTTER	D. Klumpp	U. Freiburg, Germany
UKOJAK	M. Bentele	U. Freiburg, Germany
UPARALIZER new	M. Barth	LMU Munich, Germany
UTAIPAN	D. Dietsch	U. Freiburg, Germany

Cummulative Score

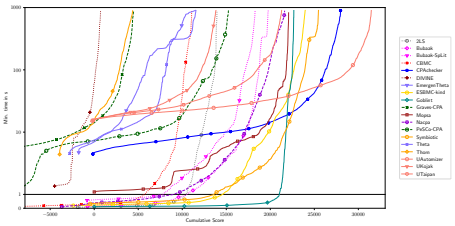


Figure 1: Quantile plot for the category *C.Overall*.

Reference

Report

D. Beyer and J. Strejček. Evaluating software verifiers for C, Java, and SV-LIB (report on SV-COMP 2026). In *Proc. TACAS (2)*, LNCS 16506. Springer, 2026

Competition Report

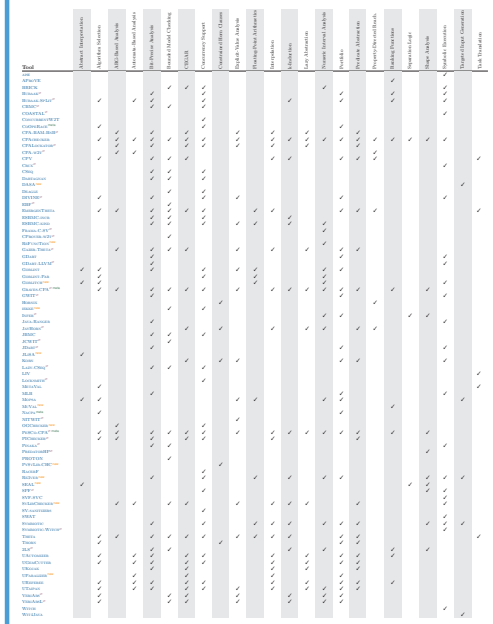
Web Site

Acknowledgment

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

Features

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



Sponsors

HUAWEI

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Ranking

Table 6: Overview of the top-three verifiers for each category; measurements for CPU time rounded to two significant digits.

Rank	Verifier	Score	CPU Time (in h)	Solved Tasks	Unconf. Tasks	False Alarms	Wrong Proofs
C.ReachSafety (14375 tasks, max. score 23331)							
1	CPAchecker	12866	150	8053	365	2	0
2	ESBMC-KIND	11037	57	7273	2456	0	0
3	SYMBIOTIC	10788	110	4080	207	1	0
C.MemSafety (4145 tasks, max. score 6329)							
1	Symbiotic	4738	3.9	3834	11	0	2
2	CPACHECKER	4646	19	3814	5	1	0
3	UAUTOMIZER	3957	35	2226	80	0	0
C.Concurrency (3124 tasks, max. score 5656)							
1	Deagle	4630	3.1	2513	19	1	3
2	DARTAGNAN	3516	19	2052	43	3	1
3	IEKKE new	3428	2.3	2473	216	44	22
C.NoOverflows (8218 tasks, max. score 13281)							
1	UParalizer new	11049	88	6700	22	0	0
2	UAUTOMIZER	10831	67	6669	59	1	0
3	UTAIPAN	10645	75	6572	56	0	0
C.Termination (2146 tasks, max. score 3734)							
1	PROTON	3368	22	1780	162	1	0
2	UAUTOMIZER	2616	17	1558	5	0	0
3	APROVE	2077	31	1011	42	0	0
C.SoftwareSystems (4394 tasks, max. score 7140)							
1	Mopsa	2746	11	1920	28	0	0
2	SYMBIOTIC	2614	6.2	1165	94	1	0
3	CPACHECKER	1496	28	1857	257	0	0
C.FalseOverall (36402 tasks, max. score 12195)							
1	Symbiotic	7833	42	7378	52	2	
2	CPACHECKER	7389	100	7668	454	4	
3	UAUTOMIZER	5835	53	4864	569	7	
C.TrueOverall (36402 tasks, max. score 48413)							
1	UAutomizer	25637	200	11 686	167		3
2	GOBLINT	22673	24	10 364	352		0
3	MOPSA	20637	26	9766	26		0
C.Overall (36402 tasks, max. score 60609)							
1	UAutomizer	31472	260	16 550	736	7	3
2	CPACHECKER	27979	240	21 632	930	4	0
3	SYMBIOTIC	25471	160	16 158	347	2	3
Java.Overall (1731 tasks, max. score 2821)							
1	JBMC	1561	1.5	1033	315	1	13
2	GDART	1470	4.3	920	107	1	1
3	JLISA new	1311	2.0	617	1	0	0

