From Programs to Circuits:

Bridging Software and Hardware Model Checking with CPV and Pono

2025-12-22 Invited Talk @ NTU GIEE

Po-Chun Chien
sosy-lab.org/people/chien

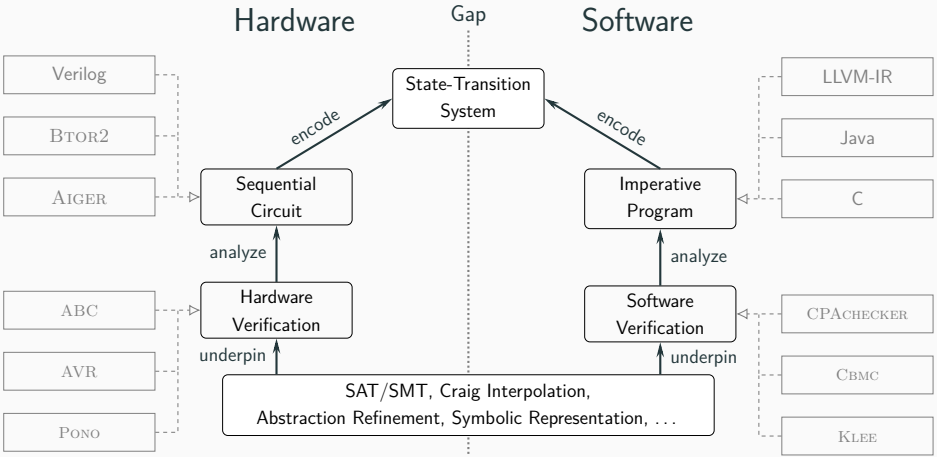
LMU Munich, Germany

About Me

- Doctoral researcher at LMU Munich, Germany (since 2021)
  - Advised by Dirk Beyer and mentored by Nian-Ze Lee
- M.S. at NTU GIEE (2020)
  - Advised by Jie-Hong Roland Jiang
- B.S. at NTU EE (2018)
- Research interests: formal methods, hardware/software verification

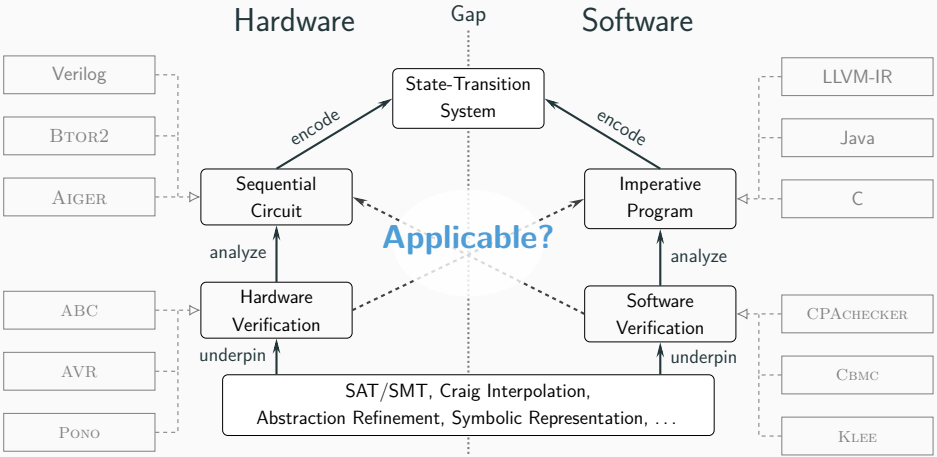
Bridging Hardware & Software Verification

DFG project 536040111 – PIs: Dirk Beyer & Nian-Ze Lee



Bridging Hardware & Software Verification

DFG project 536040111 – PIs: Dirk Beyer & Nian-Ze Lee

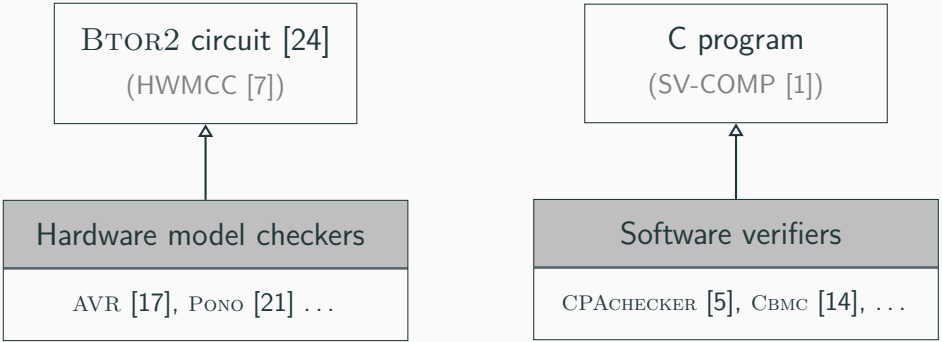


Verifying Software via Hardware Model Checking

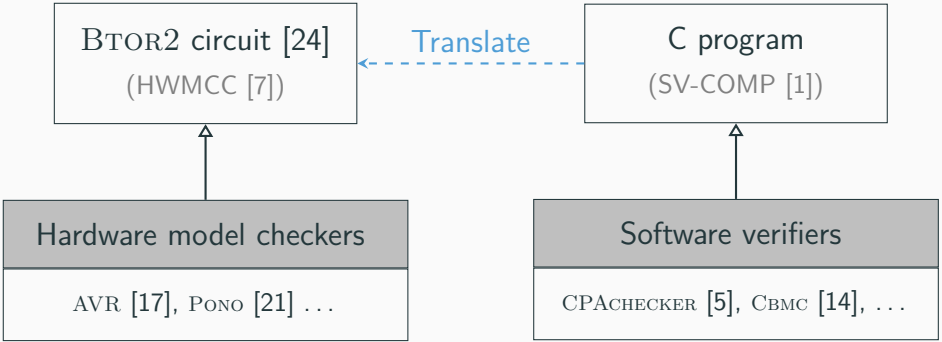
BTOR2 circuit [24]
(HWMCC [7])

C program
(SV-COMP [1])

