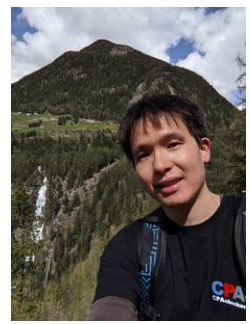
Bridging Hardware and Software Formal Verification for Reliable Computing Systems

Nian-Ze Lee

LMU Munich, Germany

April 7, COOP 2024



LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

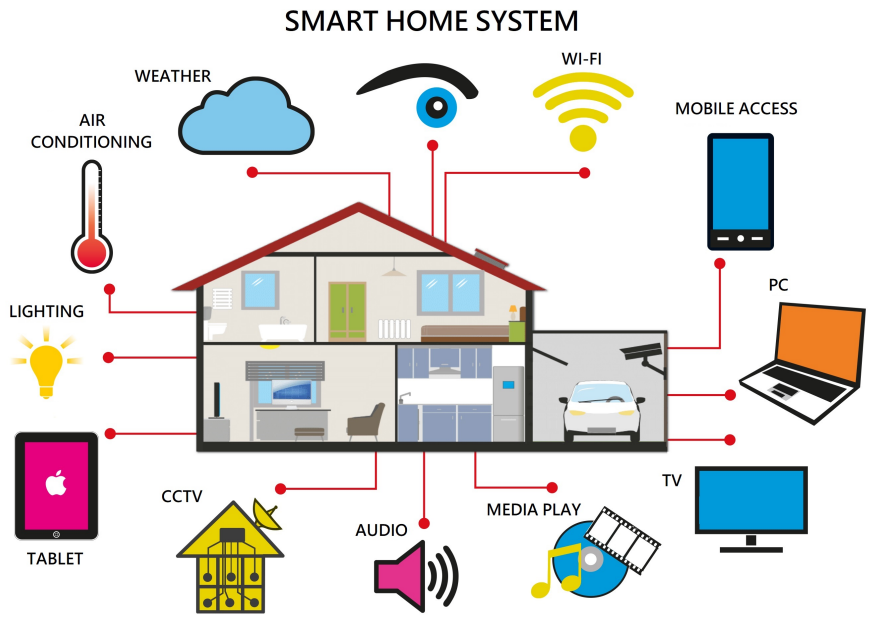
SoSy-Lab Software Systems

About Me

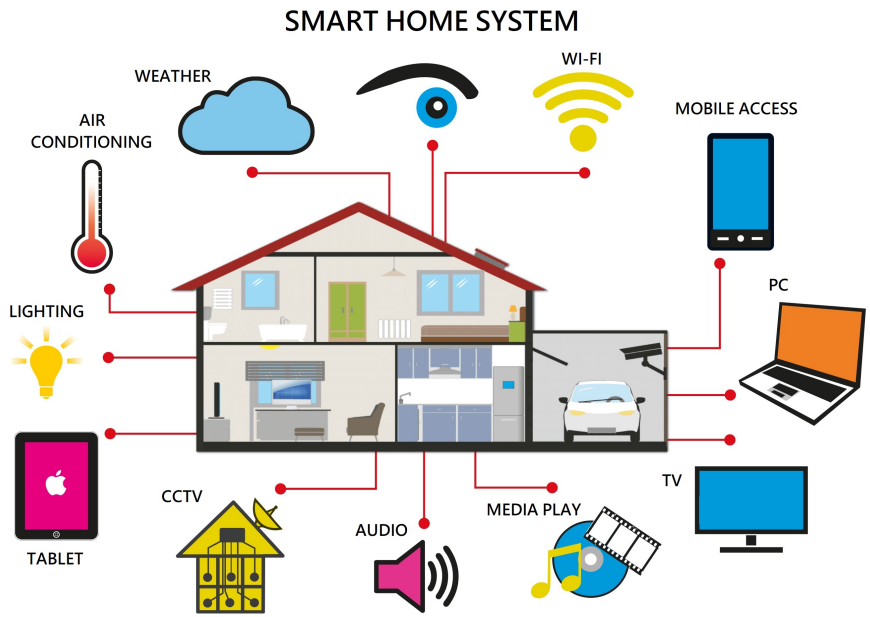
- Since 2021, PostDoc in Computer Science, LMU Munich
- 2021, Ph.D. in Electronics Engineering, NTU, Taipei

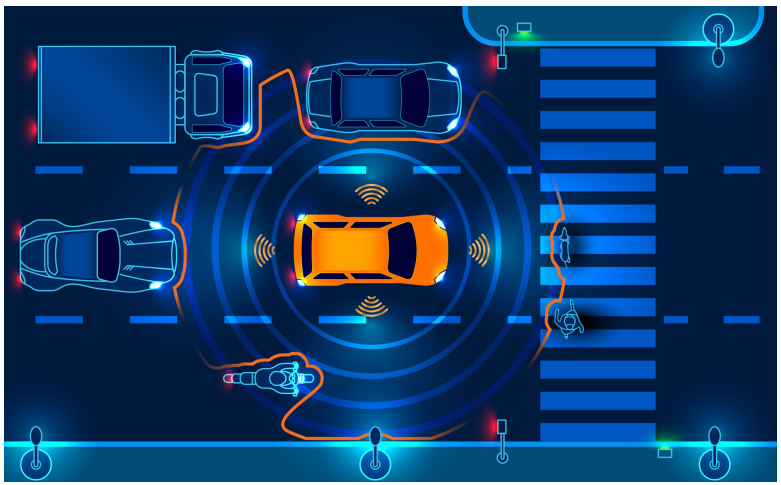
Modern Computing Systems

Modern Computing Systems

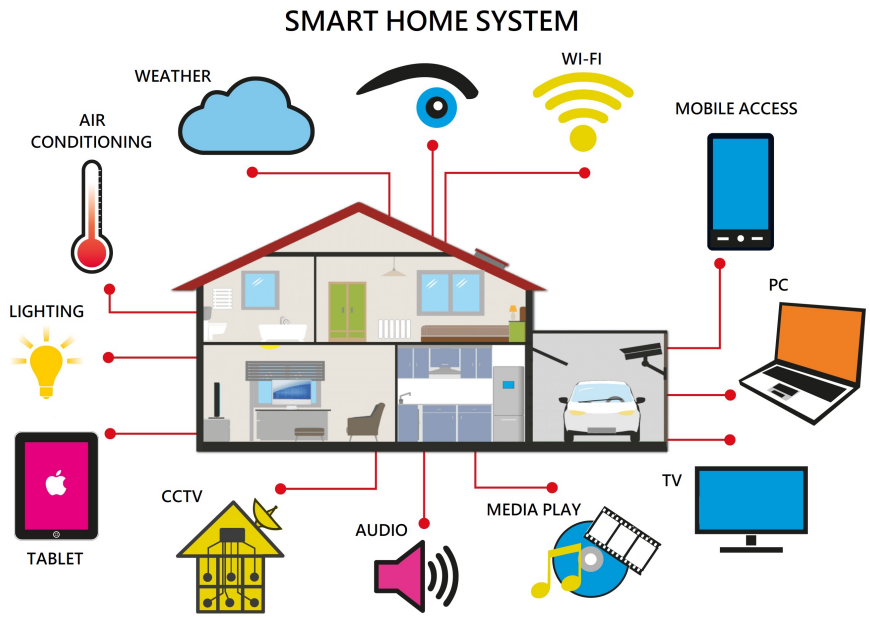


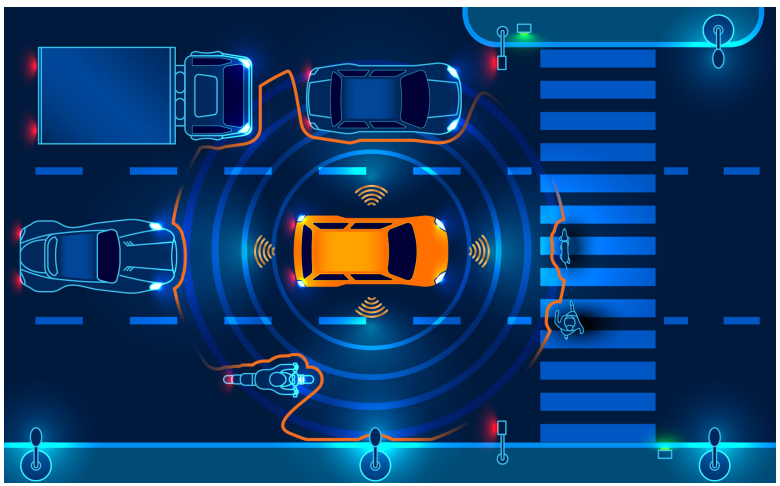
Modern Computing Systems





Modern Computing Systems





*Images from the Internet

Heterogeneous Components

- Hardware VLSI circuits
- Software programs
- Cyber-physical devices

Challenges of Analyzing Heterogeneous Systems

- Complex interactions
- Distributed computing
- Modeling difficulties

Challenges of Analyzing Heterogeneous Systems

- Complex interactions
- Distributed computing
- Modeling difficulties
- Scalability
- Tool development
- ...