Frameworks

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

Results

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

Verifier	C.ReachSafety 14375 tasks max. score 23331	C.MemSafety 4145 tasks max. score 6329	C.Concurrency 3124 tasks max. score 5656	C.NoOverflows 8218 tasks max. score 13281	C.Termination 2146 tasks max. score 3734	C.SoftwareSystems 4394 tasks max. score 7140	C.FalseOverall 36402 tasks max. score 12195	C.TrueOverall 36402 tasks max. score 48413	C.Overall 36402 tasks max. score 60609	C.Huawei-Con.-Chall. demo 60 tasks max. score 123
aise										
APROVE					2077					
BRICK										
COOPERACE meta										
CPAchecker	12866	4646	2146	8474	1152	1496	7389	20589	27979	3
CPV	10626				1358					
CSeq			-12677							-48
Dartagnan			3516							71
Deagle			4630							-11
EmergentTheta	4414	642	2636	1821	796	-18	1091	10400	11491	0
ESBMC-incr			2435							-70
ESBMC-kind	11037	3309	2410	9109	1074	14	5232	18732	23964	-70
Goblint	4664	2840	2555	8658	1614	457		22673	22673	34
Goblitch new	6552					611				
Hornix										
iekke new			3428							-12
Korn										
Mopsa	4501	3165	0	9906	1571	2746	1442	20637	22079	0
MuVal new					356					
Nacpa meta	5959	3940	1578	7869	860	1477	4089	17538	21627	1
OGChecker new			131							0
PROTON					3368					
RacerF										
ReFunCTion new					294					
ReSver new	2583									
SEAL new										
SV-sanitizers		878		837						
SVF-SVC	-9866	0				-346				
Symbiotic	10788	4738	42	8476	1427	2614	7833	17638	25471	0
Theta	5252	695	2157	2586	796	-18	-329	11885	11556	0
Thorn	738	12	1247	-174	583	69	187	4178	4365	0
UAutomizer	8243	3957	2986	10831	2616	730	5835	25637	31472	0
UGemCutter			3189							1
UKojak	6012	2982	0	8821	0	290	3468	10347	13815	0
UParalizer new	8279			11049						
UTaipan	7013	3517	2453	10645	0	404	4243	17044	21287	0

Score Schema

Table 5: Scoring schema for SV-COMP 2026 (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)

8th Competition on Software Testing

Dirk Beyer

SoSy-Lab Software Systems

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Features

Table 2: Technologies and features that the test generators used

Tool	Bit-Precise Analysis	Bounded Model Checking	CEGAR	Concurrency Support	Explicit-Value Analysis	Floating-Point Arithmetics	Guidance by Coverage Measures	Interpolation	k-Induction	Portfolio	Predicate Abstraction	Random Execution	Symbolic Execution	Targeted Input Generation
CoVeriTest	✓		✓		✓	✓				✓	✓	✓		
ESBMC-incr	✓	✓		✓					✓					
ESBMC-kind	✓	✓		✓	✓	✓			✓					
FDSE						✓	✓					✓	✓	
Fizzer	✓													
FuSeBMC		✓				✓	✓			✓				✓
FuSeBMC-AI∅		✓				✓	✓			✓				✓
HybridTiger∅			✓		✓	✓					✓			
KLEE∅						✓							✓	✓
KLEEF∅	✓					✓	✓						✓	✓
Owi∅	✓					✓						✓	✓	✓
PRTest						✓						✓		
Rizzer∅	✓												✓	
Sikraken													✓	
Symbiotic	✓			✓		✓	✓	✓	✓	✓	✓		✓	✓
racerx		✓				✓		✓					✓	✓
racerxwp								✓						
UTestGen			✓					✓		✓				
WASP-C∅						✓						✓	✓	

Participants

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

Tester	License	Jury member	Affiliation
AFL-to-TC new	Apache	H. Wachowitz	LMU Munich, Germany
cetfuzz∅	Apache	–	–
CoVeriTest	Apache	M.-C. Jakobs	LMU Munich, Germany
ESBMC-incr	Apache	C. Wei	U. Manchester, UK
ESBMC-kind	Apache	C. Wei	U. Manchester, UK
FDSE	Apache	Z. Chen	National U. Defense Techn., China
Fizzer	Zlib	M. Trtík	Masaryk U., Brno, Czechia
FuSeBMC	MIT	K. Alshmrany	Inst. Public Admin., Saudi Arabia
FuSeBMC-AI∅	MIT	–	–
HybridTiger∅	Apache	–	–
KLEE∅	NCSA	–	–
KLEEF∅	NCSA	–	–
Owi∅	AGPL	–	–
PRTest	Apache	T. Lemberger	LMU Munich, Germany
Rizzer∅	Zlib	–	–
Sikraken	LGPL	C. Meudec	South East Technological U., Ireland
Symbiotic	MIT	M. Jonáš	Masaryk U., Brno, Czechia
TracerX∅	Apache	–	–
TracerX-WP∅	Apache	–	–
UTestGen	LGPL	M. Barth	LMU Munich, Germany
WASP-C∅	Apache	–	–
TestCoCa new	Zlib	M. Trtík	Masaryk U., Brno, Czechia
TestCov	Apache	M. Kettl	LMU Munich, Germany

