Cross-Application of Hardware and Software Verification

Po-Chun Chien

SoSy-Lab, LMU Munich

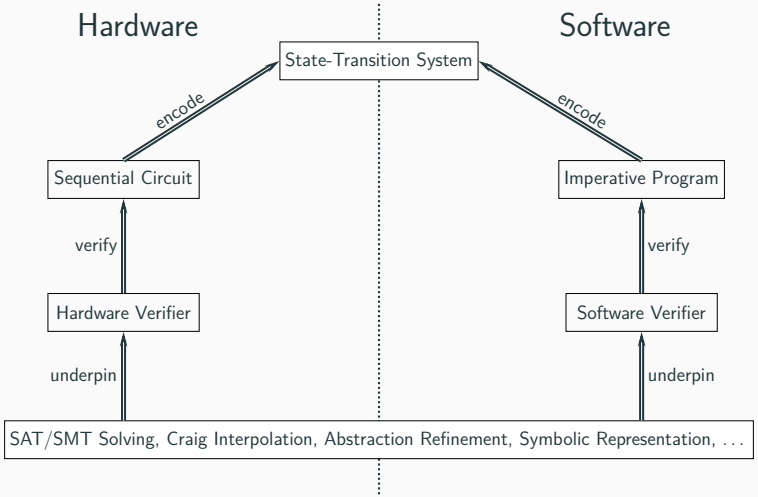
ConVeY Spring Workshop
2024-04-06 @ Luxembourg

SoSy-Lab
Software Systems

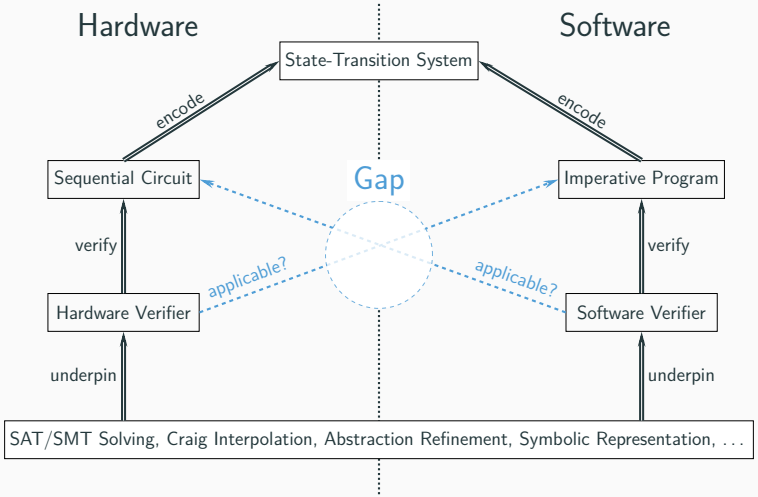
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

ConVeY

Motivation


Hardware
Software
State-Transition System
encode
encode
Sequential Circuit
Imperative Program
verify
verify
Hardware Verifier
Software Verifier
underpin
underpin
SAT/SMT Solving, Craig Interpolation, Abstraction Refinement, Symbolic Representation, ...

Motivation


Hardware
Software
State-Transition System
encode
encode
Sequential Circuit
Gap
Imperative Program
verify
verify
applicable?
applicable?
Hardware Verifier
Software Verifier
underpin
underpin
SAT/SMT Solving, Craig Interpolation, Abstraction Refinement, Symbolic Representation, ...

Briding Hardware and Software Verification

1. Cross-application of HW and SW verifiers

1.1 Applying SW analyzers to HW verification tasks

1.2 Applying HW model checkers to SW verification tasks

2. Knowledge consolidation of HW and SW verification

2.1 Transferring HW algorithms for SW verification

Agenda

1. Cross-application of HW and SW verifiers

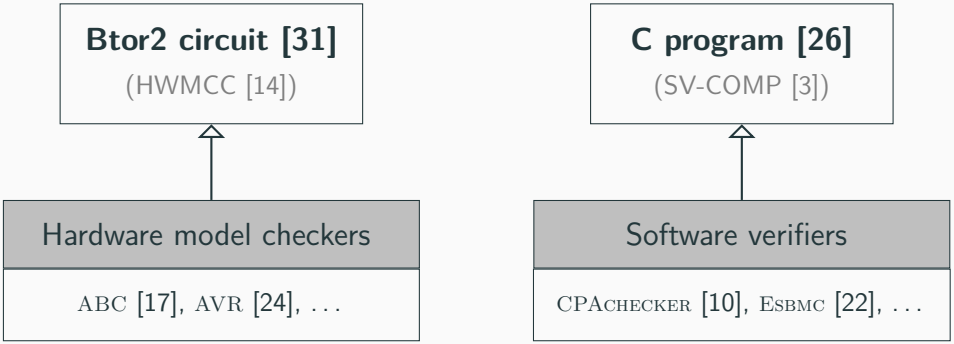
1.1 Applying SW analyzers to HW verification tasks