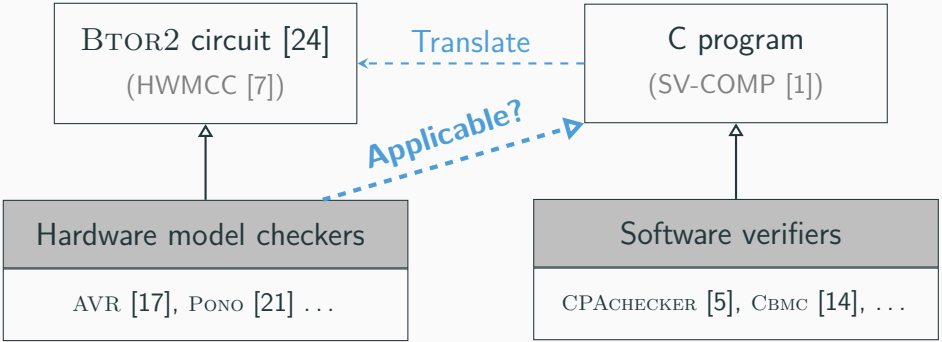
Verifying Software via Hardware Model Checking



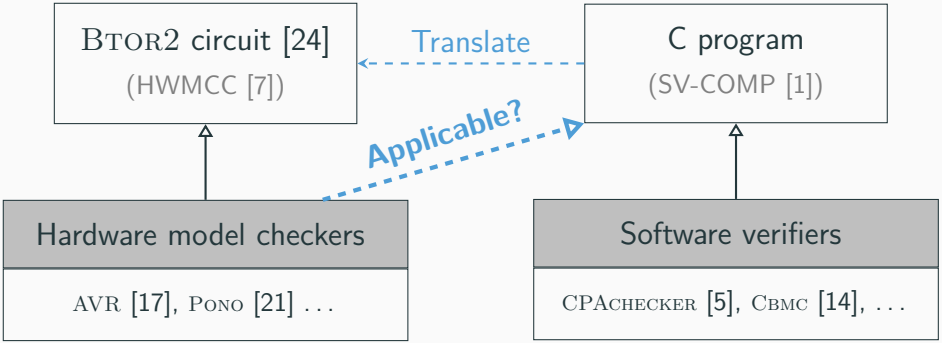
Verifying Software via Hardware Model Checking



Verifying Software via Hardware Model Checking



Verifying Software via Hardware Model Checking



Could circuits serve as IR for program verification?

Outline

- CPV: A Circuit-Based Program Verifier

  https://gitlab.com/sosy-lab/software/cpv

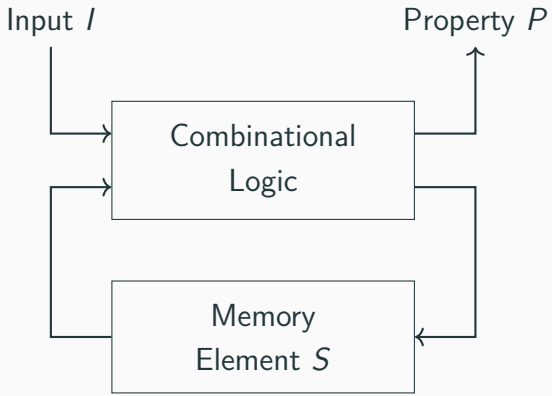
  - Translation pipeline: from C programs to sequential circuits
  - Verification backends and supported analyses
  - Experimental and competition results
- Pono: A Versatile SMT-Based Model Checker for Safety and Liveness

  https://github.com/stanford-centaur/pono

  - Architecture and solver-agnostic design
  - Recent developments in Pono 2.0
  - Comparison with other model checkers

Background

Hardware Model: Sequential Circuit



- State transition: $S' \leftarrow T_{func}(S, I)$
- Property: $P(S)$ or $P(S, I)$

Formats to Describe Circuits

Bit-level: Aiger

- Sorts: bool
- Ops: and, not

Word-level: Btor2

- Sorts: bitvec, array
- Ops: and, not, eq, add, mul, ite, read, write, ...

Example BTOR2 circuit:

```
1 sort bitvec 3
2 zero 1
3 state 1
4 init 1 3 2
5 input 1
6 add 1 3 5
7 one 1
8 sub 1 6 7
9 next 1 3 8
10 ones 1
11 sort bitvec 1
12 eq 11 3 10
13 bad 12
```

Hardware Model Checking

$$M \models \phi ?$$

Safety Property

Something bad never happens

$$\phi : \mathbf{G}\, P$$

Liveness Property

Something good eventually happens

$$\phi : \mathbf{F}\, P$$

Hardware Model Checking

$$M \vDash \phi?$$

Safety Property

Something bad never happens

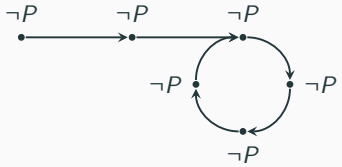
$$\phi : \mathbf{G}\, P$$

- Counterexample:

Liveness Property

Something good eventually happens

$$\phi : \mathbf{F}\, P$$



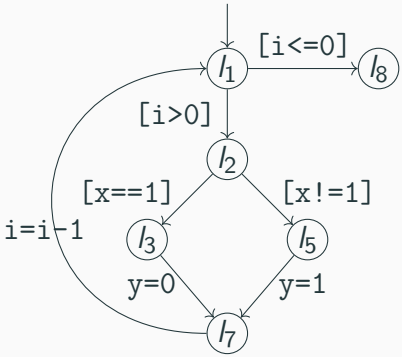
Model-Checking Algorithms

For checking safety properties:

- Bounded model checking (BMC) [9]
- k-induction (KI) [25]
- Interpolation-based model checking (IMC) [23]
- Property-directed reachability (IC3/PDR) [10, 15]
- Combined with CEGAR [13] using different abstraction techniques
  - Syntax-guided abstraction [16]
  - Predicate abstraction [19]

Software Model: Control-Flow Graph

```
1 while (i > 0) {
2   if (x == 1) {
3     y = 0;
4   } else {
5     y = 1;
6   }
7   i = i - 1;
8 }
```



Software Verification

$$M \models \phi?$$

Reachability Analysis (Safety)

Is an error location reachable?

$$\phi : \mathbf{G}\ \neg \mathit{error}$$

Termination Analysis (Liveness)

