From Programs to Circuits:

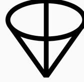
Bridging Software and Hardware Model Checking with CPV

Po-Chun Chien[1],

Dirk Beyer[1], Armin Biere[2], and Nian-Ze Lee[3,1]

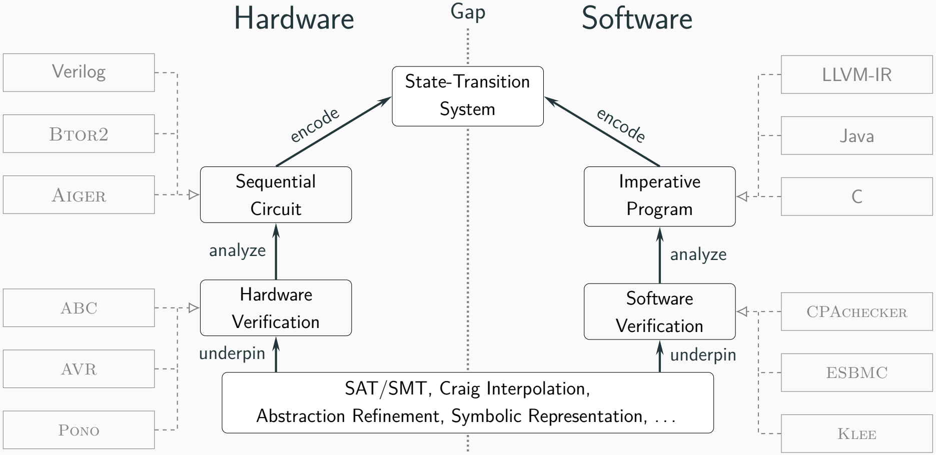
[1]LMU Munich · [2]University of Freiburg · [3]National Taiwan University

COOP 2026 @ Turin, Italy



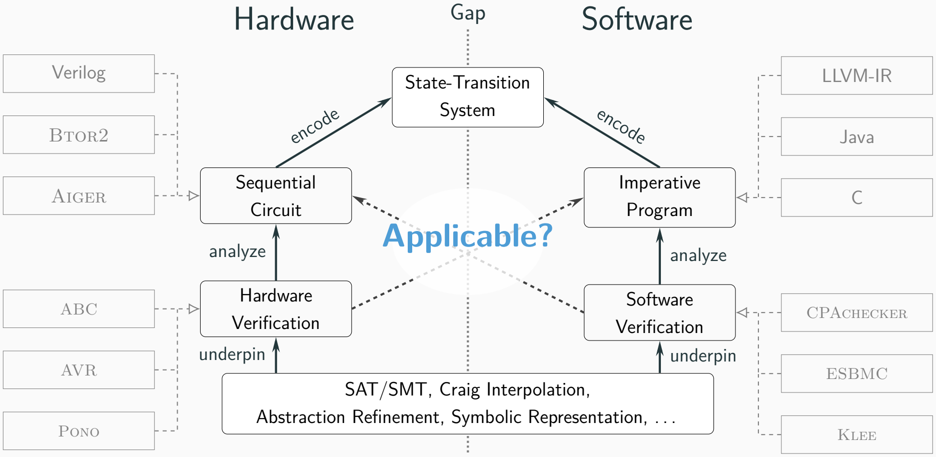
Bridging Hardware and Software Verification