1.2 Applying HW model checkers to SW verification tasks

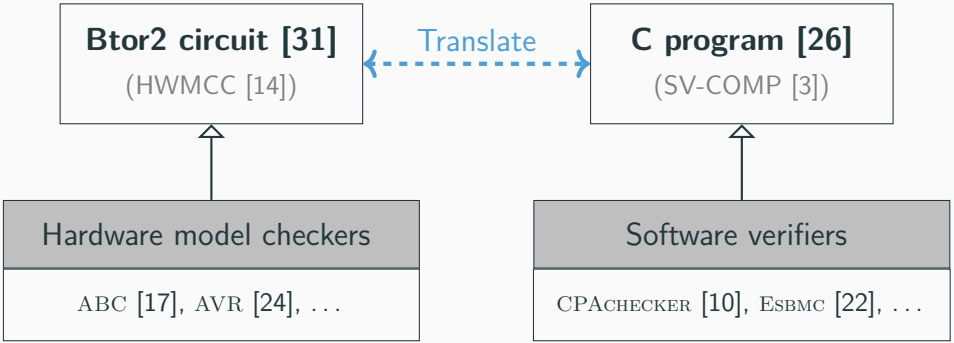
2. Knowledge consolidation of HW and SW verification

2.1 Transferring HW algorithms for SW verification

Translating Verification Tasks



Translating Verification Tasks



Agenda

1. Cross-application of HW and SW verifiers

1.1 Applying SW analyzers to HW verification tasks

1.2 Applying HW model checkers to SW verification tasks

2. Knowledge consolidation of HW and SW verification

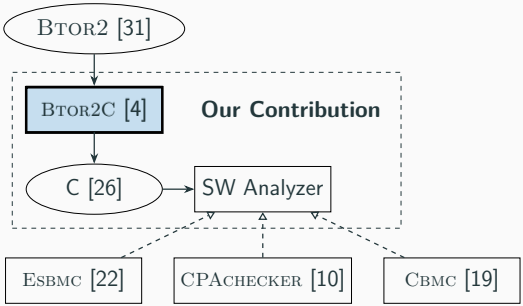
2.1 Transferring HW algorithms for SW verification

Employing SW Analyzers for HW Verification

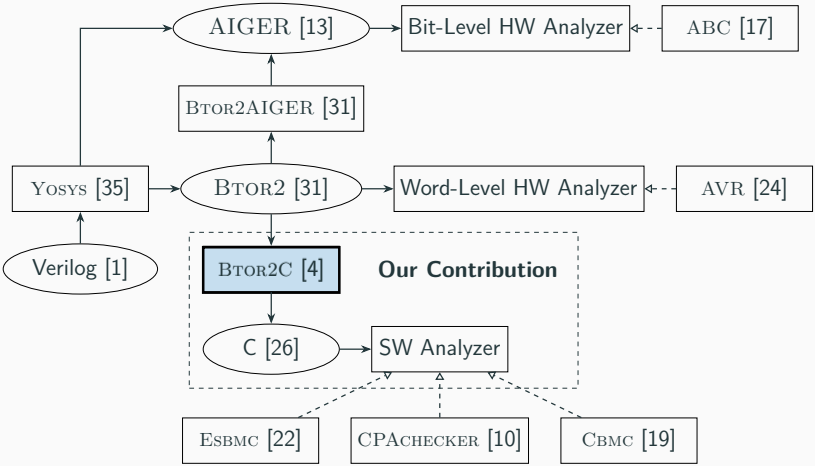
- Bridging Hardware and Software Analysis with Btor2C: A Word-Level-Circuit-to-C Translator (TACAS 2023 [4])
- **Btor2-Cert: A Certifying Hardware-Verification Framework Using Software Analyzers (TACAS 2024 [2])**
- Joint work with Zsófia Ádám, Dirk Beyer, Nian-Ze Lee, and Nils Sirrenberg

gitlab.com/sosy-lab/software/btor2-cert

HW-Analysis Tool Chain with Btor2C

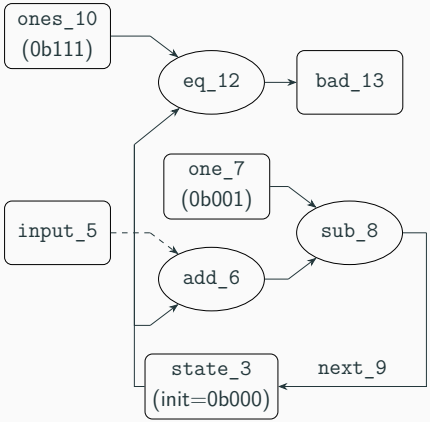


HW-Analysis Tool Chain with Btor2C



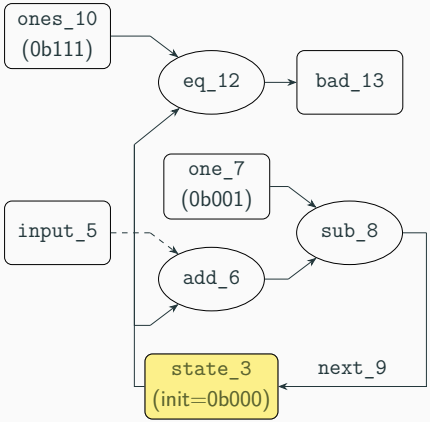
The Btor2 Language

```
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
```