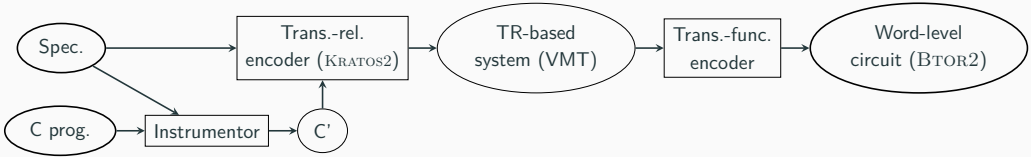
Can the program always terminate?

$$\phi : \mathbf{F}\ \mathit{end}$$

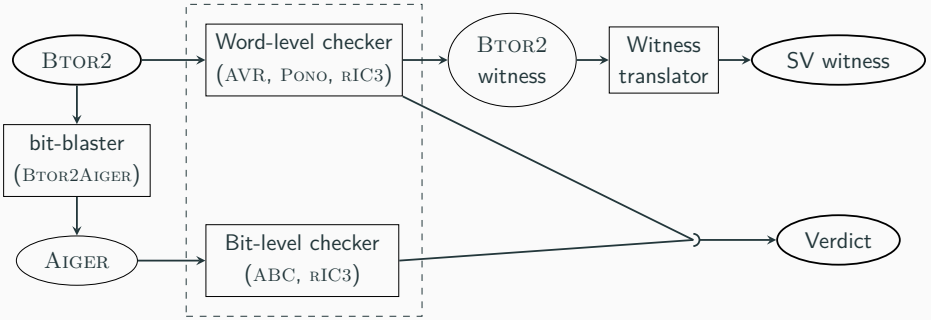
Circuit-Based Program Verification

Joint work with Dirk Beyer (LMU Munich),
Armin Biere (University of Freiburg), and
Nian-Ze Lee (National Taiwan University)

System Architecure: Frontend



System Architecure: Backend

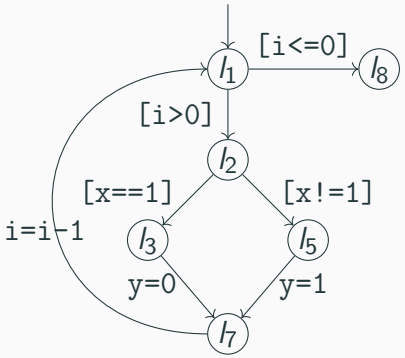


Managed by CoVeriTeam

Frontend: C Program → Transition Relation

Single-Block Encoding

C Program → Transition Relation

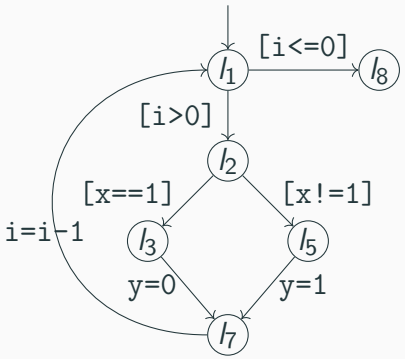


Each statement as a block:

$$TR = (pc = l_1 \wedge pc' = l_2 \wedge \mathbf{i} > \mathbf{0} \wedge i' = i \ldots)$$
$$\vee\, (pc = l_2 \wedge pc' = l_3 \wedge \mathbf{x} = \mathbf{1} \wedge i' = i \ldots)$$
$$\vee\, (pc = l_3 \wedge pc' = l_7 \wedge i' = i \wedge \mathbf{y}' = \mathbf{0} \ldots)$$
$$\vee \ldots$$

Large-Block Encoding

C Program → Transition Relation



Each loop-free section as a block:

$$TR = (pc = l_1 \wedge pc' = l_1 \wedge i > 0 \wedge i' = i - 1 \wedge y' = ite(x = 1, 0, 1) \ldots) \vee (pc = l_1 \wedge pc' = l_8 \wedge i \leq 0 \wedge i' = i \ldots) \vee \ldots$$

Encoding Details

C Program → Transition Relation

- KRATOS2 [18] is employed
- Large-block encoding [2] is used by default
- Function calls are inlined or treated as blocks
- C program uses `__VERIFIER_nondet_X()` to model nondeterministic values
  → Primary inputs in $TR(S, I, S')$

Encoding Details

C Program → Transition Relation

- KRATOS2 [18] is employed
- Large-block encoding [2] is used by default
- Function calls are inlined or treated as blocks
- C program uses `__VERIFIER_nondet_X()` to model nondeterministic values
  → Primary inputs in $TR(S, I, S')$
- CPV vs. other software verifiers:
  Monolithic transition relation vs. ARG-based exploration [3]

Frontend: Transition Relation → Btor2 Circuit

Relational Encoding

Transition Relation → Btor2 Circuit

- Given $TR(S, I, S')$ and $P(S)$

Relational Encoding

Transition Relation → Btor2 Circuit

- Given $TR(S, I, S')$ and $P(S)$
- Auxiliary variables:
  - New state var *valid*
  - For each state var $s \in S$, introduce an input var s_{in}

Relational Encoding

Transition Relation → Btor2 Circuit

- Given $TR(S, I, S')$ and $P(S)$
- Auxiliary variables:
  - New state var *valid*
  - For each state var $s \in S$, introduce an input var s_{in}
- Transition *functions* and property in BTOR2:
  - $valid' \leftarrow valid \wedge TR(S, I, S_{in})$
  - $s' \leftarrow s_{in}$ for each $s \in S$
  - $P_{Btor2} : valid \Rightarrow P(S)$

Functional Encoding

Transition Relation → Btor2 Circuit

- Identify control and data flows in a block formula:

$$pc = l_1 \wedge pc' = l_1 \wedge i > 0 \wedge i' = i - 1 \wedge y' = ite(x = 1, 0, 1) \ldots$$

Functional Encoding

Transition Relation → Btor2 Circuit

- Identify control and data flows in a block formula:

$$pc = l_1 \wedge pc' = l_1 \wedge i > 0 \wedge i' = i - 1 \wedge y' = ite(x = 1, 0, 1) \ldots$$

- Derive next-state functions directly:

$$i' \leftarrow \begin{cases} i - 1 & , if\ pc = l_1 \wedge i > 0 \\ i & , if\ pc = l_1 \wedge i \leq 0 \\ \ldots & \end{cases}$$