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



Bridging Hardware and Software Verification

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



Verifying Software via Hardware Model Checking

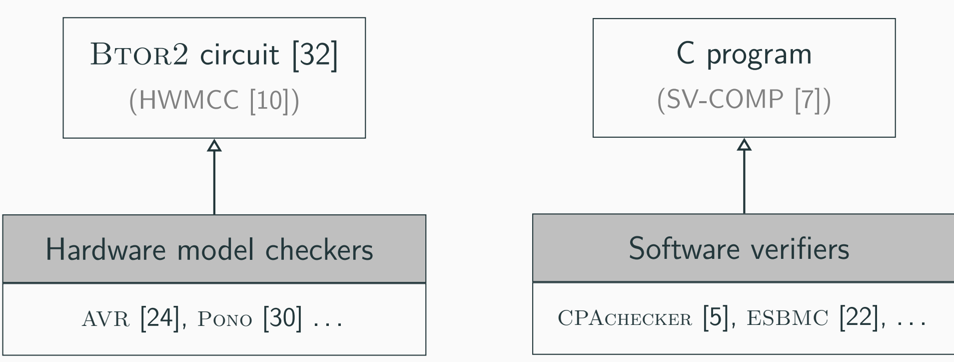
Btor2 circuit [32]

(HWMCC [10])

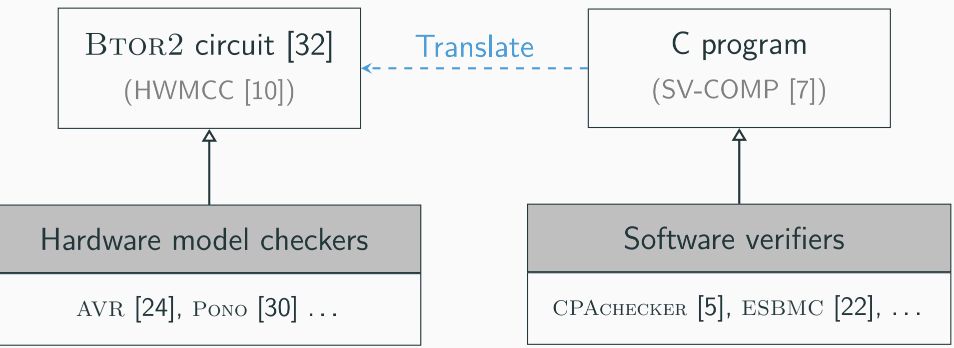
C program

(SV-COMP [7])

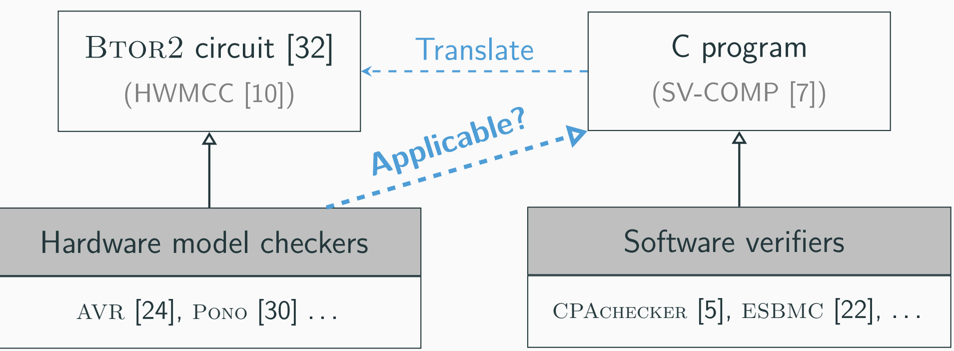
Verifying Software via Hardware Model Checking



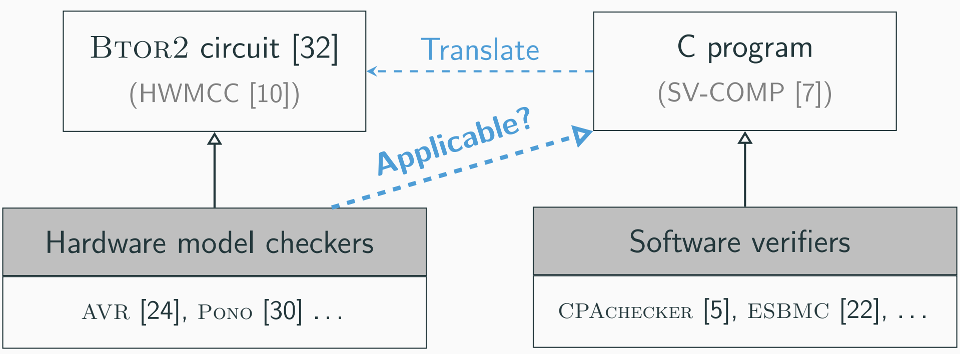
Verifying Software via Hardware Model Checking