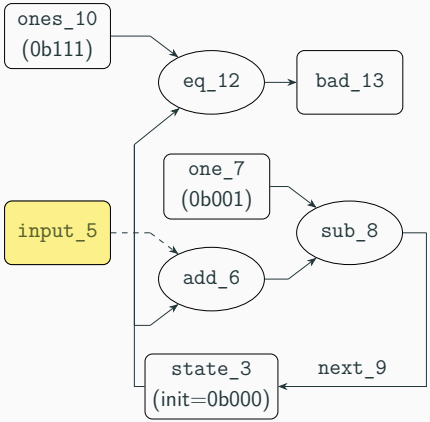
The Btor2 Language

```
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
```



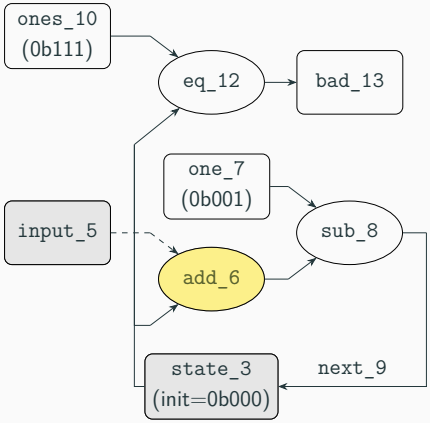
The Btor2 Language

```
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
```



The Btor2 Language

```
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
```



Translating Btor2 to C

```
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
```

```
1  void main() {
2    typedef unsigned char SORT_1;
3    typedef unsigned char SORT_11;
4    const SORT_1 var_2 = 0b000;
5    const SORT_1 var_7 = 0b001;
6    const SORT_1 var_10 = 0b111;
7    SORT_1 state_3 = var_2;
8    for (;;) {
9      SORT_1 input_5 = nondet_uchar();
10     input_5 = input_5 & 0b111;
11     SORT_11 var_12 = state_3 == var_10;
12     SORT_11 bad_13 = var_12;
13     if (bad_13) { ERROR: abort(); }
14     SORT_1 var_6 = state_3 + input_5;
15     var_6 = var_6 & 0b111;
16     SORT_1 var_8 = var_6 - var_7;
17     var_8 = var_8 & 0b111;
18     state_3 = var_8;
19   }
20 }
```

Translating Btor2 to C

```
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
```

```
1  void main() {
2    typedef unsigned char SORT_1;
3    typedef unsigned char SORT_11;
4    const SORT_1 var_2 = 0b000;
5    const SORT_1 var_7 = 0b001;
6    const SORT_1 var_10 = 0b111;
7    SORT_1 state_3 = var_2;
8    for (;;) {
9      SORT_1 input_5 = nondet_uchar();
10     input_5 = input_5 & 0b111;
11     SORT_11 var_12 = state_3 == var_10;
12     SORT_11 bad_13 = var_12;
13     if (bad_13) { ERROR: abort(); }
14     SORT_1 var_6 = state_3 + input_5;
15     var_6 = var_6 & 0b111;
16     SORT_1 var_8 = var_6 - var_7;
17     var_8 = var_8 & 0b111;
18     state_3 = var_8;
19   }
20 }
```

Translating Btor2 to C

```
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
```

```
1  void main() {
2    typedef unsigned char SORT_1;
3    typedef unsigned char SORT_11;
4    const SORT_1 var_2 = 0b000;
5    const SORT_1 var_7 = 0b001;
6    const SORT_1 var_10 = 0b111;
7    SORT_1 state_3 = var_2;
8    for (;;) {
9      SORT_1 input_5 = nondet_uchar();
10     input_5 = input_5 & 0b111;
11     SORT_11 var_12 = state_3 == var_10;
12     SORT_11 bad_13 = var_12;
13     if (bad_13) { ERROR: abort(); }
14     SORT_1 var_6 = state_3 + input_5;
15     var_6 = var_6 & 0b111;
16     SORT_1 var_8 = var_6 - var_7;
17     var_8 = var_8 & 0b111;
18     state_3 = var_8;
19   }
20 }
```

Translating Btor2 to C

```
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
```

```
1  void main() {
2    typedef unsigned char SORT_1;
3    typedef unsigned char SORT_11;
4    const SORT_1 var_2 = 0b000;
5    const SORT_1 var_7 = 0b001;
6    const SORT_1 var_10 = 0b111;
7    SORT_1 state_3 = var_2;
8    for (;;) {
9      SORT_1 input_5 = nondet_uchar();
10     input_5 = input_5 & 0b111;
11     SORT_11 var_12 = state_3 == var_10;
12     SORT_11 bad_13 = var_12;
13     if (bad_13) { ERROR: abort(); }
14     SORT_1 var_6 = state_3 + input_5;
15     var_6 = var_6 & 0b111;
16     SORT_1 var_8 = var_6 - var_7;
17     var_8 = var_8 & 0b111;
18     state_3 = var_8;
19   }
20 }
```