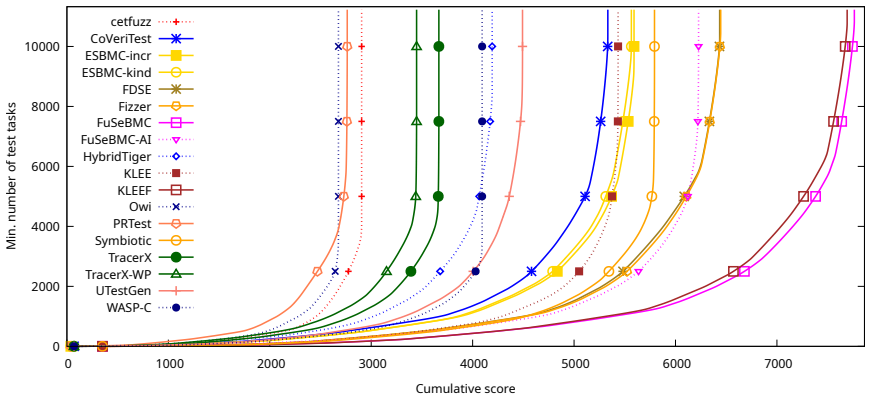
Results

Table 3: Quantitative overview over all results

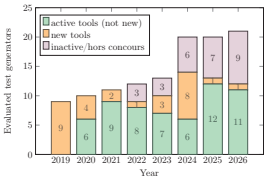
Participant	C.Cover-Error 1895 tasks max. score 1895	C.Cover-Branches 14322 tasks max. score 14322	C.Overall 16217 tasks max. score 16217
AFL-to-TC new	1082	6817	8488
cetfuzz∅	556	3584	4410
CoVeriTest	853	7084	7660
ESBMC-incr	445	1378	2685
ESBMC-kind	472	2508	3438
FDSE	1103	**8121**	**9319**
Fizzer	**1123**	7967	9317
FuSeBMC	**1486**	**8237**	**11024**
FuSeBMC-AI∅	1338	5960	9099
HybridTiger∅	727	5688	6329
KLEE∅	1213	4476	7723
KLEEF∅	1413	8282	10735
Owi∅	383	3411	3569
PRTest	331	4443	3932
Rizzer∅	997		
Sikraken		4460	
Symbiotic	**1151**	6192	8429
TracerX∅	664	4864	5594
TracerX-WP∅	543	4679	4974
UTestGen	629	5643	5888
WASP-C∅	818	3844	5674

Final Score

Figure 1: Quantile functions for category *Overall*.



Participation



Ranking

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

Rank	Tester	Score	CPU Time
C.Cover-Error (1895 tasks, max. score 1895)			
1	FuSeBMC	**1486**	290
2	Symbiotic	1151	160
3	Fizzer	1123	230
C.Cover-Branches (14322 tasks, max. score 14322)			
1	FuSeBMC	**8237**	3500
2	FDSE	8121	3400
3	Fizzer	7967	2500
C.Overall (16217 tasks, max. score 16217)			
1	FuSeBMC	**11024**	3800
2	FDSE	9319	3800
3	Fizzer	9317	2800

References

Reference

D. Beyer. Evaluating tools for automatic software testing (report on Test-Comp 2026). In *Proc. FASE*, LNCS 16504. Springer, 2026

Competition Report

Web Site

https://test-comp.sosy-lab.org/2026/

Funding

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

CPV: A Circuit-Based Program Verifier

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

[1]LMU Munich [2]National Taiwan University

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

Motivation

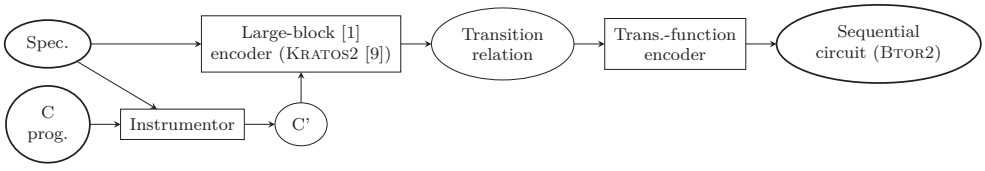
HWMCC [3]
(Input: BTOR2 circuit [11])

AVR [8], RIC3 [14], PONO [12]...

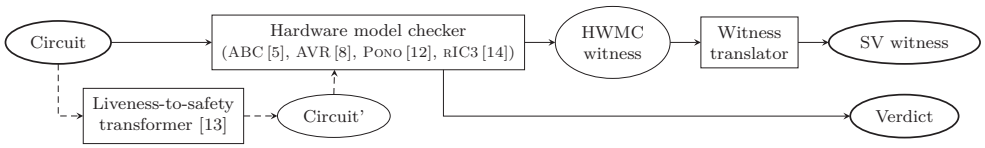
Applicable?

SV-COMP [2]
(Input: C program)

CPV's Verification Pipeline



Frontend: Program instrumentation and encoding



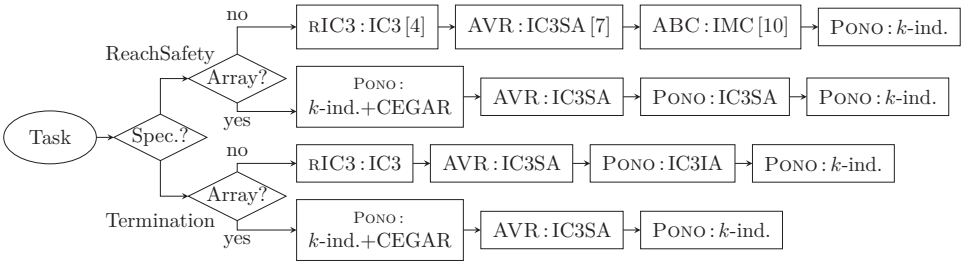
Backend: Model checking and witness translation

Try CPV!