Verifying Software via Hardware Model Checking



Verifying Software via Hardware Model Checking



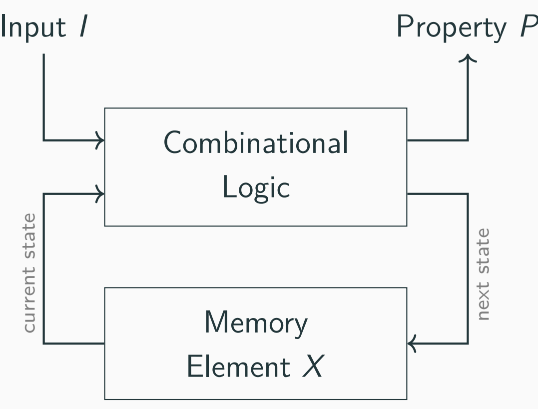
Could circuits serve as an IR for program verification?

Outline

- Background
- Verification pipeline of CPV
  - Encoding C verification tasks as sequential circuits
  - Hardware model checking
  - Witness translation
- Evaluation: Comparing to SOTA software verifiers

Background

Hardware Model: Sequential Circuit



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

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

$$M \models \phi?$$

Safety Property

Something bad never happens

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

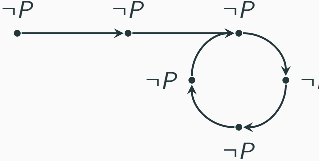
- Counterexample:

$P \to P \to P \to P \to \neg P$

Liveness Property

Something good eventually happens

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



Software Model: Control-Flow Automaton

```
1  extern unsigned int nondet();
2  int main() {
3    unsigned int a = 1;
4    unsigned int b = nondet();
5    while (a <= b) {
6      a = nondet();
7      b = (b + 1) % 3;
8      if (a + b == 5) {
9        ERROR: return 1;
10     }
11   }
12   return 0;
13 }
```

Software Verification

$$M \models \phi?$$

Reachability Analysis (Safety)

Is an error location reachable?

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

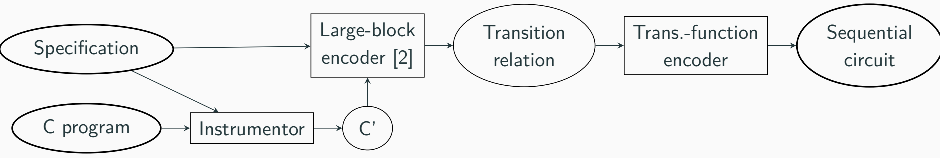
Termination Analysis (Liveness)

Can the program always terminate?