Evaluation of SW Analyzers on HW Tasks

On 1008 safe and 490 unsafe BTOR2 verification tasks:

Evaluation of SW Analyzers on HW Tasks

On 1008 safe and 490 unsafe BTOR2 verification tasks:

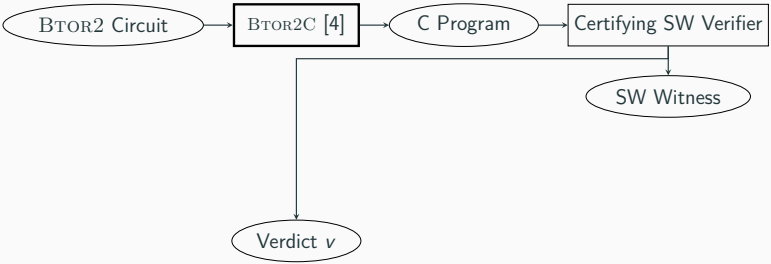
- **RQ1**: How do SW analyzers perform on HW tasks?
  Quite decent! Each analyzer showcases different strength

Evaluation of SW Analyzers on HW Tasks

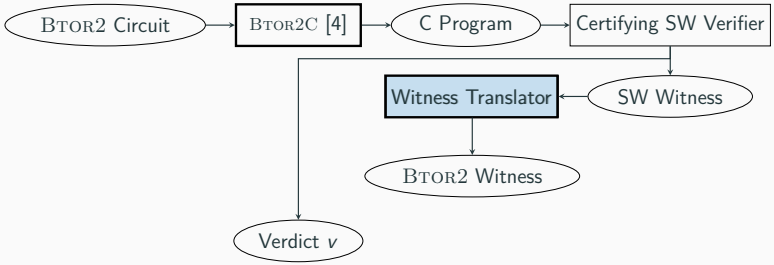
On 1008 safe and 490 unsafe BTOR2 verification tasks:

- **RQ1**: How do SW analyzers perform on HW tasks?
  Quite decent! Each analyzer showcases different strength
- **RQ2**: Can SW analyzers complement HW model checkers?
  Yes, **43** tasks were uniquely solved by SW verifiers

Btor2-Cert Framework

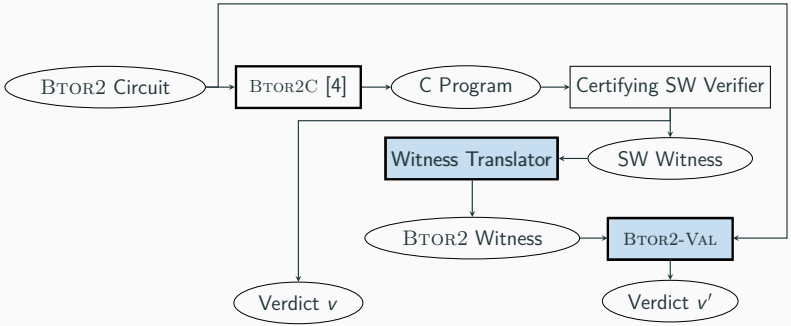


Btor2-Cert Framework



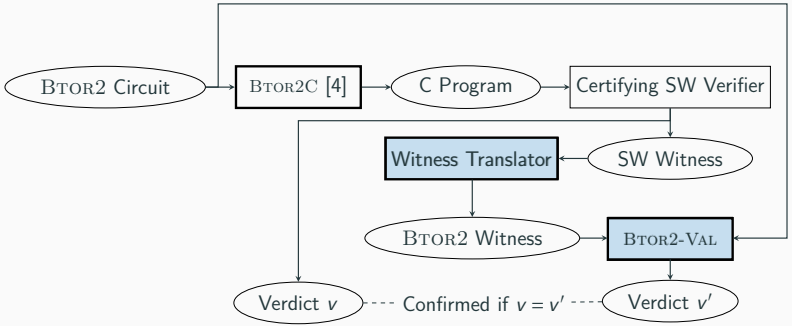
- Translate verification witness: automata [7] to BTOR2 [31]

Btor2-Cert Framework



- Translate verification witness: automata [7] to BTOR2 [31]
- BTOR2-VAL: witness validator for BTOR2
  - Correctness: circuit instrumentation and validation via verification [12]
  - Violation: execution-based validation [8] via BTORSIM

Btor2-Cert Framework



- Translate verification witness: automata [7] to BTOR2 [31]
- BTOR2-VAL: witness validator for BTOR2
  - Correctness: circuit instrumentation and validation via verification [12]
  - Violation: execution-based validation [8] via BTORSIM

What Can Btor2-Cert Do for You?