gitlab.com/sosy-lab/software/cpv

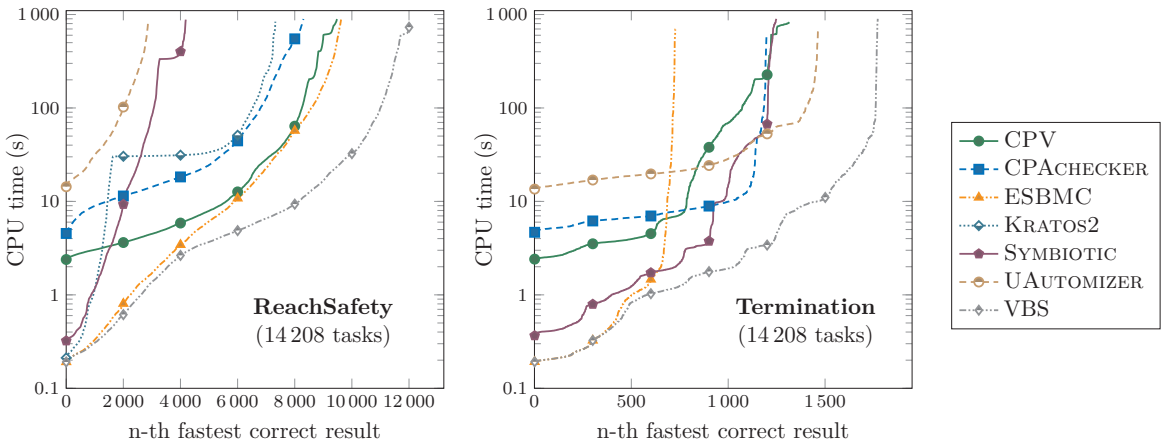
TACAS 2024 [6]

Strategy for SV-COMP 2026



Comparison with State-of-the-Art Software Verifiers

- Benchmark set: SV-COMP 2026 tasks without recursive functions
- Resource limits: 900 s CPU time and 15 GB memory
- Cactus plots show the number of solved tasks, without witness validation or subcategory weighting
- In the comparison, CPV
  - derived the **most proofs** for ReachSafety and
  - found the **most alarms** for Termination



Summary

- Sequential circuits can serve as an intermediate representation for software verification
- Offer various instrumentation, encoding and transformation options
- Leverage powerful word- and bit-level hardware model checkers as backend
- Perform competitively against state-of-the-art verifiers in SV-COMP
- Export violation witnesses in GraphML (v1) and YAML (v2) formats
- Next step: Produce correctness witnesses by translating invariants derived by hardware model checkers

References

[1] Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: Software model checking via large-block encoding. In: Proc. FMCAD. pp. 25–32 (2009)
[2] Beyer, D., Strejček, J.: Evaluating software verifiers for C, Java, and SV-LIB (report on SV-COMP 2026). In: Proc. TACAS (2). LNCS 16506 (2026)
[3] Biere, A., Froleyks, N., Preiner, M.: Hardware model checking competition 2025. In: Proc. FMCAD. p. 7 (2025)
[4] Bradley, A.R.: SAT-based model checking without unrolling. In: Proc. VMCAI. pp. 70–87. LNCS 6538 (2011)
[5] Brayton, R., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: Proc. CAV. pp. 24–40. LNCS 6174 (2010)
[6] Chien, P.C., Lee, N.Z.: CPV: A circuit-based program verifier (competition contribution). In: Proc. TACAS (3). pp. 365–370. LNCS 14572 (2024)
[7] Goel, A., Sakallah, K.: Model checking of Verilog RTL using IC3 with syntax-guided abstraction. In: Proc. NFM. pp. 166–185 (2019)
[8] Goel, A., Sakallah, K.: AVR: Abstractly verifying reachability. In: Proc. TACAS. pp. 413–422. LNCS 12078 (2020)
[9] Griggio, A., Jonáš, M.: KRATOS2: An SMT-based model checker for imperative programs. In: Proc. CAV. pp. 423–436 (2023)
[10] McMillan, K.L.: Interpolation and SAT-based model checking. In: Proc. CAV. pp. 1–13. LNCS 2725 (2003)
[11] Niemetz, A., Preiner, M., Wolf, C., Biere, A.: BTOR2, BTORMC, and BOOLECTOR 3.0. In: Proc. CAV. pp. 587–595. LNCS 10981 (2018)
[12] Perez-Lopez, A.R., Chien, P.C., Lonsing, F., Archer, S., Irfan, A., Barrett, C.: Pono 2.0: A versatile SMT-based model checker for safety and liveness (long tool paper). In: Proc. FM. LNCS (2026)
[13] Schuppan, V., Biere, A.: Liveness checking as safety checking for infinite state spaces. Electr. Notes Theor. Comput. Sci. **149**(1), 79–96 (2006)
[14] Su, Y., Yang, Q., Ci, Y., Bu, T., Huang, Z.: The rIC3 hardware model checker. In: Proc. CAV (1). pp. 185–199. LNCS 15931 (2025)