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

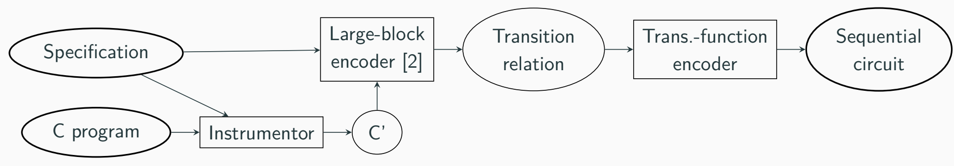
Verification Pipeline

Verification Pipeline of CPV

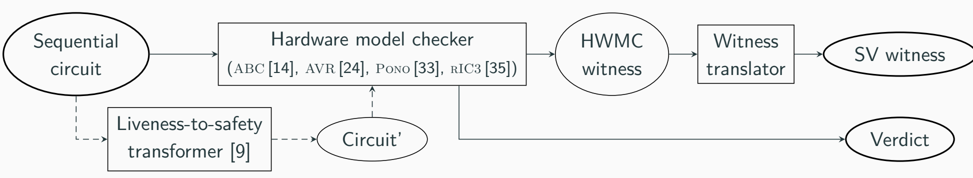


Frontend: Program instrumentation and encoding

Verification Pipeline of CPV



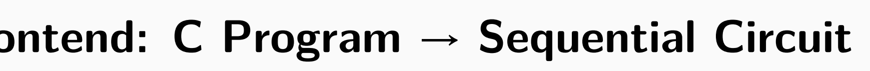
Frontend: Program instrumentation and encoding



Backend: Model checking and witness translation

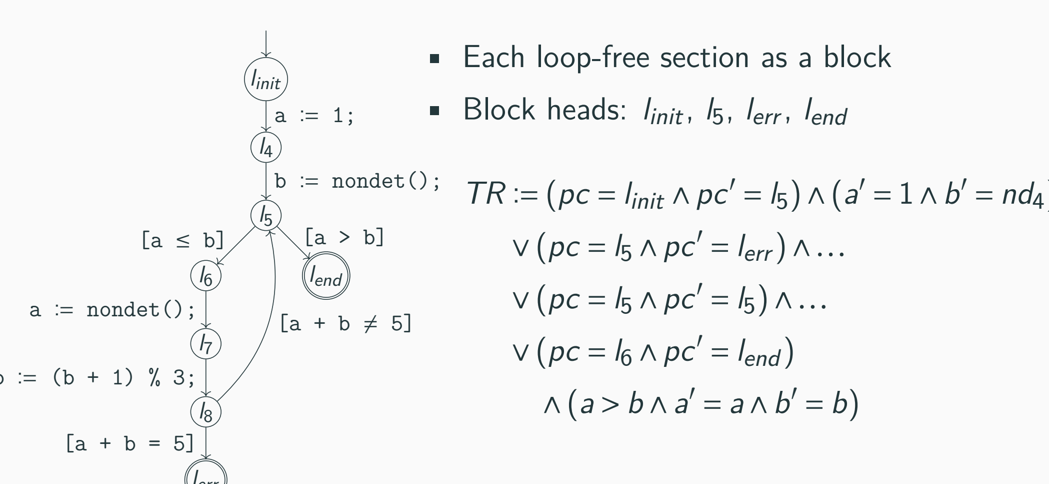
Design Principle

- Modular design with clear interfaces
- Exchange artifacts in standardized formats
  - Transition relation: VMT-LIB [17]
  - Sequential circuit: BTOR2 [32] (and AIGER [11])
  - HWMC witness: BTOR2
  - SV witness: GraphML (v1) and YAML (v2)


Frontend: C Program → Sequential Circuit

Large-Block Encoding

C Program → Transition Relation



LBE Details

C Program → Transition Relation

- Kratos2 [25] is employed (C → K2 IR → VMT-LIB)
- Function calls are inlined or treated as blocks
- `nondet()` calls are modeled by free input variables in TR

LBE Details

C Program → Transition Relation

- KRATOS2 [25] is employed (C → K2 IR → VMT-LIB)
- Function calls are inlined or treated as blocks
- `nondet()` calls are modeled by free input variables in TR
- CPV vs. other software verifiers:
  Monolithic transition relation vs. path-based exploration [3]

Functional Encoding

Transition Relation → Sequential Circuit

- Derive *explicit* next-state functions

Functional Encoding

Transition Relation → Sequential Circuit

- Derive *explicit* next-state functions
- Identify condition and state-update predicates in a block formula:

$$pc = l_6 \land pc' = l_{end} \land a > b \land a' = a \land b' = b \ldots$$

Functional Encoding

Transition Relation → Sequential Circuit

- Derive *explicit* next-state functions
- Identify condition and state-update predicates in a block formula:

$$pc = l_6 \land pc' = l_{end} \land a > b \land a' = a \land b' = b \ldots$$

- Next-state function for state variable a:

$$a' := \begin{cases} 1 & , if\ pc = l_{init} \\ nd_6 & , if\ pc = l_6 \land a \leq b \\ a & , if\ pc = l_6 \land a > b \\ * & , otherwise \end{cases}$$