- For HW designers
  - Certified verification results from SW tools
  - Explanations in HW domain as test cases or invariants

What Can Btor2-Cert Do for You?

- For HW designers
  - Certified verification results from SW tools
  - Explanations in HW domain as test cases or invariants
- For developers of SW analyzers
  - Validator: witness translation and BTOR2-VAL for performance comparison

What Can Btor2-Cert Do for You?

- For HW designers
  - Certified verification results from SW tools
  - Explanations in HW domain as test cases or invariants
- For developers of SW analyzers
  - Validator: witness translation and BTOR2-VAL for performance comparison
  - Verifier: testbed for witness generation
    → Discovery of several bugs in mature software verifiers[1]

[1] `https://www.sosy-lab.org/research/btor2-cert/`

Agenda

1. Cross-application of HW and SW verifiers

1.1 Applying SW analyzers to HW verification tasks

1.2 Applying HW model checkers to SW verification tasks

2. Knowledge consolidation of HW and SW verification

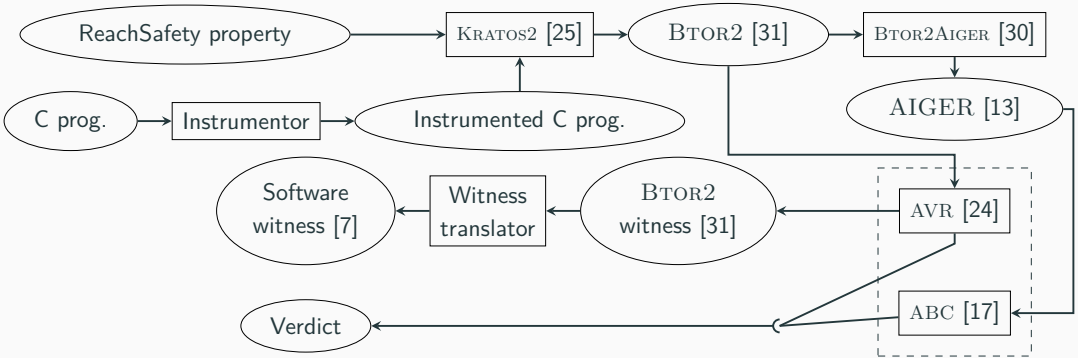
2.1 Transferring HW algorithms for SW verification

Utlizing HW Model Checkers for Software Verification

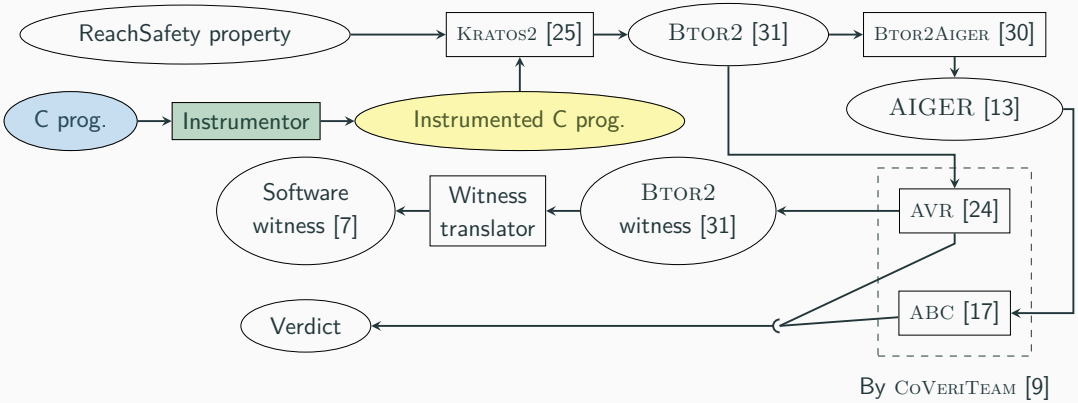
- **CPV: A Circuit-Based Program Verifier**
  (SV-COMP 2024 [18])
- Joint work with Dirk Beyer and Nian-Ze Lee