Challenges of Analyzing Heterogeneous Systems

- Complex interactions
- Distributed computing
- Modeling difficulties
- Scalability
- Tool development
- ...

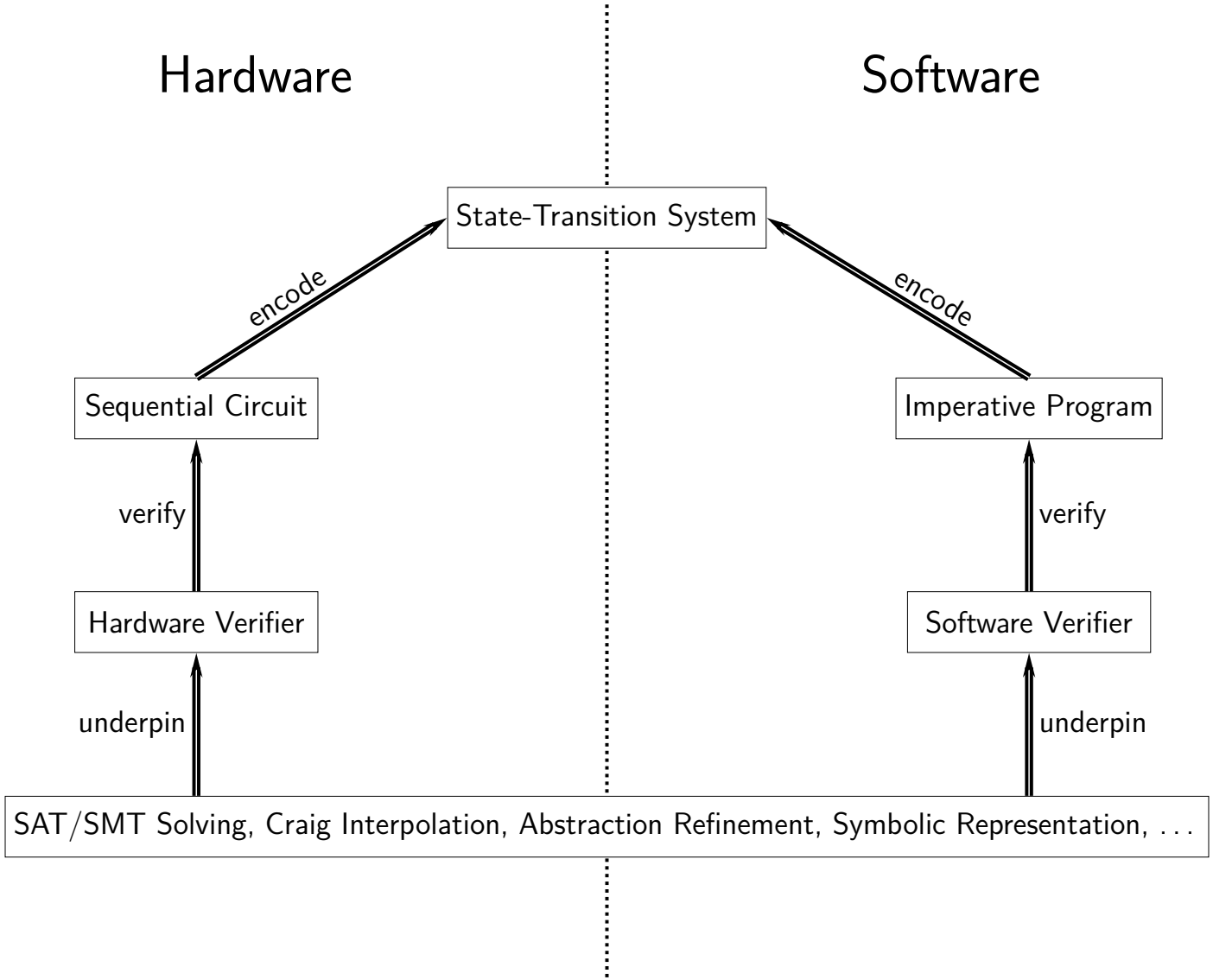
Cooperative and cross-disciplinary approaches are necessary.

Challenges of Analyzing Heterogeneous Systems

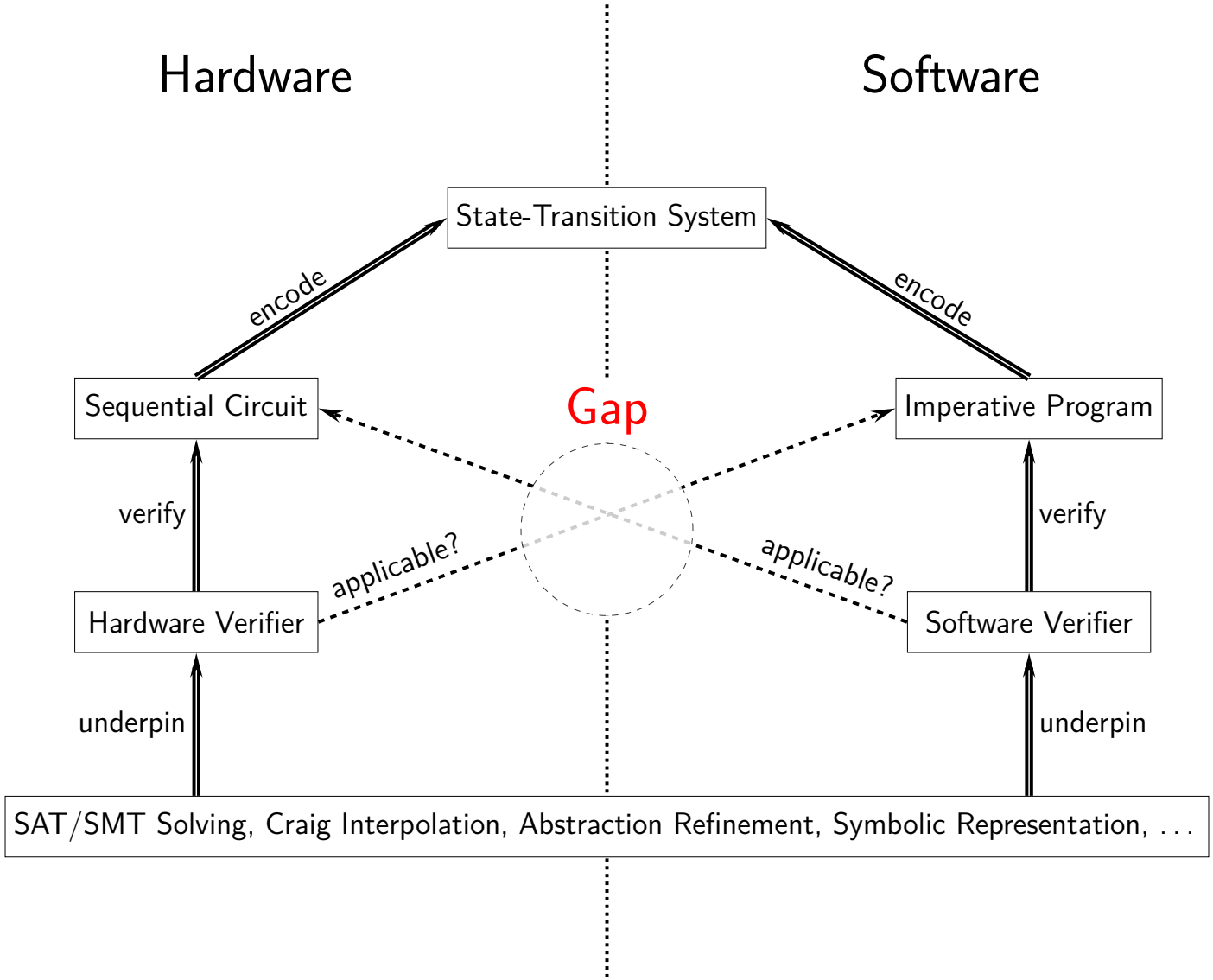
- Complex interactions
- Distributed computing
- Modeling difficulties
- Scalability
- Tool development
- ...