SoSy-Lab Software Systems

CPAchecker

A Tool for Configurable Program Analysis

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

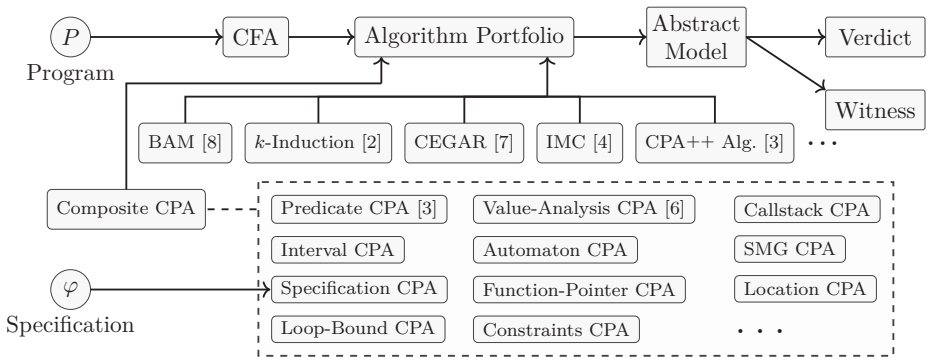
Daniel Baier, Levente Bajczi, Dirk Beyer, Marek Jankola, Matthias Kettl, Thomas Lemberger, Marian Lingsch-Rosenfeld, and Philipp Wendler

CPAchecker

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

cpachecker.sosy-lab.org

Overview



Verification

We used CPAchecker 4.2.2 as a verifier in SV-COMP 2026. More details about the respective analyses can be found in our tutorial [1].

- Strategy selection for choosing a parallel portfolio of analyses
- Support for all properties and categories of C programs
- 1st place in the category *ReachSafety*
- 2nd place in the categories *Overall*, *FalseOverall*, and *MemSafety*
- Correctly solved 21 632 tasks with only 4 wrong alarms (and 0 wrong proofs) out of 36 523 tasks
- New and improved analyses for:
  - Concurrent programs using **sequentialization** with SMG analysis
  - **SV-LIB** verification tasks

Tutorial [1]

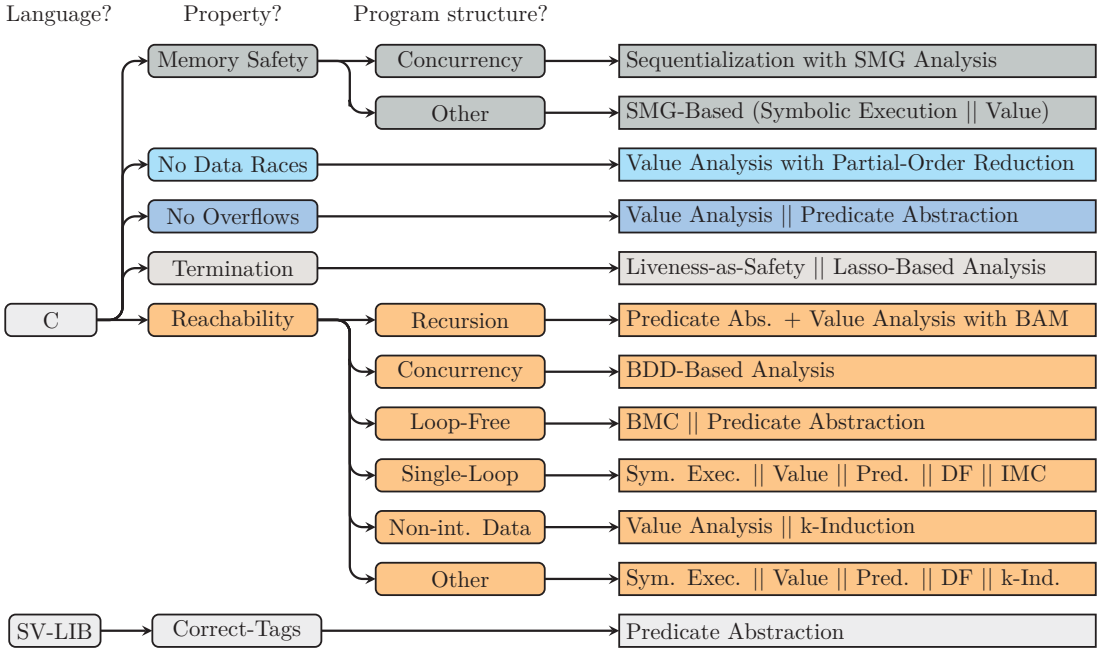
Witness Validation

We used CPAchecker 4.2.2 as a validator in SV-COMP 2026. More details can be found in our description of the competition contribution [5].

- Strategy selection for choosing an analysis
- Support for witnesses, for each property of C programs for which witness format exists
- 1st place in the categories *MemSafety*, and *SoftwareSystems* for violation witnesses 1.0, and *ReachSafety* for violation witnesses 2.0
- 2nd place in the categories *NoOverflows* and *Overall* for correctness witnesses 2.0, and *NoOverflows*, *Termination* and *Overall* for violation witnesses 2.0/2.1
- New and improved witness-validation analyses for:
  - Termination in format 2.1
  - Non-termination in format 2.1

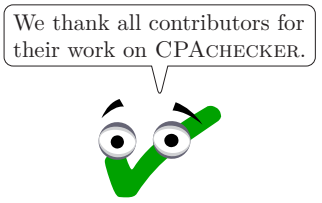
Paper [5] available here

Verification Strategy for SV-COMP 2026



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 Budapest, U Passau, U Oldenburg, U Paderborn, ISP RAS, TU Prague, TU Vienna, TU Darmstadt, and VERIMAG in Grenoble, along with several other universities and institutes.