gitlab.com/sosy-lab/
software/cpv

System Architecure

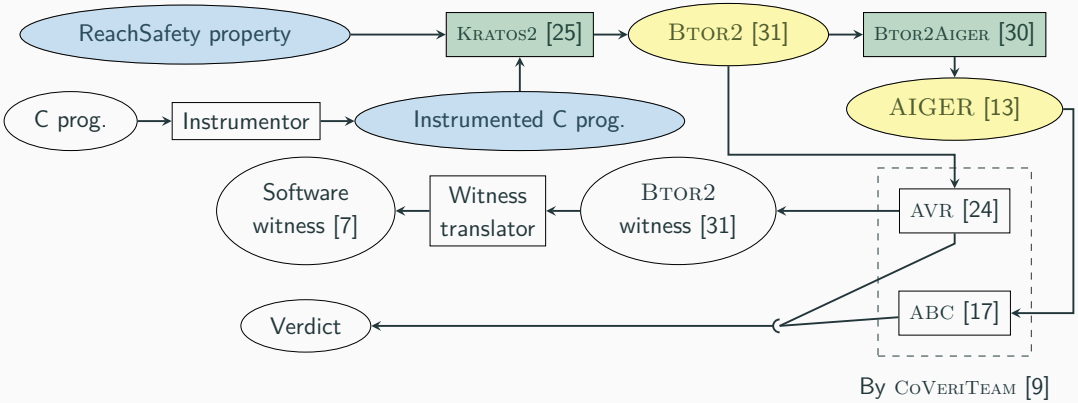


System Architecure



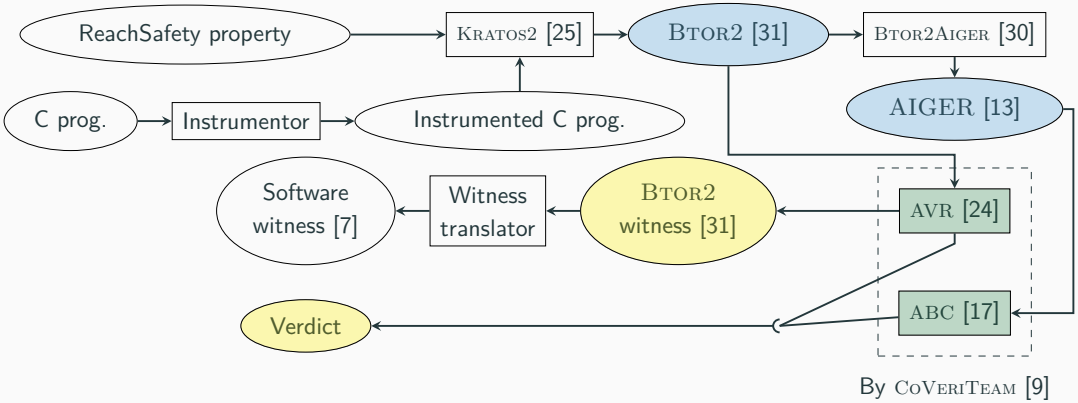
1. Instrument the input program

System Architecure



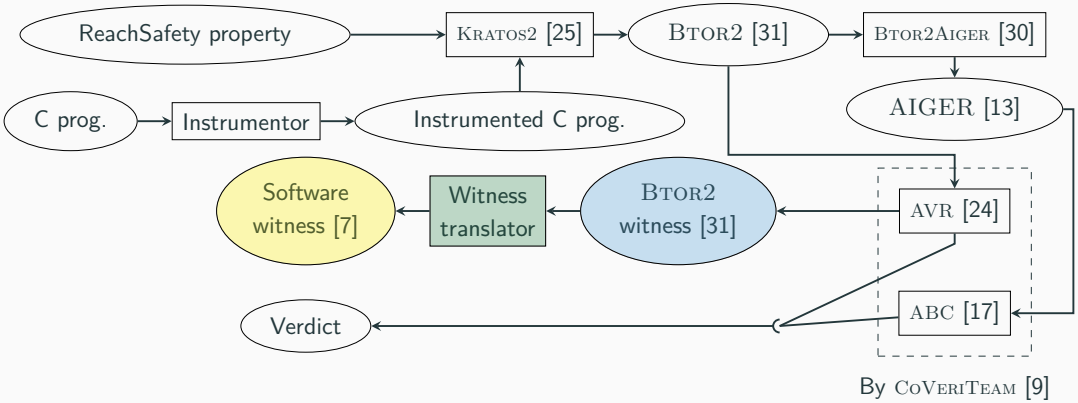
2. Translate the program to a circuit

System Architecure



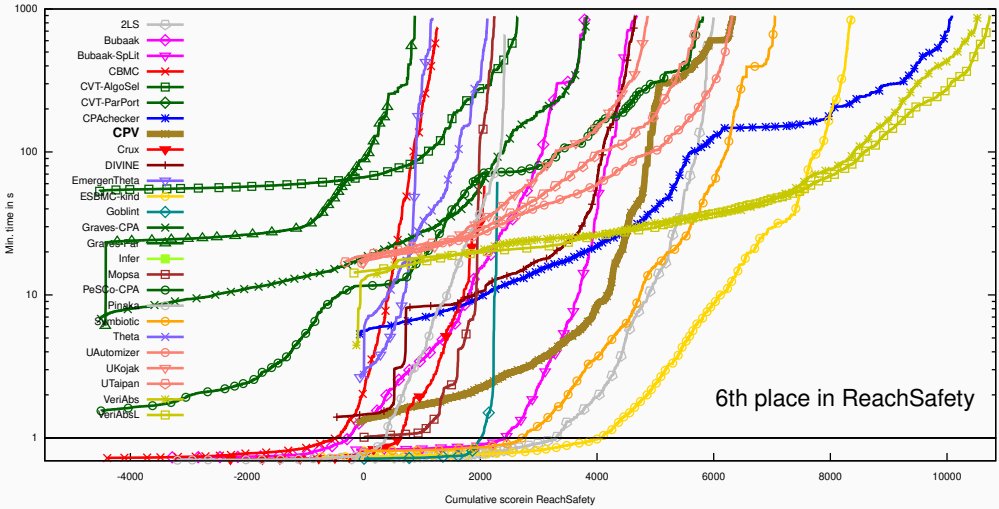
3. Verify the translated circuit with hardware model checkers