Functional Encoding

Transition Relation → Sequential Circuit

- Derive *explicit* next-state functions
- Identify condition and state-update predicates in a block formula:

$$pc = l_6 \land pc' = l_{end} \land a > b \land a' = a \land b' = b \ldots$$

- Next-state function for state variable a:

$$a' := \begin{cases} 1 & , \textit{if } pc = l_{init} \\ nd_6 & , \textit{if } pc = l_6 \land a \leq b \\ a & , \textit{if } pc = l_6 \land a > b \\ * & , \textit{otherwise} \end{cases}$$

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

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:

$pc = l_{init}$ → • → • → • → $pc = l_{err}$

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:

$pc = l_{init}$ → • → • → • → $pc = l_{err}$

Termination Analysis (Liveness)

Can the program always terminate?

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

`justice` (+ `constraint`)

$pc = l_{init}$ → $pc \neq l_{end}$ → $pc \neq l_{end}$ → $pc \neq l_{end}$ → $pc \neq l_{end}$ → $pc \neq l_{end}$ → (back to the third state, forming a loop)

Backend: Model Checking

Model-Checking Algorithms

For checking safety properties:

- Bounded model checking (BMC) [12]
- k-induction (KI) [34]
- Interpolation-based model checking (IMC) [31]
- Property-directed reachability (IC3/PDR) [13, 21]
- Combined with CEGAR [19] using different abstraction techniques
  - Syntax-guided abstraction [23]
  - Predicate abstraction [27, 18]

Integrated Hardware Model Checkers

	Tool	Input formats	Engines
Bit-level (QF_BV)	ABC	Aiger	BMC, IMC, PDR
	rIC3	Aiger, Btor2	BMC, KI, IC3
Word-level (QF_ABV)	AVR	Btor2	BMC, KI, IC3sa
	Pono	Btor2	BMC, IMC, KI, IC3sa, IC3ia

Implementation Details

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

Implementation Details

- Use CoVeriTeam [4] to coordinate backend model checkers
  - Containerized execution with resource limits
  - Assemble command lines and extract tool outputs (logs and witnesses)
  - Easy to combine multiple tools
- Most tools do not support `justice` in Btor2
  → CPV does liveness-to-safety transformation [9]

Witness Translation

- HWMC witness:

Time step	State			Input	
	pc	a	b	nd_4	nd_6
0	l_{init}	-	-	`UINT_MAX`	-
1	l_5	1	`UINT_MAX`	-	**5**
2	l_{err}	5	0	-	-

Witness Translation

- HWMC witness:

Time step	State			Input	
	pc	a	b	nd_4	nd_6
0	l_{init}	-	-	`UINT_MAX`	-
1	l_5	1	`UINT_MAX`	-	**5**
2	l_{err}	5	0	-	-

- SV witness (test case): Enter `main`
  → `nondet()` returns **`UINT_MAX`** at line 4
  → `nondet()` returns **5** at line 6
  → reach `ERROR`