Cooperative and cross-disciplinary approaches are necessary. We should know the state of the art before developing new methods to avoid reinventing the wheel!

Conventional HW and SW Formal Verification



Conventional HW and SW Formal Verification



Bridging HW and SW Analysis

- Current work
  - Cross-application of each other's advancements
  - Consolidation of formal-verification knowledge

Bridging HW and SW Analysis

- Current work
  - Cross-application of each other's advancements
  - Consolidation of formal-verification knowledge

Software verifiers can detect bugs that hardware verifiers overlook.

Hardware-verification algorithms can improve software verifiers.

Bridging HW and SW Analysis

- Current work
  - Cross-application of each other's advancements
  - Consolidation of formal-verification knowledge

Software verifiers can detect bugs that hardware verifiers overlook.

Hardware-verification algorithms can improve software verifiers.

- Outlook: collaboration/mutual learning for new challenges
  - Scalable full-stack verification of the entire system
  - Neural networks, embedded or cyber-physical systems

Agenda

1. Cross-application of HW and SW formal verification
1.1 Applying SW analyzers to HW verification tasks
1.2 Applying HW model checkers to SW verification tasks

2. Knowledge consolidation of formal verification
2.1 Transferability of HW algorithms to SW verification

3. Reflection and outlook

Agenda

1. Cross-application of HW and SW formal verification
1.1 Applying SW analyzers to HW verification tasks
1.2 Applying HW model checkers to SW verification tasks

2. Knowledge consolidation of formal verification
2.1 Transferability of HW algorithms to SW verification

3. Reflection and outlook

Cross-Application of HW and SW Techniques

- Applying SW analyzers to HW verification tasks
  - Publications at TACAS 2023 and 2024 [1, 2]

Cross-Application of HW and SW Techniques

- Applying SW analyzers to HW verification tasks
  - Publications at TACAS 2023 and 2024 [1, 2]
- Applying HW model checkers to SW verification tasks
  - CPV: a new circuit-based verifier in SV-COMP 2024 [3]

Cross-Application of HW and SW Techniques

- Applying SW analyzers to HW verification tasks
  - Publications at TACAS 2023 and 2024 [1, 2]
- Applying HW model checkers to SW verification tasks
  - CPV: a new circuit-based verifier in SV-COMP 2024 [3]
- Adopting HW algorithms to verify programs
  - Publication in *Journal of Automated Reasoning* [4]

Cross-Application of HW and SW Techniques

- Applying SW analyzers to HW verification tasks
  - Publications at TACAS 2023 and 2024 [1, 2]
- Applying HW model checkers to SW verification tasks
  - CPV: a new circuit-based verifier in SV-COMP 2024 [3]
- Adopting HW algorithms to verify programs
  - Publication in *Journal of Automated Reasoning* [4]

Agenda

1. Cross-application of HW and SW formal verification
1.1 Applying SW analyzers to HW verification tasks
1.2 Applying HW model checkers to SW verification tasks

2. Knowledge consolidation of formal verification
2.1 Transferability of HW algorithms to SW verification

3. Reflection and outlook

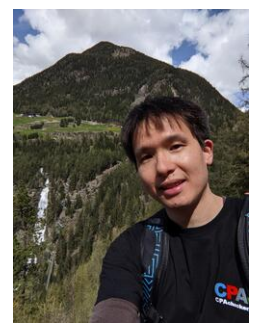
Bridging Hardware and Software Analysis with Btor2C: A Word-Level-Circuit-to-C Translator

Dirk Beyer, Po-Chun Chien, and Nian-Ze Lee

LMU Munich, Germany

Published at the 29th International Conference on

Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2023



Research Questions

- Evaluating SW analyzers on HW verification tasks
- Complementing HW verification with SW analyzers

Research Questions

- Evaluating SW analyzers on HW verification tasks
- Complementing HW verification with SW analyzers

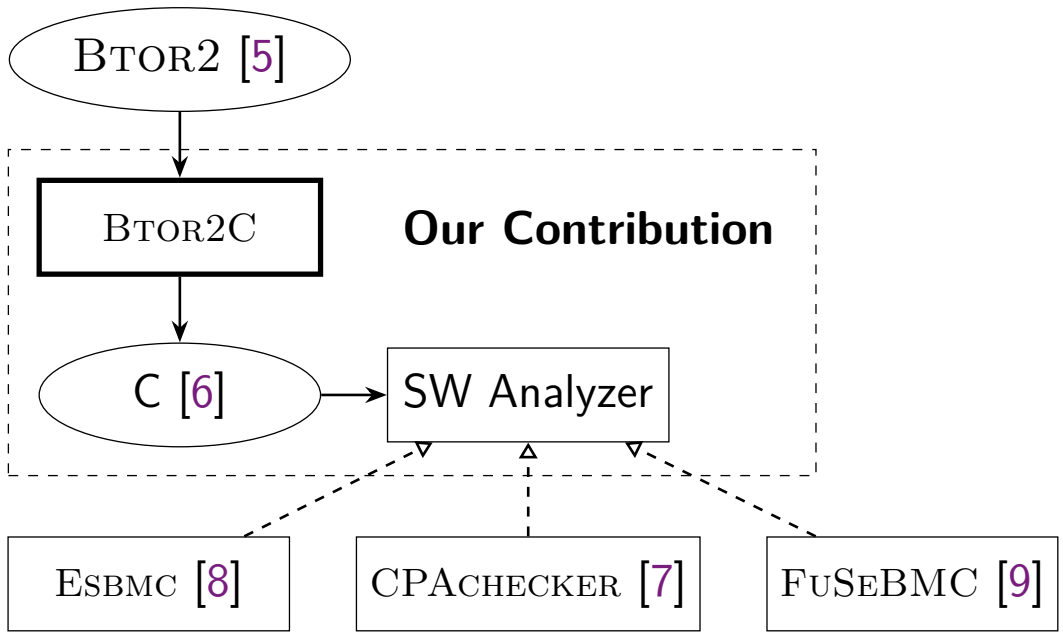
43 HW-verification tasks uniquely solved by SW analyzers in our evaluation