Functional Encoding

Transition Relation → Btor2 Circuit

- Identify control and data flows in a block formula:

$$pc = l_1 \wedge pc' = l_1 \wedge i > 0 \wedge i' = i - 1 \wedge y' = ite(x = 1, 0, 1) \ldots$$

- Derive next-state functions directly:

$$i' \leftarrow \begin{cases} i - 1 & , if\ pc = l_1 \wedge i > 0 \\ i & , if\ pc = l_1 \wedge i \leq 0 \\ \ldots & \end{cases}$$

- TR is not right-total (e.g., $pc = l_{end}$ has no successor)
  → add an auxiliary sink state l_{sink}

Relational vs. Functional Encoding

Transition Relation → Btor2 Circuit

Relational

- More general (works for any transition relation)
- Usually result in smaller circuits

Functional

- Fewer input variables
- Allow optimization when circuits are unrolled [20]

Encoding Verification Property

Reachability Analysis (Safety)

Is an error location reachable?

```
int main() {  // l_init
  ...
  if (error_cond) {
    reach_error();  // l_err
  }
  ...
  return 0;
}
```

Termination Analysis (Liveness)

Can the program always terminate?

```
// instrumented by CPV
int main() {  // l_init
  // execute the original main
  original_main();
  // add termination marker
  reach_end();  // l_end
  return 0;
}
```

Encoding Verification Property

Reachability Analysis (Safety)

Is an error location reachable?

$$\phi : \mathbf{G}\,(pc \neq l_{err})$$

Termination Analysis (Liveness)

Can the program always terminate?

$$\phi : \mathbf{F}(pc = l_{end})$$

Encoding Verification Property

Reachability Analysis (Safety)

Is an error location reachable?

$$\phi : \mathbf{G}\,(pc \neq l_{err})$$

- In BTOR2: `bad`
- Counterexample:

Termination Analysis (Liveness)

Can the program always terminate?

$$\phi : \mathbf{F}(pc = l_{end})$$

Encoding Verification Property

Reachability Analysis (Safety)

Is an error location reachable?

$$\phi : \mathbf{G}\,(pc \neq l_{err})$$

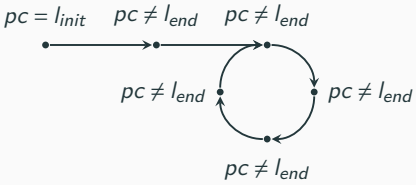
- In BTOR2: `bad`
- Counterexample:

Termination Analysis (Liveness)

Can the program always terminate?

$$\phi : \mathbf{F}(pc = l_{end})$$

`justice` (+ `constraint`)



Backend: Model Checking

Integrated Hardware Model Checkers

	Tool	Input formats	Engines
Bit-level	ABC	AIGER	BMC, IMC, PDR
	rIC3	AIGER, BTOR2	BMC, KI, IC3
Word-level	AVR	BTOR2	BMC, KI, IC3sa
	PONO	BTOR2	BMC, IMC, KI, IC3sa, IC3ia

Implementation Details

- CoVeriTeam [4] is used to coordinate backend model checkers
  - Containerized execution with resource limits
  - Assemble command lines and extract tool outputs (logs and files)
  - Easy to combine multiple tools

Implementation Details

- CoVeriTeam [4] is used to coordinate backend model checkers
  - Containerized execution with resource limits
  - Assemble command lines and extract tool outputs (logs and files)
  - Easy to combine multiple tools
- Most tools do not support `justice` in Btor2
  → CPV does liveness-to-safety transformation

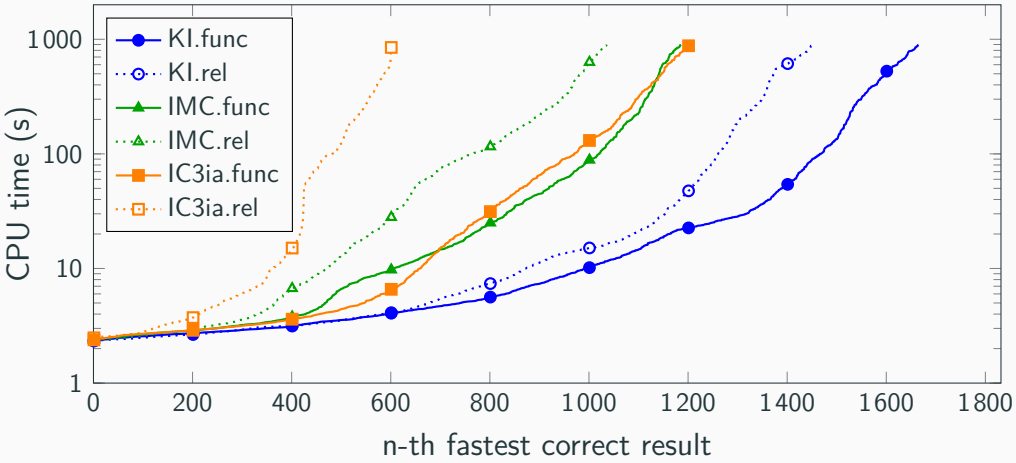
Current Limitations

- Lack support for many library functions (e.g., `<math.h>`)
- Limited support for recursion (*finite* unrolling)
- Does not handle concurrency
- Unsupported properties in SV-COMP:
  - Memory safety
  - Overflow detection

Experimental Evaluation

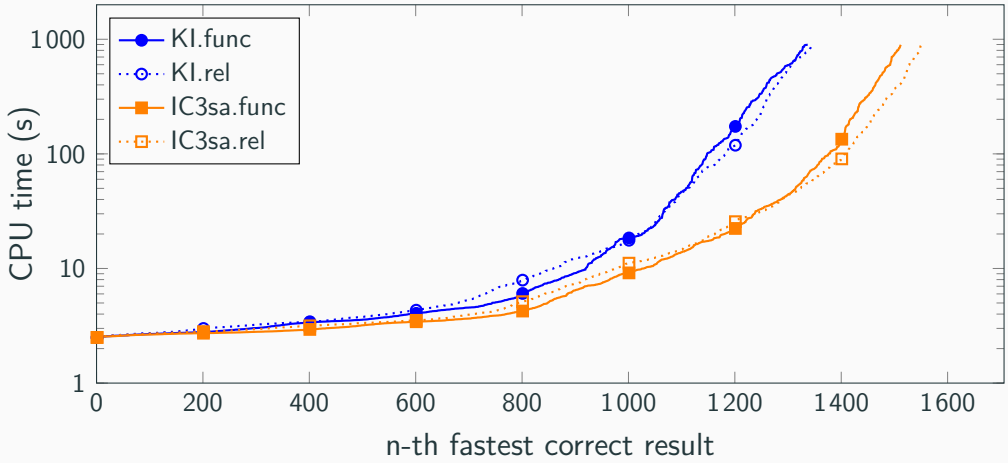
Experimental Results: Relational vs. Functional Encoding