System Architecure



4. Translate the Btor2 witness back to software domain

Results in SV-COMP



Agenda

1. Cross-application of HW and SW verifiers

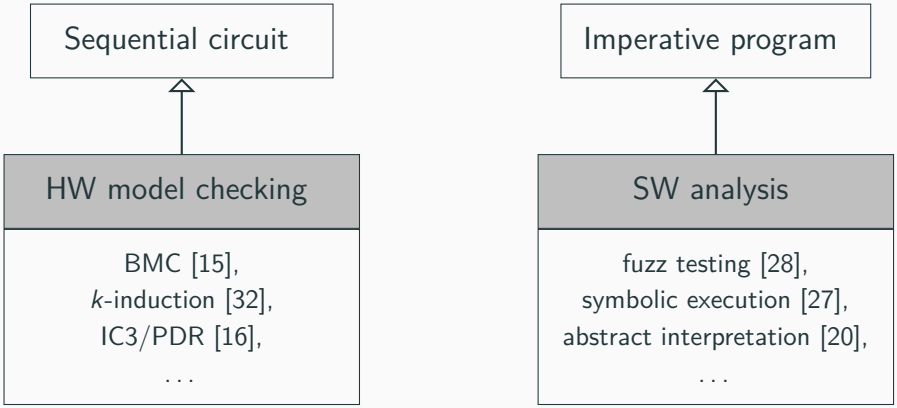
1.1 Applying SW analyzers to HW verification tasks

1.2 Applying HW model checkers to SW verification tasks

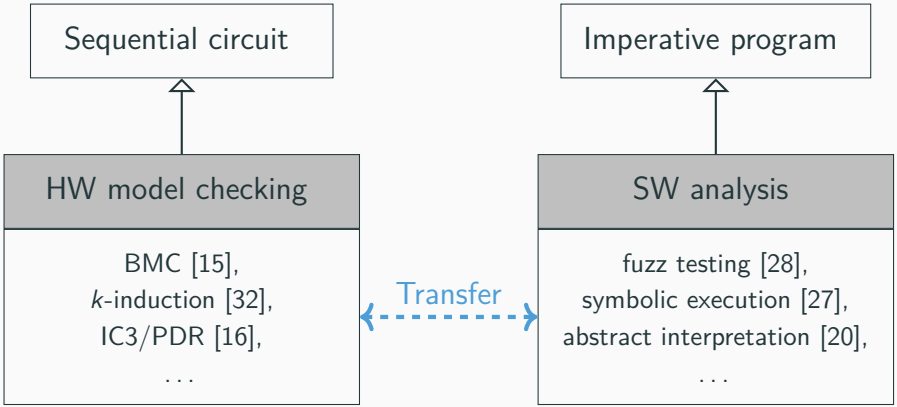
2. Knowledge consolidation of HW and SW verification

2.1 Transferring HW algorithms for SW verification

Transferring Verification Techniques Across Domains



Transferring Verification Techniques Across Domains



Agenda

1. Cross-application of HW and SW verifiers

1.1 Applying SW analyzers to HW verification tasks

1.2 Applying HW model checkers to SW verification tasks

2. Knowledge consolidation of HW and SW verification

2.1 Transferring HW algorithms for SW verification

Transferring HW Model Checking for SW Verifcation

- Interpolation and SAT-Based Model Checking Revisited: Adoption to Software Verification (JAR 2024 [11])
- **Augmenting Interpolation-Based Model Checking with Auxiliary Invariants (SPIN 2024 [6])**
- Joint work with Dirk Beyer, Marek Jankola, Nian-Ze Lee, and Philipp Wendler

www.sosy-lab.org/research/imc-df/

Interpolation-Based HW Model Checking

- HW algorithms implemented in CPAchecker:
  - *Interpolation-Based Model Checking (IMC)* [29]
  - *Interpolation-Sequence-Based Model Checking (ISMC)* [33]
  - *Dual Approximated Reachability (DAR)* [34]

Interpolation-Based HW Model Checking

- HW algorithms implemented in CPAchecker:
  - *Interpolation-Based Model Checking (IMC)* [29]
  - *Interpolation-Sequence-Based Model Checking (ISMC)* [33]
  - *Dual Approximated Reachability (DAR)* [34]
- Our study shows:
  - Characteristics of these algorithms transferrable

Interpolation-Based HW Model Checking

- HW algorithms implemented in CPAchecker:
  - *Interpolation-Based Model Checking (IMC)* [29]
  - *Interpolation-Sequence-Based Model Checking (ISMC)* [33]
  - *Dual Approximated Reachability (DAR)* [34]
- Our study shows:
  - Characteristics of these algorithms transferrable
  - These HW algorithms can tackle tasks unsolvable by existing methods
    → cross-disciplinary adoption is beneficial