Research Questions

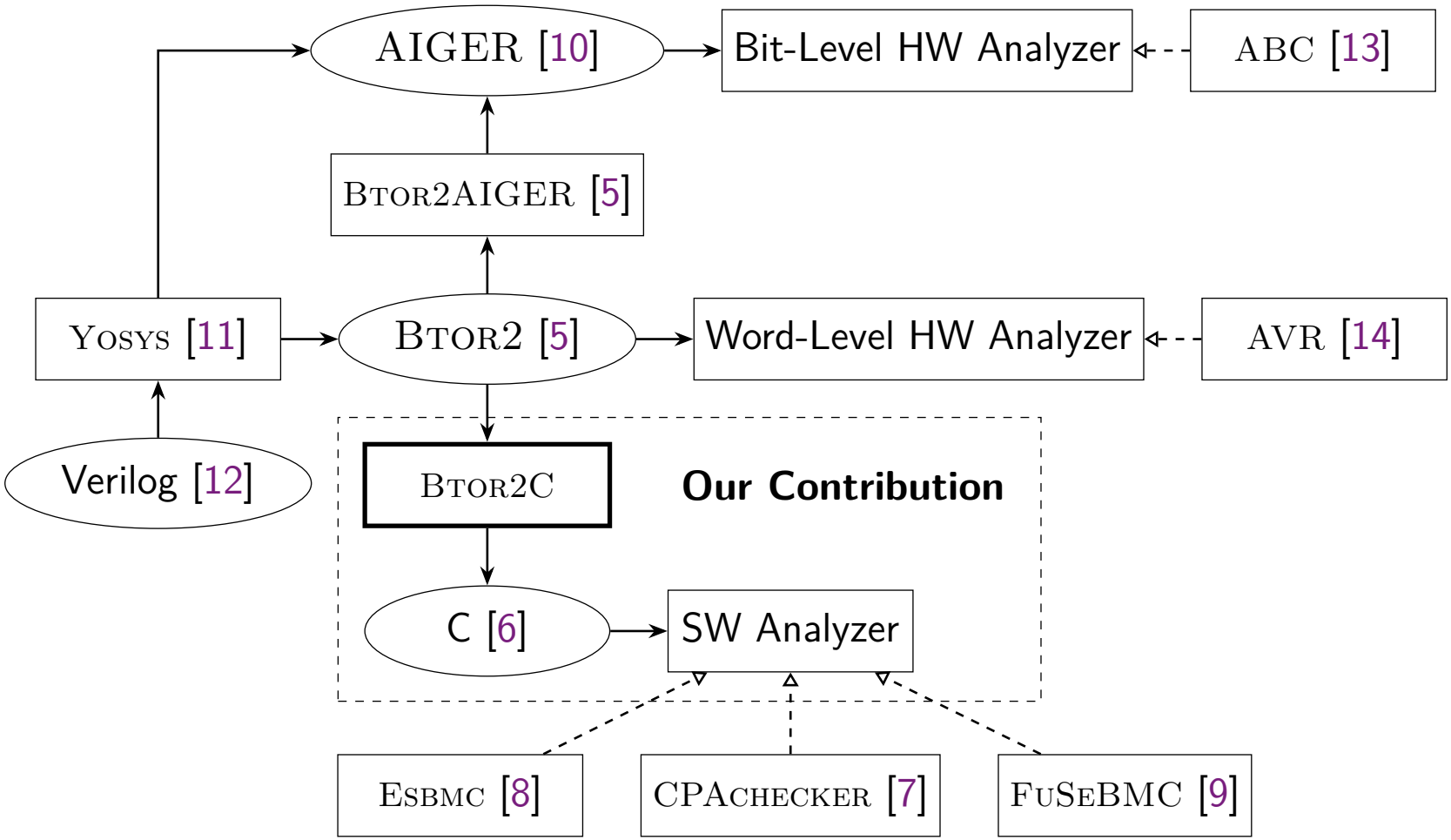
- Evaluating SW analyzers on HW verification tasks
- Complementing HW verification with SW analyzers

43 HW-verification tasks uniquely solved by SW analyzers in our evaluation $\rightarrow$ enhance HW quality assurance

Our Contribution

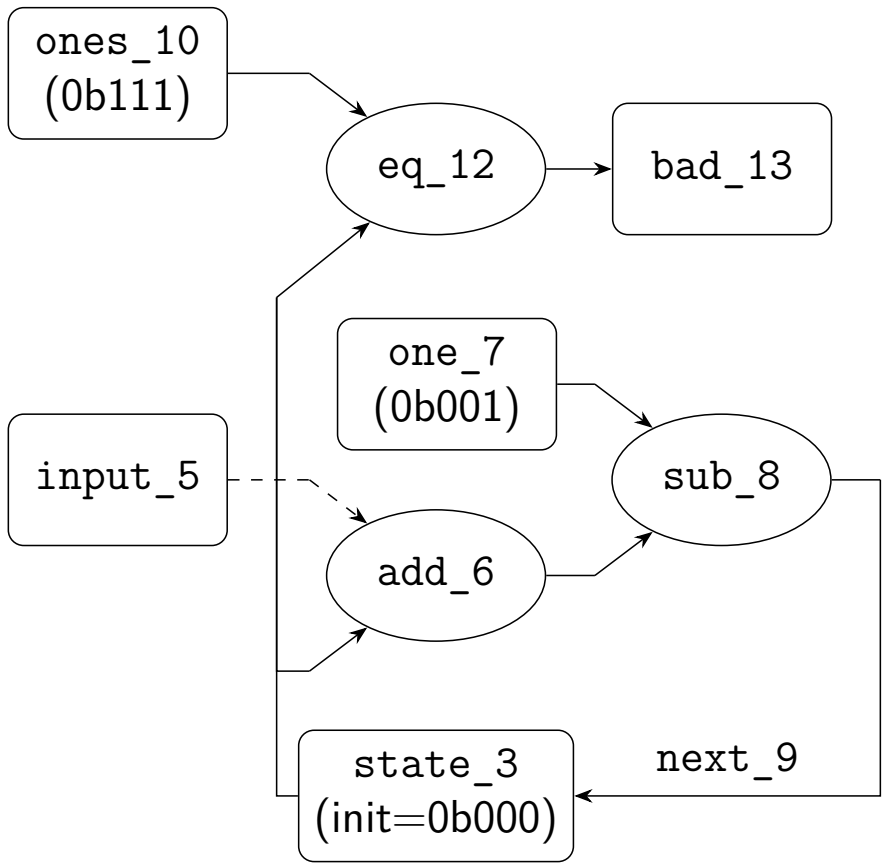


Our Contribution



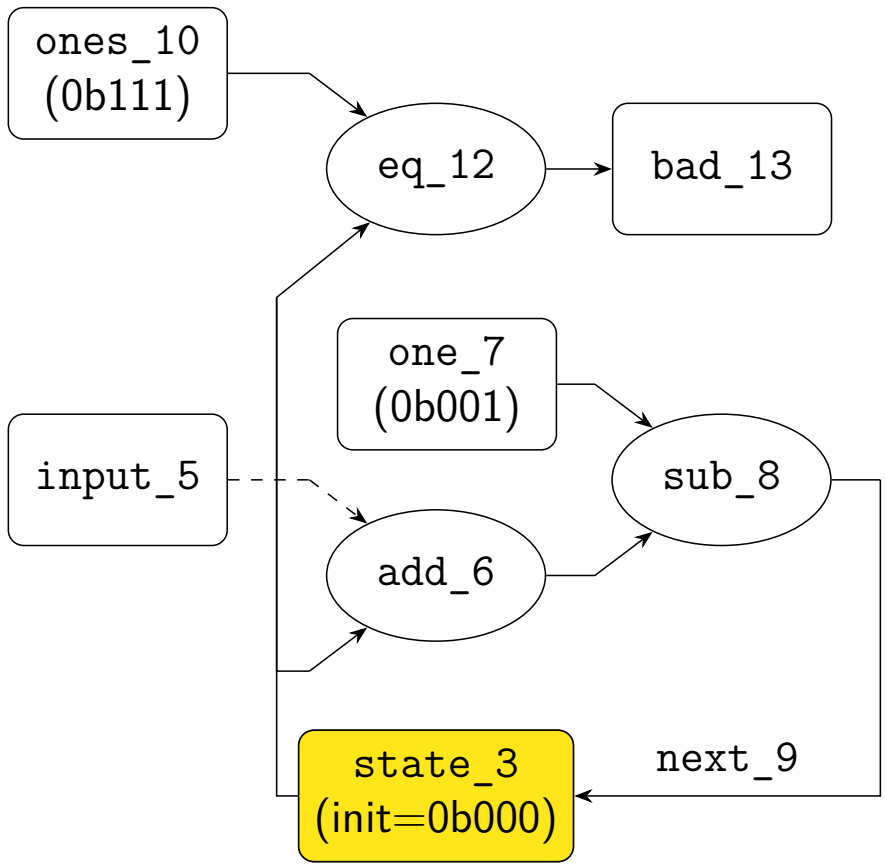
The BTOR2 Language

```
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
```