Reachability analysis using Pono



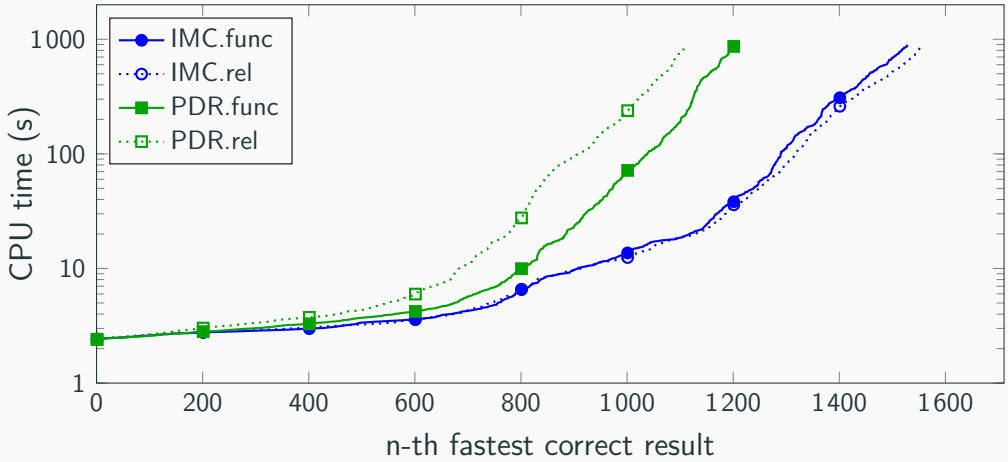
Experimental Results: Relational vs. Functional Encoding

Reachability analysis using AVR



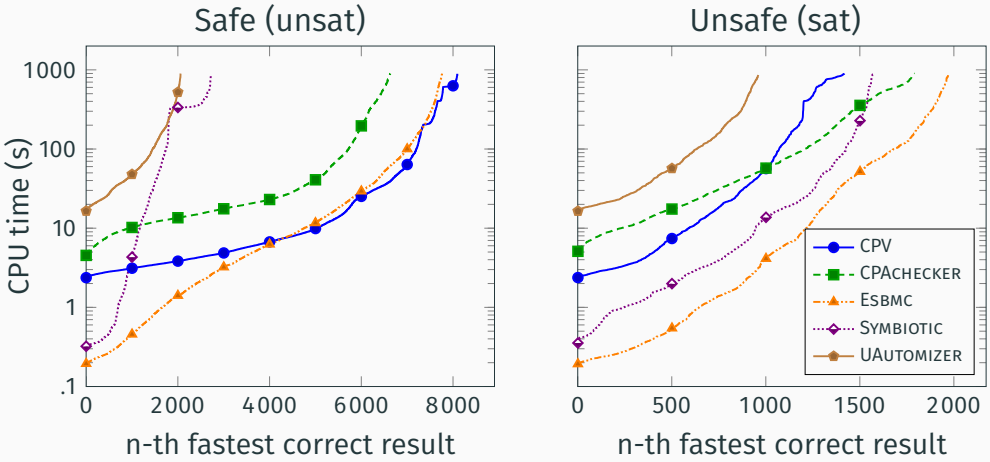
Experimental Results: Relational vs. Functional Encoding

Reachability analysis using ABC

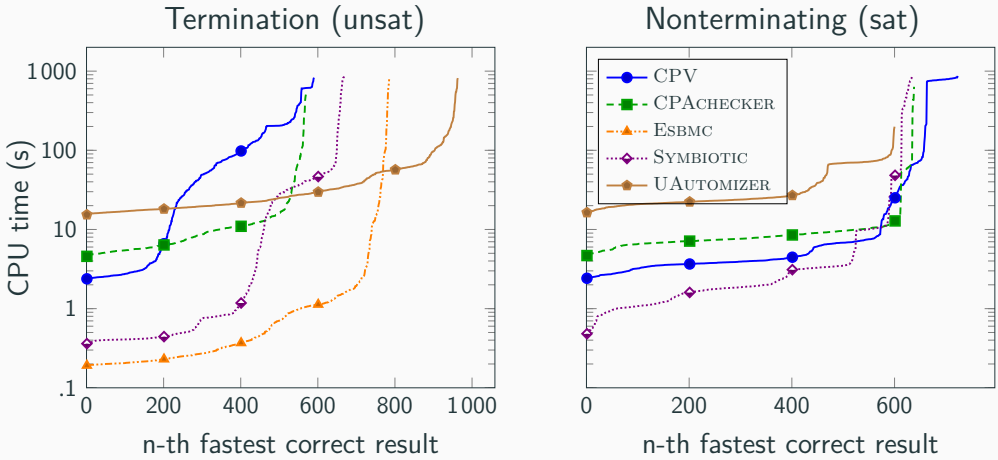


Preliminary Results in SV-COMP 2026: Reachability

Part of TACAS 2026 (April 11–16) @ Turin, Italy



Preliminary Results in SV-COMP 2026: Termination

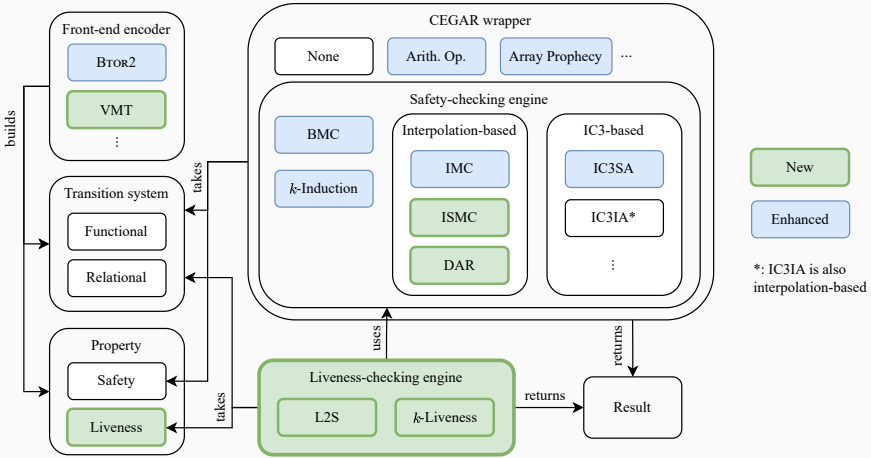


Pono:

A Versatile SMT-Based Model Checker

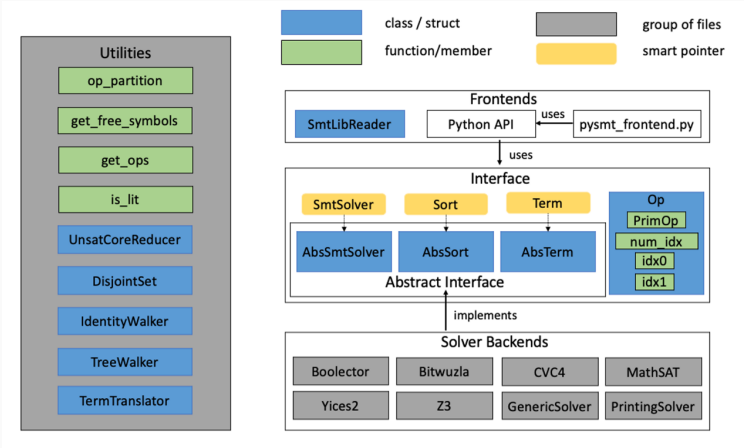
Joint work with Áron Ricardo Perez-Lopez[1], Florian Lonsing,
Samantha Archer[1], Ahmed Irfan[2], and Clark Barrett[1]
([1]Stanford University · [2]SRI International)

System Architecure



SMT Solver Integration

SmtSwtich: A Solver-Agnostic C++ API for SMT Solving

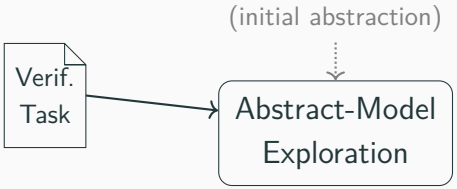


(Taken from [22])

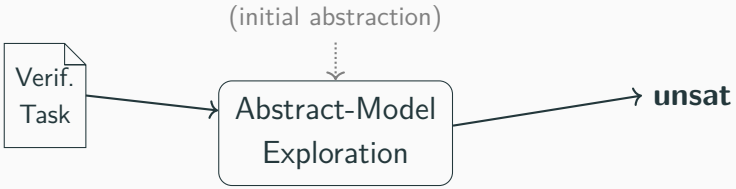
New:

- Incremental interpolation
- BITWUZLA interpolation
- CVC5 integration

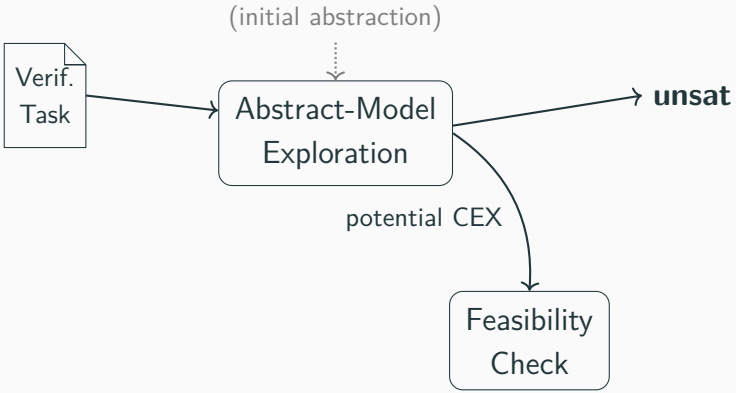
Counterexample-Guided Abstraction Refinement (CEGAR)



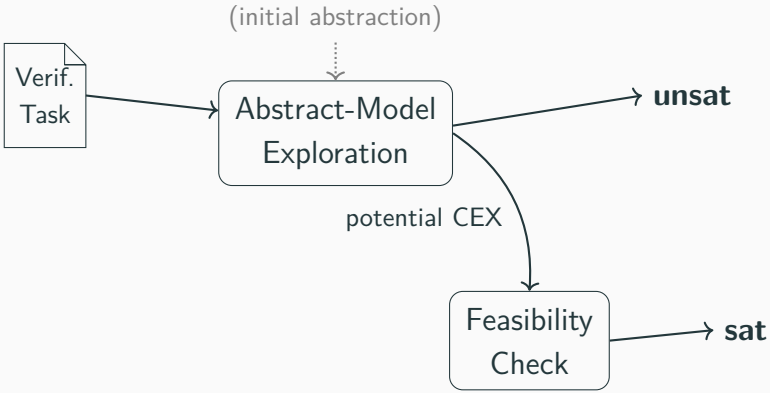
Counterexample-Guided Abstraction Refinement (CEGAR)



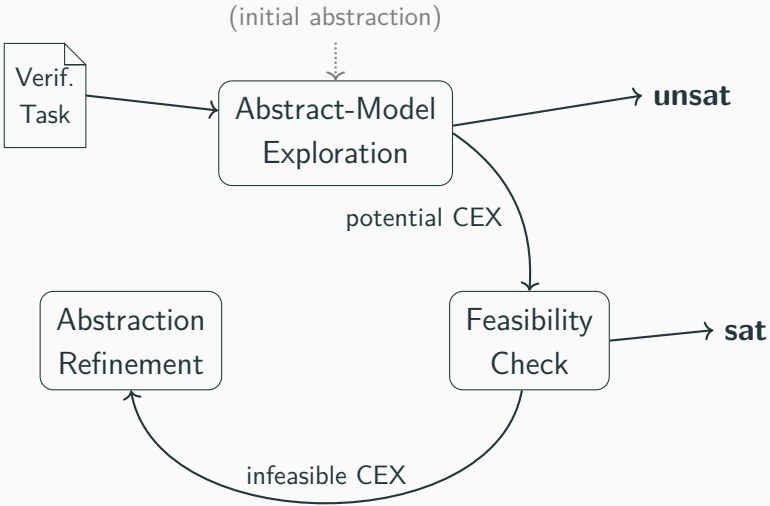
Counterexample-Guided Abstraction Refinement (CEGAR)