Current Limitations of CPV

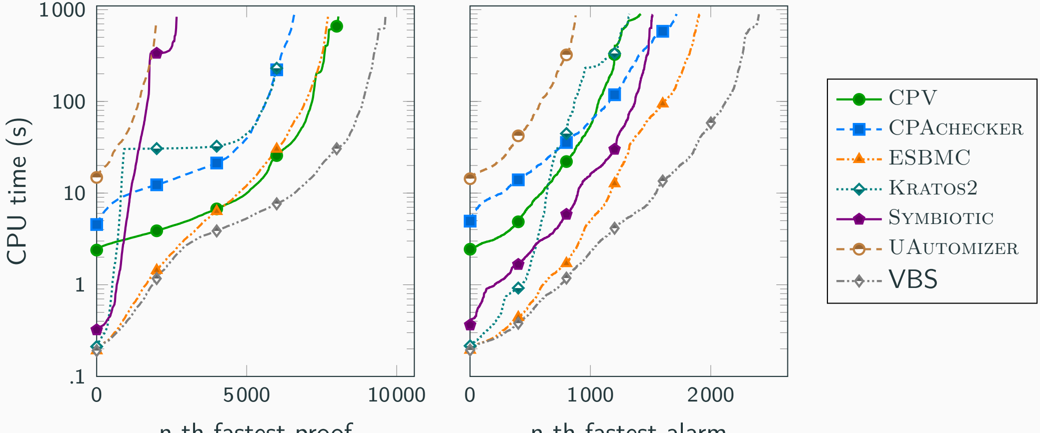
- Lack modeling for many library functions (e.g., `<math.h>`)
- Limited support for recursion (*finite* unrolling)
- Does not handle concurrency
- No support yet for correctness witness (invariant) translation
- Unsupported properties in SV-COMP:
  - Memory safety
  - Overflow detection

Experimental Evaluation

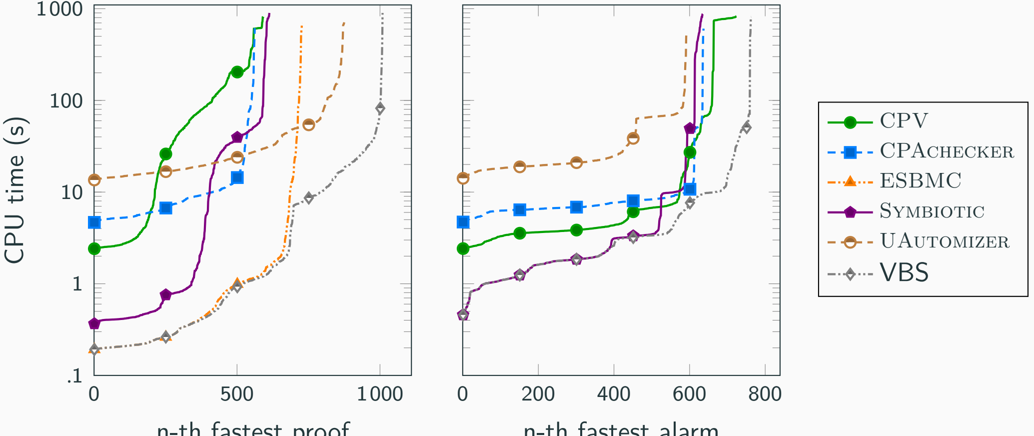
Experimental Setup

- Benchmark sets:
  ReachSafety and *Termination* tasks from SV-COMP 2026 without recursion
- Compared CPV 1.1 [16] against CPAchecker [8], ESBMC [29], Kratos2 [26], Symbiotic [1], and UAutomizer [20]
  (running in SV-COMP configurations)
- Resource limits for each run:
  15 min of CPU time, 15 GB of memory, and 2 CPU cores
- Reliable benchmarking by BenchExec [6]

Experimental Results: SV-COMP ReachSafety



Experimental Results: SV-COMP Termination



Experimental Results: Highlights

ReachSafety

1st:

- #Proofs overall
- Subcategories: *BitVectors*, *Hardware*

2nd:

- #Solved overall
- Subcategories: *Arrays*, *Hardness*, *ProductLines*, *XCSP*

Termination

1st:

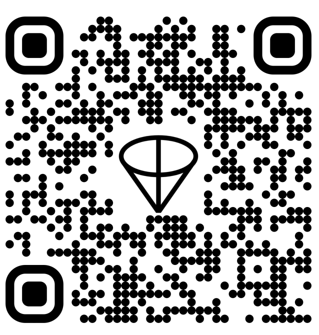
- #Alarms overall
- Subcategory: *Other*

2nd:

- #Solved overall
- Subcategory: *BitVectors*

Conclusion

- With CPV [15], we can
  - encode SV tasks as circuits
  - leverage powerful hardware model checkers as backend
- Strong performance in SV-COMP!
- Submitted translated BTOR2 circuits to HWMCC
- Next steps:
  - Correctness witness translation
  - Extend frontend support



gitlab.com/sosy-lab/software/cpv

References i

[1] Ayaziová, P., Jonáš, M., Mihalkovič, V., Sedláček, J., Strejček, J.: Symbiotic: Submission to SV-COMP 2026. Zenodo (2025). https://doi.org/10.5281/zenodo.17698724

[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

[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

References ii

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

[8] Beyer, D., Wendler, P.: CPAchecker release 4.2.2 (unix). Zenodo (2025). https://doi.org/10.5281/zenodo.17777566

[9] Biere, A., Artho, C., Schuppan, V.: Liveness checking as safety checking. In: Proc. FMICS. pp. 160–177. No. 2 in ENTSC 66, Elsevier (2002). https://doi.org/10.1016/S1571-0661(04)80410-9

[10] 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

[11] Biere, A., Heljanko, K., Wieringa, S.: AIGER 1.9 and beyond. Tech. Rep. 11/2, Institute for Formal Models and Verification, Johannes Kepler University (2011). https://doi.org/https://doi.org/10.35011/fmvtr.2011-2

[12] 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

[13] 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

[14] Brayton, R., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: Proc. CAV. pp. 24–40. LNCS 6174, Springer (2010). https://doi.org/10.1007/978-3-642-14295-6_5

[15] 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

[16] Chien, P.C., Lee, N.Z.: CPV release 1.1. Zenodo (2026). https://doi.org/10.5281/zenodo.18773927

[17] Cimatti, A., Griggio, A., Tonetta, S.: The VMT-LIB language and tools. In: Proc. SMT. CEUR Workshop Proceedings, vol. 3185, pp. 80–89. CEUR-WS.org (2022). https://ceur-ws.org/Vol-3185/extended9547.pdf

[18] Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: IC3 modulo theories via implicit predicate abstraction. In: Proc. TACAS. pp. 46–61. LNCS 8413, Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_4

[19] 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

[20] Dietsch, D., Bentele, M., Ebbinghaus, M., Heizmann, M., Klumpp, D., Podelski, A., Schüssele, F.: Ultimate Automizer SV-COMP 2026. Zenodo (2025). https://doi.org/10.5281/zenodo.17735224

References v

[21] 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

[22] Gadelha, M.R., Monteiro, F.R., Morse, J., Cordeiro, L.C., Fischer, B., Nicole, D.A.: ESBMC 5.0: An industrial-strength C model checker. In: Proc. ASE. pp. 888–891. ACM (2018). https://doi.org/10.1145/3238147.3240481

[23] 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

[24] 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

[25] 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

[26] Griggio, A., Jonáš, M.: Kratos version 2.3 (for Linux x64). https://kratos.fbk.eu/releases/kratos-2.3-linux64.tar.gz (2026), accessed: 2026-03-20

[27] 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

[28] 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

[29] Li, X.: ESBMC submission for the pre-run of SV-COMP'26. Zenodo (2025). https://doi.org/10.5281/zenodo.17741226

[30] 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

[31] 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

[32] 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

[33] 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, Springer (2026)

[34] 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

[35] Su, Y., Yang, Q., Ci, Y., Bu, T., Huang, Z.: The rIC3 hardware model checker. In: Proc. CAV (1). pp. 185–199. LNCS 15931, Springer (2025). https://doi.org/10.1007/978-3-031-98668-0_9