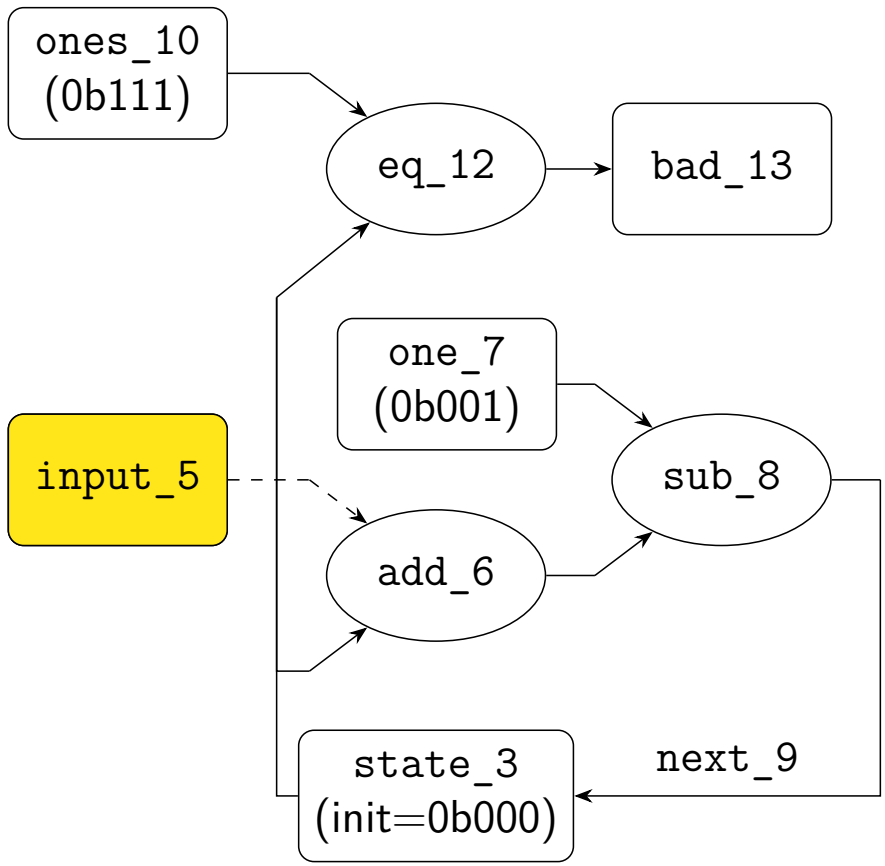
The BTOR2 Language

```
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
```



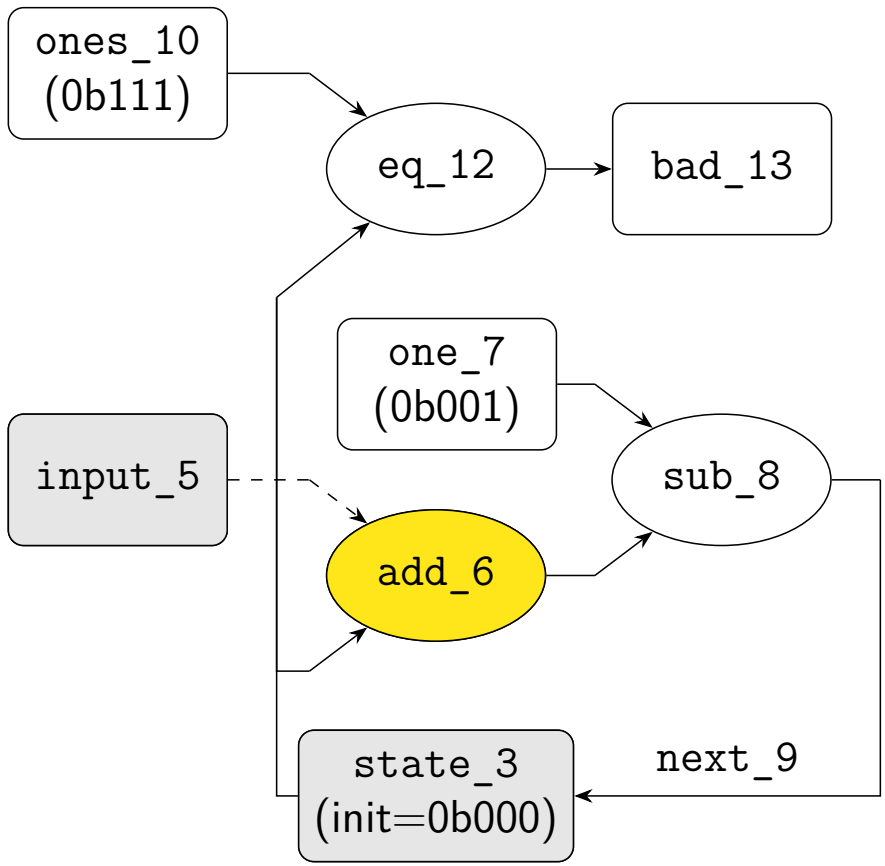
The BTOR2 Language

```
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
```



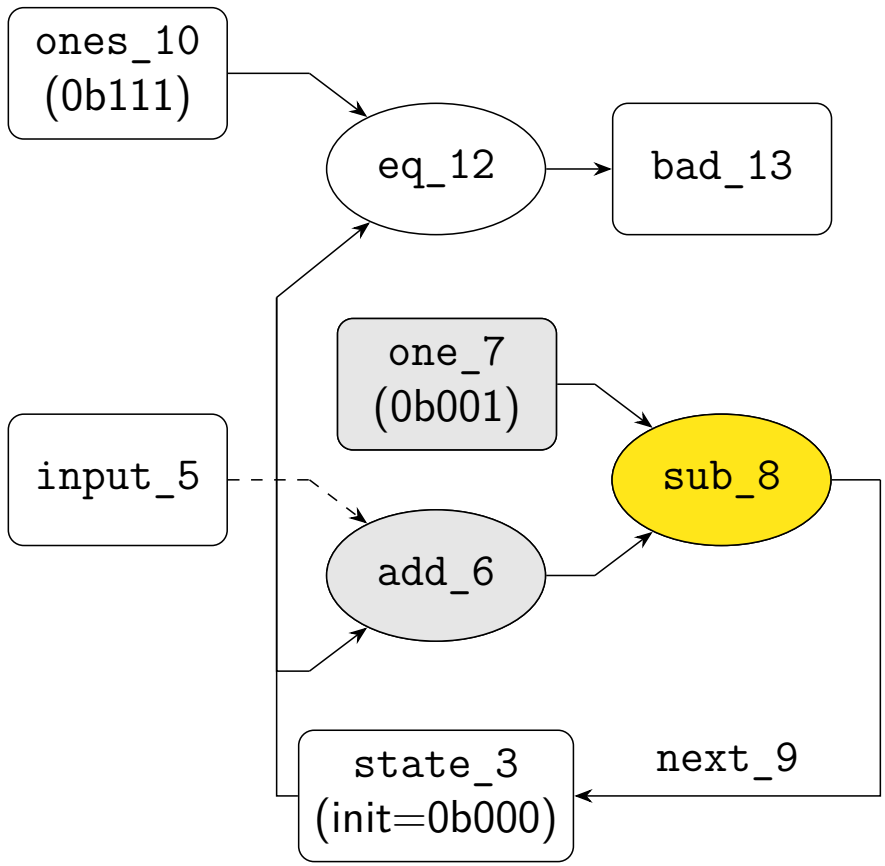
The BTOR2 Language

```
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
```