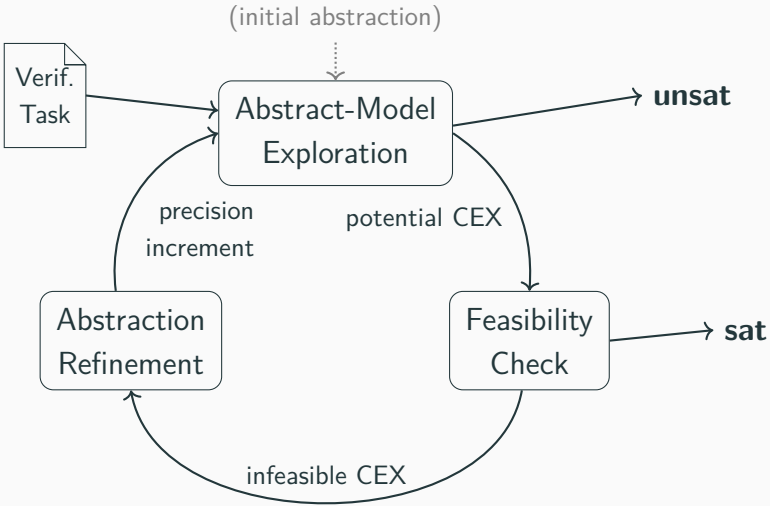
Counterexample-Guided Abstraction Refinement (CEGAR)



Counterexample-Guided Abstraction Refinement (CEGAR)



Counterexample-Guided Abstraction Refinement (CEGAR)



Arithmetic-Operation Abstraction

- SMT queries with complex arithmetic ops (e.g., $\div$, $\times$) are hard to solve
- Abstract these ops as uninterpreted functions (UFs) or free variables

Arithmetic-Operation Abstraction

- SMT queries with complex arithmetic ops (e.g., $\div$, $\times$) are hard to solve
- Abstract these ops as uninterpreted functions (UFs) or free variables
- Abstraction refinement (given a spurious CEX):
  - Check feasibility of the CEX with on concrete system
  - If unsat, identify ops to concretize from the unsat core

Arithmetic-Operation Abstraction

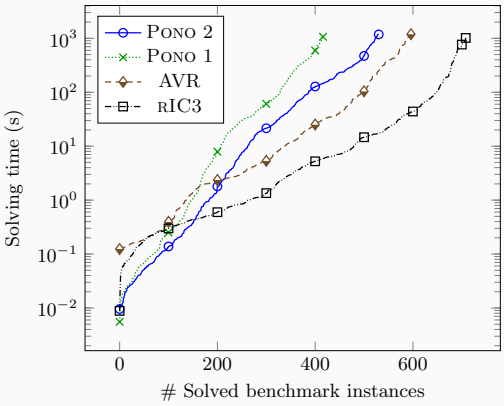
- SMT queries with complex arithmetic ops (e.g., $\div$, $\times$) are hard to solve
- Abstract these ops as uninterpreted functions (UFs) or free variables
- Abstraction refinement (given a spurious CEX):
  - Check feasibility of the CEX with on concrete system
  - If unsat, identify ops to concretize from the unsat core
- Can be combined with any safety-checking engine in Pono

  (In version 1.0, only IC3IA and IC3SA)

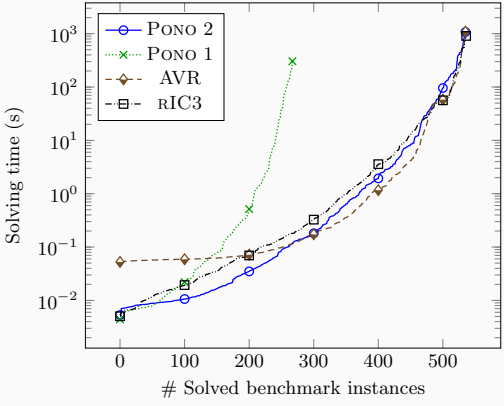
Liveness Checking

- Two approaches that reduce liveness to safety:
  - Liveness-to-Safety (L2S) [8]
  - *k*-liveness [12]
- Can be combined with any safety-checking engine in Pono
- Currently only works for finite-state systems (e.g., Btor2)
- If sat, Btor2 witness (lasso) can be exported