References

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

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

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

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

[5] Beyer, D., Lingsch-Rosenfeld, M.: CPAchecker Validator 4.0 (competition contribution). In: Proc. TACAS (3). pp. 192–198. LNCS 15698, Springer (2025). https://doi.org/10.1007/978-3-031-90660-2_11

[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

Re3ver: Reverse and Verify

Adéla Štěpková, Martin Jonáš, and Jan Strejček
stepkova@mail.muni.cz
Masaryk University, Czech Republic

Motivation

- Reachability analysis checks if an error location in a program can be reached.
- The majority of standard approaches analyze programs in the forward direction: they start at the initial location.
- **Some problems are easier to solve by going backward: start at the error location and work towards the initial state.**

Key Idea: Reverse the Input Program

- Modifying each verification technique for backward traversal requires substantial effort.
- Instead, we propose to **reverse the input program**.
- By analyzing the reversed program, **the existing tools can do backward analysis of the original program without any changes to their implementation**.

Program Reversing: Concept

For an input program P, create a reversed program $rev(P)$ such that P is correct iff $rev(P)$ is correct.

- $rev(P)$ starts its execution at the error location of P and reversely follows the possible ways that may lead there in P. It tries to reach the initial state of P.
- A path in $rev(P)$ that reaches the initial state of P is a feasible error path in P.
- The initial state of P is unreachable in $rev(P)$ if and only if P is correct.

How To Reverse a Program?

Utilize the control flow graph:

- reverse the direction of control flow edges between basic blocks
- reverse the content of each basic block **statement by statement**

Reversing the control flow edges:

- basic block with multiple predecessors in $P \rightarrow$ nondeterministic branching in $rev(P)$
- conditional branch in $P \rightarrow$ assume the condition when passing the edge in $rev(P)$

Reversing a general assignment:
The reversed assignment must replicate all effects of the original assignment. When reversing, assume that the assignment holds and reassign `x` with an unknown value.

Suppose that the expression E does not reference `x`.

Original assignment:

[`x` can have any value]
`x = E`
[`x` has the value of E]

Reversed assignment:

[`x` can have any value]
`x = nondet()`
[`x` has the value of E]
`assume(x = E)`

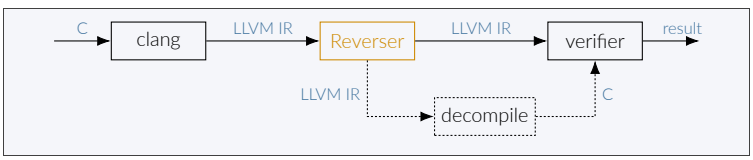
Injective operations can be inverted:

- Bijective operations: just invert the assignment
  `x += 2` → `x -= 2`
- Other operations: ban the invalid results of the operation with an `assume`:
  `x *= 2` → `assume(x % 2 == 0); x /= 2;`

Multiple error locations:
nondeterministic branch to all error locations at the start of $rev(P)$

Implementation

We target C programs, and **implement the transformation over LLVM IR** in a tool **Reverser**. In general, Reverser can be used with any verifier that accepts LLVM IR as input, or combined with a a decompiler from LLVM IR to C in principle.



Re3ver: Symbiotic + Reverser

Re3ver (pronounced *retriever*, meaning ***re**verse to **ver**ify*):

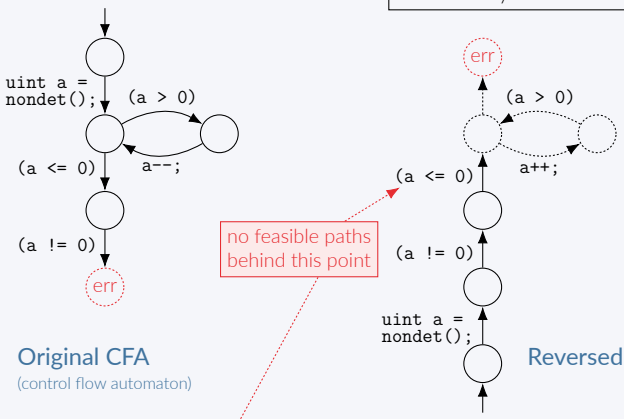
- a verification tool that employs program reversing
- Reverser integrated into the verification framework **Symbiotic**
- participation in **SV-COMP 2026**

Example: Reversing a Correct Program P

```
1 uint a = nondet();
2 while (a > 0)
3     a--;
4 assert(a == 0);
```

Original program P

Only a small part of $rev(P)$ needs to be explored to prove correctness – all feasible paths in $rev(P)$ end before the cycle starts.



Original CFA (control flow automaton)

Reversed CFA

```
1 uint a = nondet();
2 assume(a != 0);     // assume the assertion failed
3 assume(a <= 0);     // loop exited: assume negated condition
4 while (nondet()) { // guess if loop was entered now
5     a++;            // loop body
6     assume(a > 0); // loop not exited: assume condition
7 }
8 assert(false);      // start of P
```

Reversed program $rev(P)$

Evaluation

Evaluated program reversing in combination with symbolic execution from **Symbiotic**.