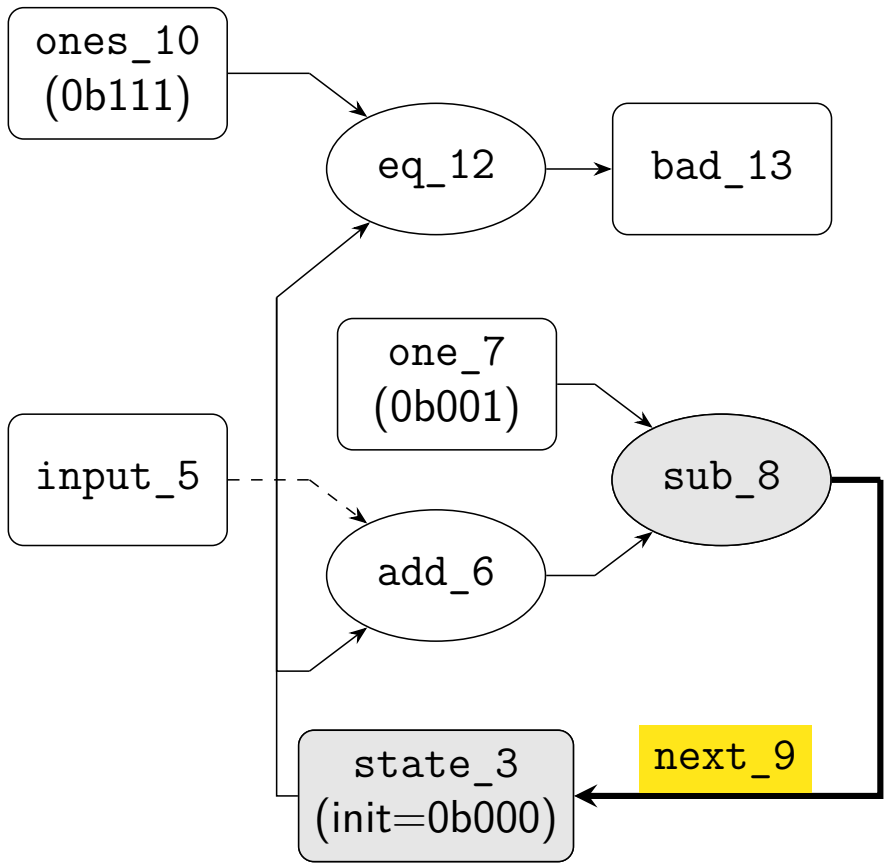
The BTOR2 Language

```
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
```



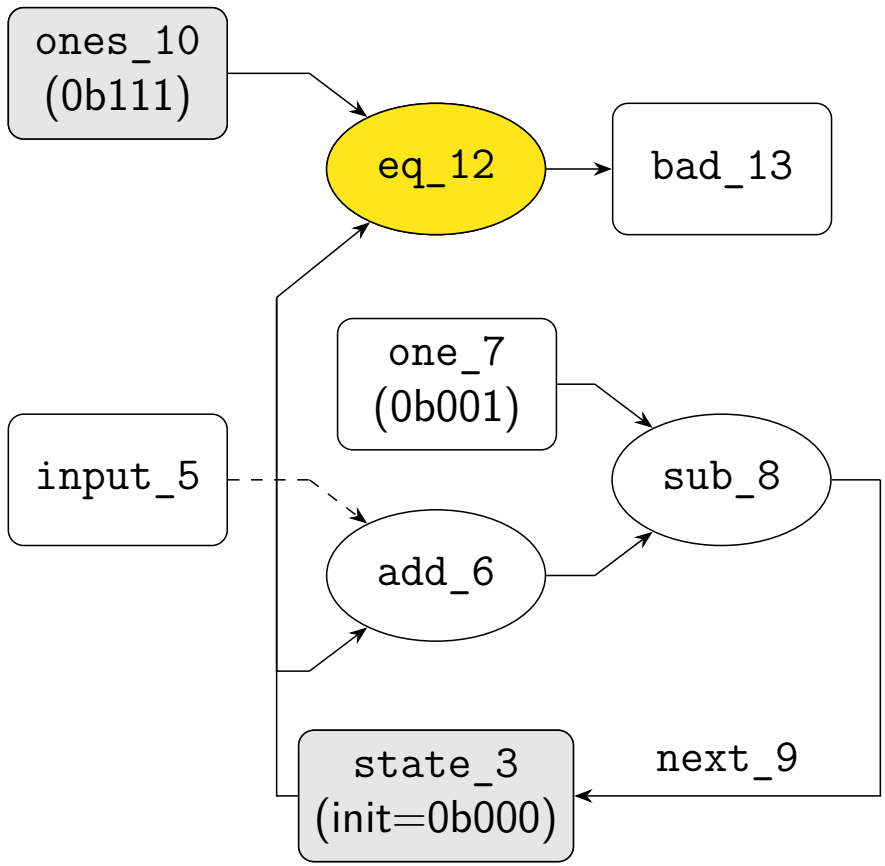
The BTOR2 Language

```
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
```



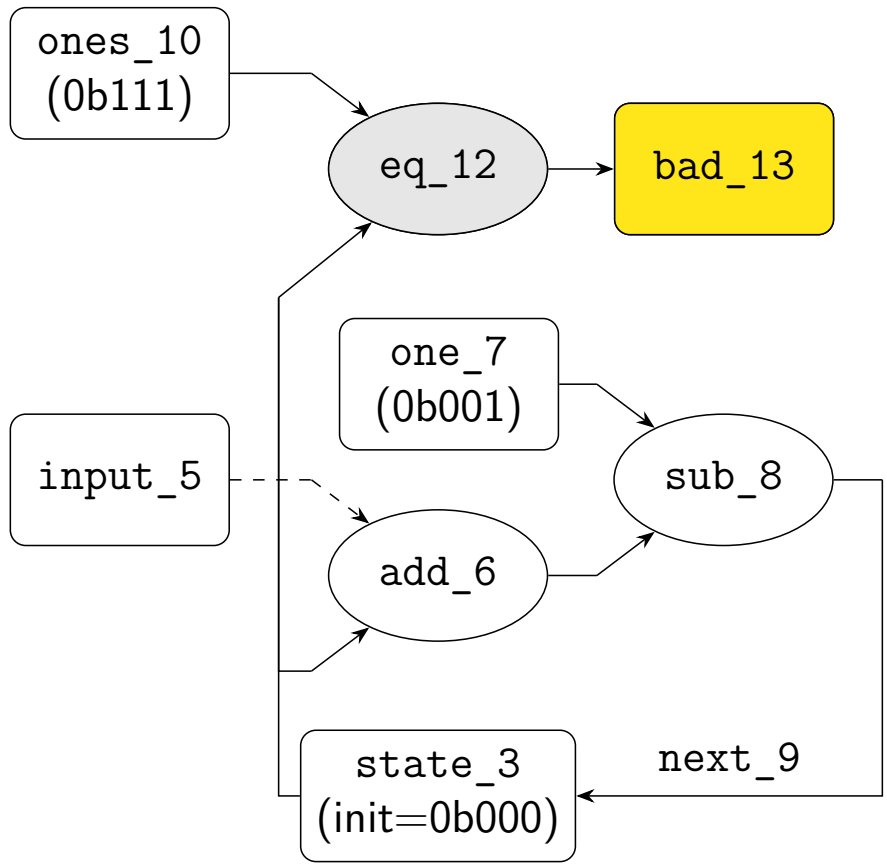
The BTOR2 Language

```
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 }
```