Evaluation: QF_BV Safety Checking

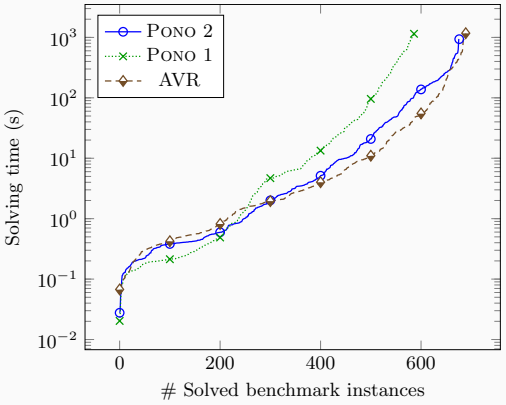


HWMCC tasks

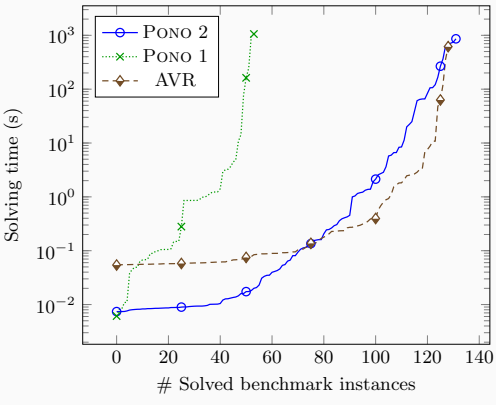


SV-COMP Reachability tasks

Evaluation: QF_ABV Safety Checking

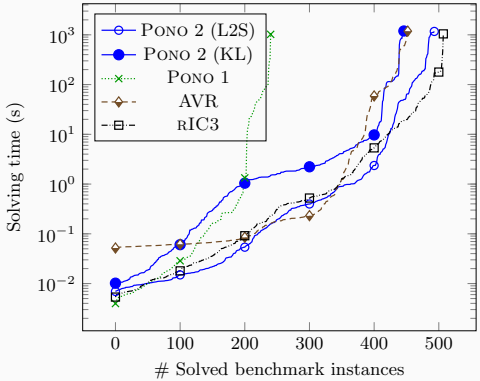


HWMCC tasks

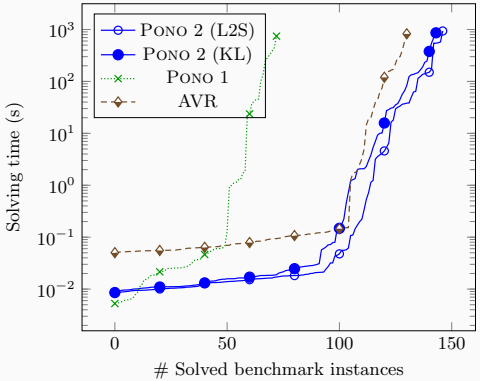


SV-COMP Reachability tasks

Evaluation: Liveness Checking



SV-COMP QF_BV Termination tasks



SV-COMP QF_ABV Termination tasks

(Pono 1, AVR, and rIC3 ran on pre-L2S-transformed tasks)

Conclusion

- CPV [11]
  - Encodes SV tasks as circuits
  - Leverages hardware model checkers as backend
- Pono [21]
  - Supports safety and liveness
  - Interfaces multiple SMT solvers
- Strong results in competitions
- Both projects are open-source

gitlab.com/
sosy-lab/software/cpv

References i

[1] Beyer, D.: State of the art in software verification and witness validation: SV-COMP 2024. In: Proc. TACAS (3). pp. 299–329. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_15

[2] Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: Software model checking via large-block encoding. In: Proc. FMCAD. pp. 25–32. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351147

[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., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. In: Proc. TACAS. pp. 561–579. LNCS 13243, Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_31

References ii

[5] Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_16

[6] Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: Requirements and solutions. Int. J. Softw. Tools Technol. Transfer **21**(1), 1–29 (2019). https://doi.org/10.1007/s10009-017-0469-y

[7] Biere, A., Froleyks, N., Preiner, M.: Hardware model checking competition 2024. In: Proc. FMCAD. pp. 7–7. TU Wien Academic Press (2024). https://doi.org/10.34727/2024/isbn.978-3-85448-065-5_6

[8] Biere, A., Artho, C., Schuppan, V.: Liveness checking as safety checking. Electr. Notes Theor. Comput. Sci. **66**(2), 160–177 (2002). https://doi.org/10.1016/S1571-0661(04)80410-9

[9] Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers **58**, 117–148 (2003). https://doi.org/10.1016/S0065-2458(03)58003-2

References iii

[10] Bradley, A.R.: SAT-based model checking without unrolling. In: Proc. VMCAI. pp. 70–87. LNCS 6538, Springer (2011). https://doi.org/10.1007/978-3-642-18275-4_7

[11] Chien, P.C., Lee, N.Z.: CPV: A circuit-based program verifier (competition contribution). In: Proc. TACAS (3). pp. 365–370. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_22

[12] Claessen, K., Sörensson, N.: A liveness checking algorithm that counts. In: Proc. FMCAD. pp. 52–59. IEEE (2012). https://ieeexplore.ieee.org/document/6462555/

[13] 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

[14] Clarke, E.M., Kröning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proc. TACAS. pp. 168–176. LNCS 2988, Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_15

References iv

[15] Eén, N., Mishchenko, A., Brayton, R.K.: Efficient implementation of property directed reachability. In: Proc. FMCAD. pp. 125–134. FMCAD Inc. (2011). https://dl.acm.org/doi/10.5555/2157654.2157675

[16] Goel, A., Sakallah, K.: Model checking of Verilog RTL using IC3 with syntax-guided abstraction. In: Proc. NFM. pp. 166–185. Springer (2019). https://doi.org/10.1007/978-3-030-20652-9_11

[17] Goel, A., Sakallah, K.: AVR: Abstractly verifying reachability. In: Proc. TACAS. pp. 413–422. LNCS 12078, Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_23

[18] Griggio, A., Jonáš, M.: KRATOS2: An SMT-based model checker for imperative programs. In: Proc. CAV. pp. 423–436. Springer (2023). https://doi.org/10.1007/978-3-031-37709-9_20

[19] Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: Proc. POPL. pp. 232–244. ACM (2004). https://doi.org/10.1145/964001.964021

[20] Jussila, T., Biere, A.: Compressing BMC encodings with QBF. Electronic Notes in Theoretical Computer Science **174**(3), 45–56 (2007). https://doi.org/10.1016/j.entcs.2006.12.022

References v

[21] Mann, M., Irfan, A., Lonsing, F., Yang, Y., Zhang, H., Brown, K., Gupta, A., Barrett, C.W.: Pono: A flexible and extensible SMT-based model checker. In: Proc. CAV. pp. 461–474. LNCS 12760, Springer (2021). https://doi.org/10.1007/978-3-030-81688-9_22

[22] Mann, M., Wilson, A., Zohar, Y., Stuntz, L., Irfan, A., Brown, K., Donovick, C., Guman, A., Tinelli, C., Barrett, C.W.: Smt-switch: A solver-agnostic C++ API for SMT solving. In: Proc. SAT. pp. 377–386. LNCS 12831, Springer (2021). https://doi.org/10.1007/978-3-030-80223-3_26

[23] McMillan, K.L.: Interpolation and SAT-based model checking. In: Proc. CAV. pp. 1–13. LNCS 2725, Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_1

[24] Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2, BtorMC, and Boolector 3.0. In: Proc. CAV. pp. 587–595. LNCS 10981, Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_32

[25] Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: Proc. FMCAD, pp. 127–144. LNCS 1954, Springer (2000). https://doi.org/10.1007/3-540-40922-X_8