- benchmarks: `C.ReachSafety` category from SV-COMP 2026, `unreach-call` property
- time limit: 5 min, memory limit: 8 GB
- TO = timeout, MO = out of memory

Effect of Program Reversing on Forward Symbolic Execution

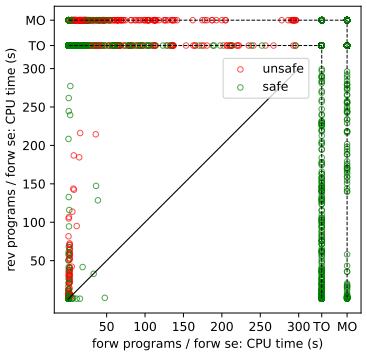
forward symbolic execution (`forw se`)

on reversed programs (`rev programs`)
→ **538** solved tasks

vs.

on forward programs (`forw programs`)
→ **1310** solved tasks

solved only on reversed programs: **370**



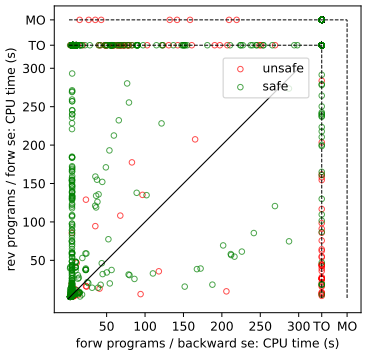
Comparison with Backward Symbolic Execution

forward symbolic execution (`forw se`)
on reversed programs (`rev programs`)
→ **475** solved tasks

vs.

backward symbolic execution (`back se`)
on forward programs (`forw programs`)
→ **375** solved tasks

solved only on reversed programs: **237**



lekkë

A SAT-Based Bounded-Round Verifier for Multi-Threaded Programs

3428 points
SV-COMP 2026 · CONCURRENCY

Competition Contribution · TACAS 2026

Paolo Di Biase[1,5] · Bernd Fischer[2] · Salvatore La Torre[3] · Peter Schrammel[4,6] · Gennaro Parlato[5]

[1] GSSI L'Aquila · [2] Stellenbosch University · [3] University of Salerno · [4] University of Sussex · [5] University of Molise · [6] Diffblue Ltd

CORE CONTRIBUTION: BOUNDED-ROUND ENCODING

lekkë is a **bounded model checking (BMC)** tool for multi-threaded C programs. Instead of classical partial-order memory constraints, it uses a **bounded-round encoding** inspired by *lazy sequentialization*: threads execute in a fixed global order, each completing a contiguous segment per round before yielding. Memory writes create a new value for each round; memory reads select the latest visible write by round & thread order.

The resulting shared-memory constraints are **linear** in both the number of memory events |E| and the round bound k, avoiding the quadratic growth of classic partial-order approaches. Because constraints are purely propositional, an off-the-shelf **SAT solver (MiniSAT)** suffices — no dedicated theory solver needed.

O(|E|²)
constraint growth
CBMC / DEAGLE

→

O(|E| · k)
constraint growth
lekkë

BOUNDED-ROUND SCHEDULE (k = 3)

Thread	Round 1	Round 2	Round 3
T1	Write Read Read	Write Read Read	Write Read Read
T2	Read Write	Read Write	Read Write
T3	Read	Read	Read

Each thread executes a contiguous SSA-segment per round. Memory writes create a new value for each round; memory reads select the latest visible write by round & thread order.

TOOL PIPELINE

01 Front-End (CBMC / Deagle)
C parsing, goto-program construction, Pthreads modeling, loop unwinding, SSA generation for thread-local control/data flow.

02 Bounded-Round Encoding [NEW]
Replaces partial-order layer. Introduces round & context-switch-point variables directly at the formula level.

03 SAT Solving (MiniSAT)
Compact propositional formula discharged by standard CDCL SAT solver. No theory-reasoning overhead.

04 Counterexamples & Witnesses
SSA-to-source variable mapping reused from CBMC/Deagle; SV-COMP violation witnesses generated.

TYPICAL INVOCATION

```
# Verify file.c with k=3 rounds, unwind bound 9
deagle_exe file.c \
  --lazy-c-seq-rounds 3 \
  --unwind 9
```

SUPPORTED PROPERTIES

- **unreach-call** — assertion violations
- **no-data-race** — conflicting unsynchronized accesses
- **valid-memsafety** — memory safety checks
- **no-overflow** — signed & unsigned arithmetic

Participates in **SV-COMP Concurrency** category only. Supports **sequential consistency**; weak memory models are left for future work.

SV-COMP 2026 RESULTS

3428
Total score — Bronze, Concurrency

308
bugs + confirmed witnesses

44 false alarms on safe benchmarks (implementation bugs).
22 unsafe benchmarks missed (due to the fixed k = 3 round limit).