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 }
```

Btor2C: Btor2-to-C Translator

- A lightweight tool
  - Written in C++ with $\sim$2 K LOC
  - Use the frontend parser provided by Btor2Tools [17]
- Open-source under Apache License 2.0:
  https://gitlab.com/sosy-lab/software/btor2c

Evaluation

On $1\,008$ safe and 490 unsafe BTOR2 verification tasks:

Evaluation

On $1\,008$ safe and 490 unsafe BTOR2 verification tasks:

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

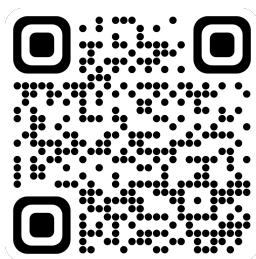
Evaluation

On $1\,008$ 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*

Summary

- Btor2C: Btor2 circuits to C programs
  - Applying off-the-shelf software analyzers to hardware
  - Improving quality assurance for hardware
- **43** tasks uniquely solved by software verifiers
- The reproduction artifact [23] is available via Zenodo.



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

Zsófia Ádám[1,2], Dirk Beyer[2], Po-Chun Chien[2],
Nian-Ze Lee[2], and Nils Sirrenberg[2]

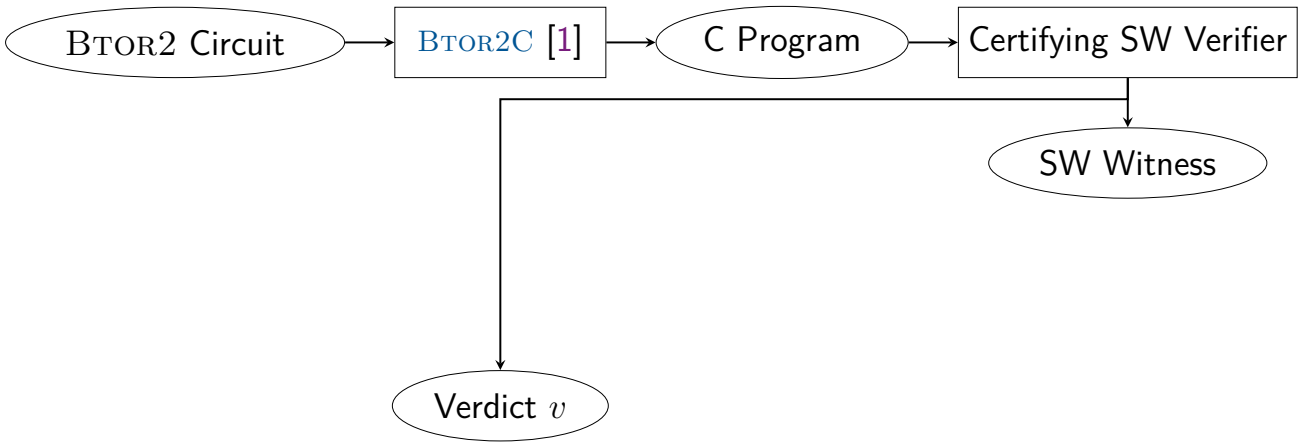
[1]Budapest University of Technology and Economics, Hungary
[2]LMU Munich, Germany

TACAS 2024-04-11



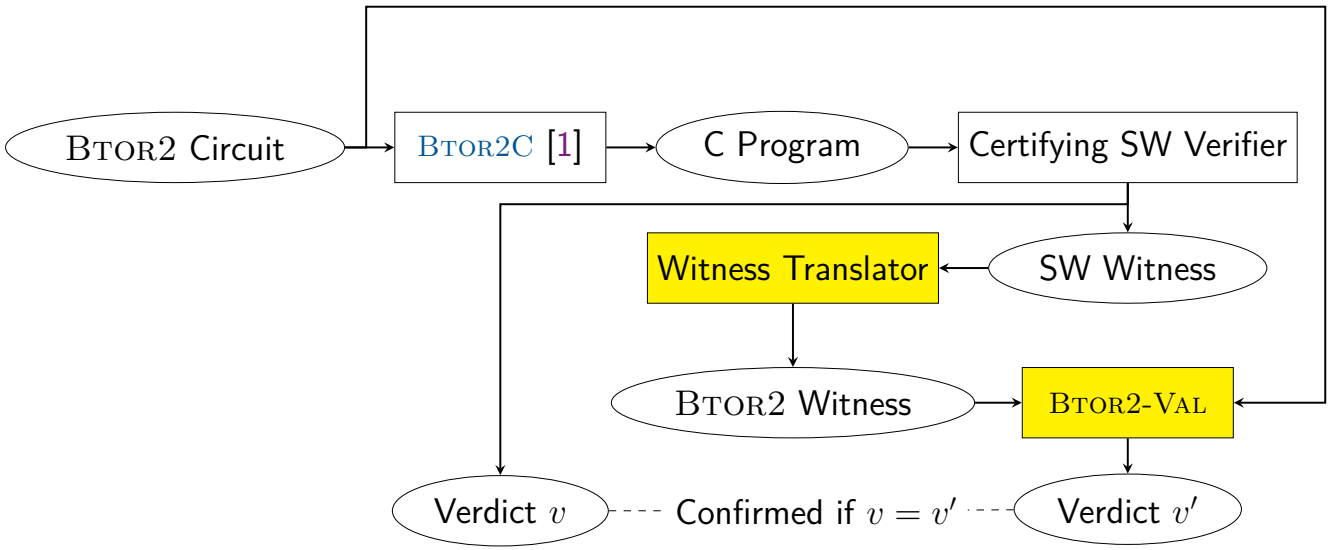


Certifying Verification for BTOR2 with SV Tools



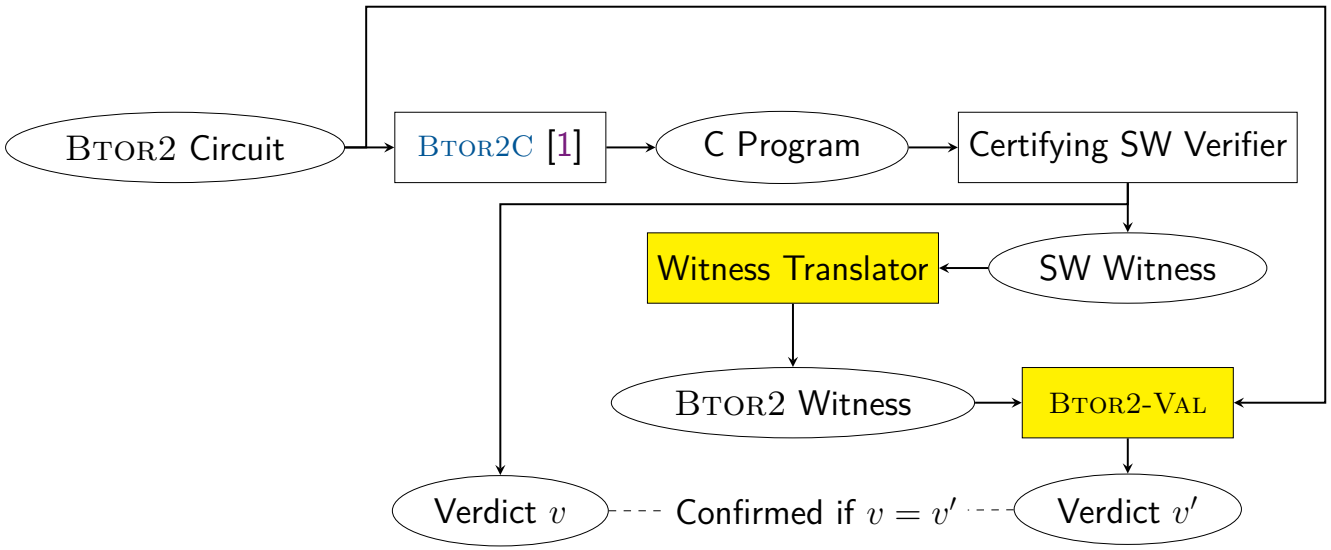
- BTOR2 [5] word-level circuits and translator BTOR2C [1]
- Software verifiers in SV-COMP [24]

Certifying Verification for BTOR2 with SV Tools



- BTOR2 [5] word-level circuits and translator BTOR2C [1]
- Software verifiers in SV-COMP [24]
- SW-to-HW witness translation and BTOR2-VAL

Certifying Verification for BTOR2 with SV Tools



- BTOR2 [5] word-level circuits and translator BTOR2C [1]
- Software verifiers in SV-COMP [24]
- SW-to-HW witness translation and BTOR2-VAL
- On 1214 BTOR2 circuits, BTOR2-CERT
  - found missed bugs by CBMC [25] to complement ABC [13]
  - derived invariants by CPACHECKER [7] to accelerate ABC

Agenda

1. Cross-application of HW and SW formal verification
1.1 Applying SW analyzers to HW verification tasks
1.2 Applying HW model checkers to SW verification tasks

2. Knowledge consolidation of formal verification
2.1 Transferability of HW algorithms to SW verification

3. Reflection and outlook

CPV: A Circuit-Based Program Verifier

Po-Chun Chien and Nian-Ze Lee

LMU Munich, Germany

First-Time Participant at SV-COMP 2024 (April 8)

Research Question

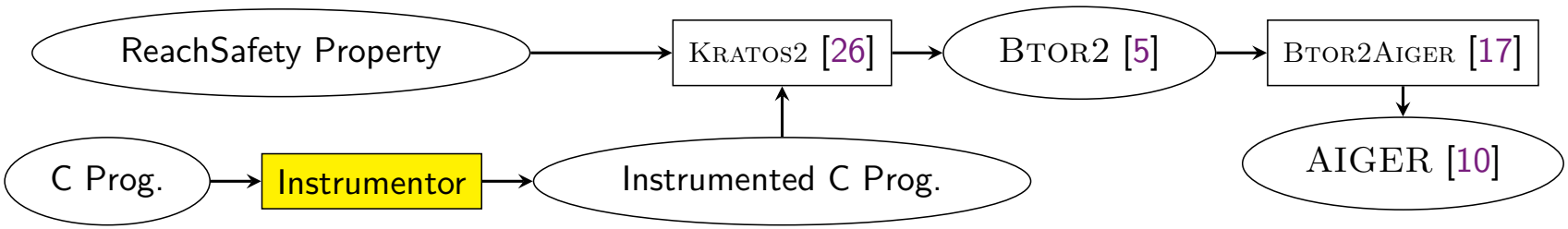
- Sequential circuits as an alternative intermediate representation for program analysis
  - Leveraging HW model checking as backend

Research Question

- Sequential circuits as an alternative intermediate representation for program analysis
  - Leveraging HW model checking as backend

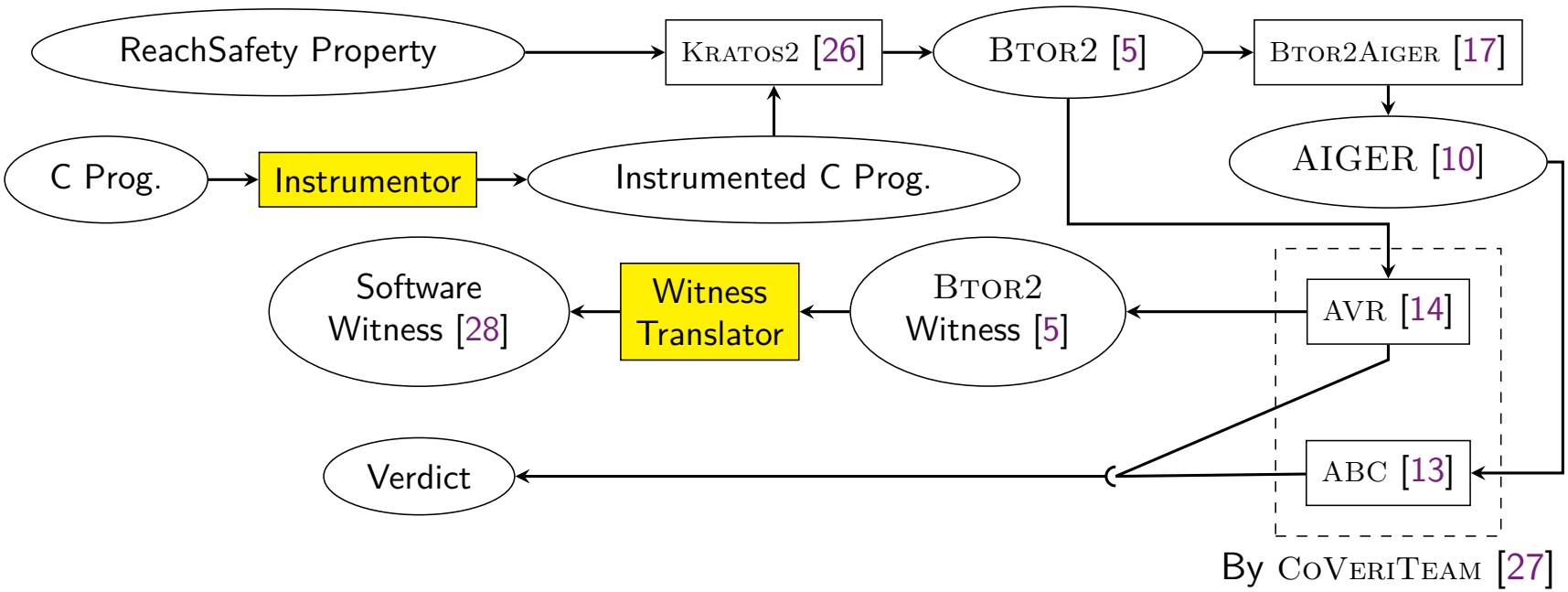
CPV ranks 6$^{\text{th}}$ out of 26 in *ReachSafety* as a first-time participant in SV-COMP 2024!

Software Architecture of CPV



- Program instrumentation for retrieving witness information

Software Architecture of CPV



- Program instrumentation for retrieving witness information