Augmenting IMC with Auxiliary Invariants

- Strengthen Craig interpolants with auxiliary invariants

Augmenting IMC with Auxiliary Invariants

- Strengthen Craig interpolants with auxiliary invariants
- Augmented vs. plain IMC
  - Improve effectiveness
  - Reduce elapsed wall-time

Conclusion

- Transformation between different representations to leverage their unique strengths

Conclusion

- Transformation between different representations to leverage their unique strengths
- Cooperative and cross-disciplinary approaches are beneficial

Conclusion

- Transformation between different representations to leverage their unique strengths
- Cooperative and cross-disciplinary approaches are beneficial
- Know the state of the art to avoid reinventing the wheel!

Conclusion

- Transformation between different representations to leverage their unique strengths
- Cooperative and cross-disciplinary approaches are beneficial
- Know the state of the art to avoid reinventing the wheel!
- Ultimate goal:
  - HW/SW co-verification
  - Tackle more complex heterogeneous systems

Advertisement

- Try our tools!

Btor2-Cert [2]

CPV [18]

CPAchecker [10]

Advertisement

- Try our tools!

BTOR2-CERT [2]

CPV [18]

CPACHECKER [10]

- **Join our talks at ETAPS!**

COOP:	Sun. 10:30 (more on our work)	SPIN:	Wed. 11:30 (IMC + inv.)
SV-COMP:	Mon. 14:00 (CPV)	TACAS:	Thu. 12:00 (BTOR2-CERT)

References i

[1] IEEE standard for Verilog hardware description language (2006). https://doi.org/10.1109/IEEESTD.2006.99495

[2] Ádám, Z., Beyer, D., Chien, P.C., Lee, N.Z., Sirrenberg, N.: Btor2-Cert: A certifying hardware-verification framework using software analyzers. In: Proc. TACAS. pp. 129–149. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_7

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

[4] Beyer, D., Chien, P.C., Lee, N.Z.: Bridging hardware and software analysis with Btor2C: A word-level-circuit-to-C translator. In: Proc. TACAS. pp. 152–172. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_12

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

References ii

[6] Beyer, D., Chien, P.C., Lee, N.Z.: Augmenting interpolation-based model checking with auxiliary invariants. In: Proc. SPIN. Springer (2024)

[7] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Lemberger, T., Tautschnig, M.: Verification witnesses. ACM Trans. Softw. Eng. Methodol. **31**(4), 57:1–57:69 (2022). https://doi.org/10.1145/3477579

[8] Beyer, D., Dangl, M., Lemberger, T., Tautschnig, M.: Tests from witnesses: Execution-based validation of verification results. In: Proc. TAP. pp. 3–23. LNCS 10889, Springer (2018). https://doi.org/10.1007/978-3-319-92994-1_1

[9] 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

[10] 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 iii

[11] Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. J. Autom. Reasoning (2024), accepted, preprint available via https://doi.org/10.48550/arXiv.2208.05046

[12] Beyer, D., Spiessl, M.: MetaVal: Witness validation via verification. In: Proc. CAV. pp. 165–177. LNCS 12225, Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_10

[13] Biere, A.: The AIGER And-Inverter Graph (AIG) format version 20071012. Tech. Rep. 07/1, Institute for Formal Models and Verification, Johannes Kepler University (2007). https://doi.org/10.35011/fmvtr.2007-1

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

[15] 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 iv

[16] 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

[17] 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

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

[19] 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

[20] Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for the static analysis of programs by construction or approximation of fixpoints. In: Proc. POPL. pp. 238–252. ACM (1977). https://doi.org/10.1145/512950.512973

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] ISO/IEC JTC 1/SC 22: ISO/IEC 9899-2018: Information technology — Programming Languages — C. International Organization for Standardization (2018), https://www.iso.org/standard/74528.html

References vi

[27] King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976). https://doi.org/10.1145/360248.360252

[28] Manès, V.J.M., Han, H., Han, C., Cha, S.K., Egele, M., Schwartz, E.J., Woo, M.: The art, science, and engineering of fuzzing: A survey. IEEE Trans. Software Eng. **47**(11), 2312–2331 (2021). https://doi.org/10.1109/TSE.2019.2946563

[29] 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

[30] Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Source-code repository of BTOR2, BTORMC, and BOOLECTOR 3.0. https://github.com/Boolector/btor2tools, accessed: 2023-01-29

[31] 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

[32] 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

References vii

[33] Vizel, Y., Grumberg, O.: Interpolation-sequence based model checking. In: Proc. FMCAD. pp. 1–8. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351148

[34] Vizel, Y., Grumberg, O., Shoham, S.: Intertwined forward-backward reachability analysis using interpolants. In: Proc. TACAS. pp. 308–323. LNCS 7795, Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_22

[35] Wolf, C.: Yosys open synthesis suite. https://yosyshq.net/yosys/, accessed: 2023-01-29