STRENGTHS & LIMITATIONS

STRENGTHS

- ✓ Linear constraint growth avoids cubic blow-up of partial-order encodings
- ✓ Competitive on unsafe benchmarks; fast bug-finding with few rounds
- ✓ Pure propositional SAT — no theory-solver dependency

LIMITATIONS

- ✗ Round bound must be set manually (no iterative deepening yet)
- ✗ Sequential consistency only — no weak memory model support yet
- ✗ BMC: no completeness guarantees beyond the bounds

github.com/paolo-di-biase/Lazy-PO
Open Source · GPLv3
Zenodo archive · commit 50ddac2 · doi:10.5281/zenodo.17708300

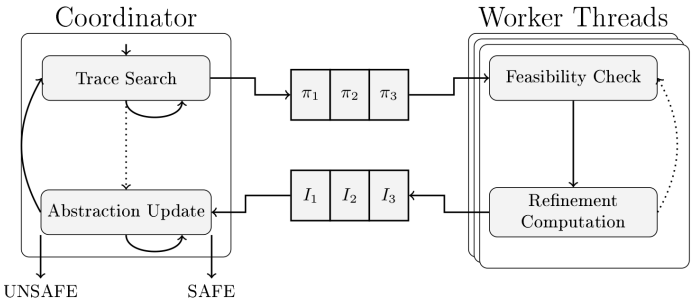
Supported by Erasmus+ International Credit Mobility 2024-1-IT02-KA171-HED-000224869, INDAM-GNCS 2024-2025, FARB 2022-25 (Università di Salerno), and AWS Amazon Research Awards. Paolo Di Biase supported by a Research Scholarship from the University of Molise.

A Parallel CEGAR loop for Trace Abstraction

Max Barth
Max.Barth@lmu.de
Supervisors: Marie-Christine Jakobs

ULTIMATE PARALIZER: A multi threaded CEGAR loop for Trace Abstraction (Software Verification)

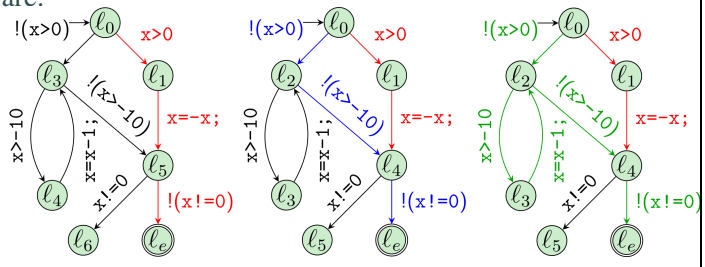
A CEGAR loop with the feasibility check, interpolation and the refinement in a separate worker thread.



- The coordinator searches and publishes fresh counterexamples (traces) π_i.
- For infeasible traces, the worker creates an Interpolant Automaton I_i that accepts π_i and all traces infeasible for similar reason.
- The abstraction automaton A is updated via $A := A \setminus I_i$.
- The loop terminates if π_i is feasible or $\mathcal{L}(A) = \emptyset$.

Diverse Trace Search

Each worker explores a different trace. We find diverse traces in the same abstraction by diverging from previous traces as soon as possible. For example, the first three counterexamples (red, blue, green) found by our search are:



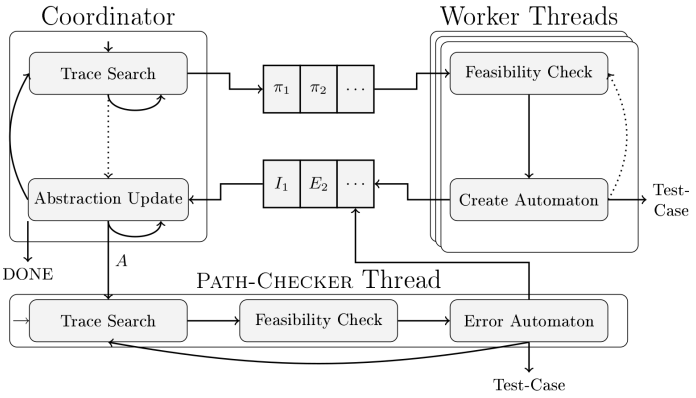
ULTIMATE PARALIZER at SV-Comp 2026

At SV-Comp 2026 ULTIMATE PARALIZER participated using 2 worker threads.

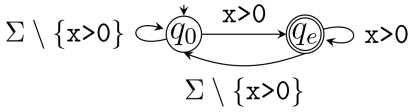
- ULTIMATE PARALIZER won the No-Overflows category.
- On reach-safety tasks ULTIMATE PARALIZER needed on average $\approx 11\%$ less walltime than ULTIMATE AUTOMIZER.

ULTIMATE TESTGEN: Transferred to Test-Case Generation

The multi-threaded framework can be transferred to test-case generation and it can be extended to cooperative verification. We show a cooperative approach with a trivial Path Checker.

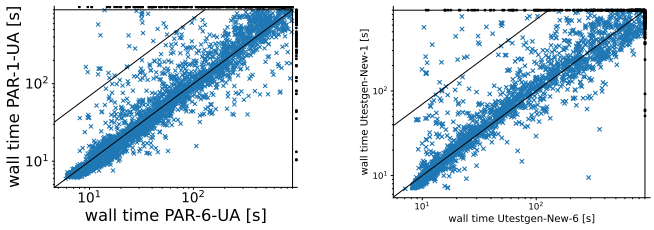


- The Path Checker uses the diverse trace search to find traces and checks their feasibility without any refinement.
- Error Automaton E_i:



Walltime Comparison

On the left ULTIMATE PARALIZER (software verification). On the right ULTIMATE TESTGEN without Path Checker (test-case generation).



- The Path Checker contributes to more coverage on tasks where interpolation is especially challenging (bit-precise reasoning, floats, etc.).