- HW-to-SW witness translation

Agenda

1. Cross-application of HW and SW formal verification
1.1 Applying SW analyzers to HW verification tasks
1.2 Applying HW model checkers to SW verification tasks

2. Knowledge consolidation of formal verification
2.1 Transferability of HW algorithms to SW verification

3. Reflection and outlook

Agenda

1. Cross-application of HW and SW formal verification
1.1 Applying SW analyzers to HW verification tasks
1.2 Applying HW model checkers to SW verification tasks

2. Knowledge consolidation of formal verification
2.1 Transferability of HW algorithms to SW verification

3. Reflection and outlook

A Transferability Study of Interpolation-Based Hardware Model Checking to Software Verification

Dirk Beyer, Po-Chun Chien, Marek Jankola, and Nian-Ze Lee

LMU Munich, Germany




Research Question

- Can research conclusions from hardware model checking be transferred to software verification?

Research Question

- Can research conclusions from hardware model checking be transferred to software verification?

Characteristics of two interpolation-based algorithms [30, 31] for hardware are shown to be transferrable to software.

Transferring HV Studies to SV Contexts

- Implement two hardware-verification algorithms in CPAchecker and evaluate on ∼9 K software tasks
  - Baseline: *interpolation-based model checking* (IMC) [32] (software adoption to appear in *JAR* [4])

Transferring HV Studies to SV Contexts

- Implement two hardware-verification algorithms in CPAchecker and evaluate on ∼9 K software tasks
  - Baseline: *interpolation-based model checking* (IMC) [32] (software adoption to appear in *JAR* [4])

- Interpolation-Sequence-Based Model Checking, 2009 [30]
  - Benchmark set: 136 tasks derived from industrial designs
  - Claims
    - Faster than IMC on unsafe tasks (✓)
    - Faster than IMC on safe tasks if higher unrolling (?)
    - Overall, faster than IMC (by 30 %) (?)

Transferring HV Studies to SV Contexts

- Implement two hardware-verification algorithms in CPAchecker and evaluate on ∼9 K software tasks
  - Baseline: *interpolation-based model checking* (IMC) [32] (software adoption to appear in *JAR* [4])

- Interpolation-Sequence-Based Model Checking, 2009 [30]
- Intertwined Forward-Backward Reachability Analysis Using Interpolants, 2013 [31]
  - Benchmark set: 37 tasks derived from industrial designs
  - Claims
    - Local strengthening enough to refute spurious bugs (✓)
    - Faster than IMC on safe tasks (?)
    - Computes more interpolants than IMC (✓)
    - More sensitive to the sizes of interpolants (?)
    - Overall, faster than IMC (by 36 %) (?)

Summary

- Transferability of algorithmic characteristics confirmed
  - Interpolation-sequence faster to find bugs
  - Local strengthening enough to refute spurious bugs
- Unifying knowledge across hardware and software for full-stack system verification

Agenda

1. Cross-application of HW and SW formal verification
1.1 Applying SW analyzers to HW verification tasks
1.2 Applying HW model checkers to SW verification tasks

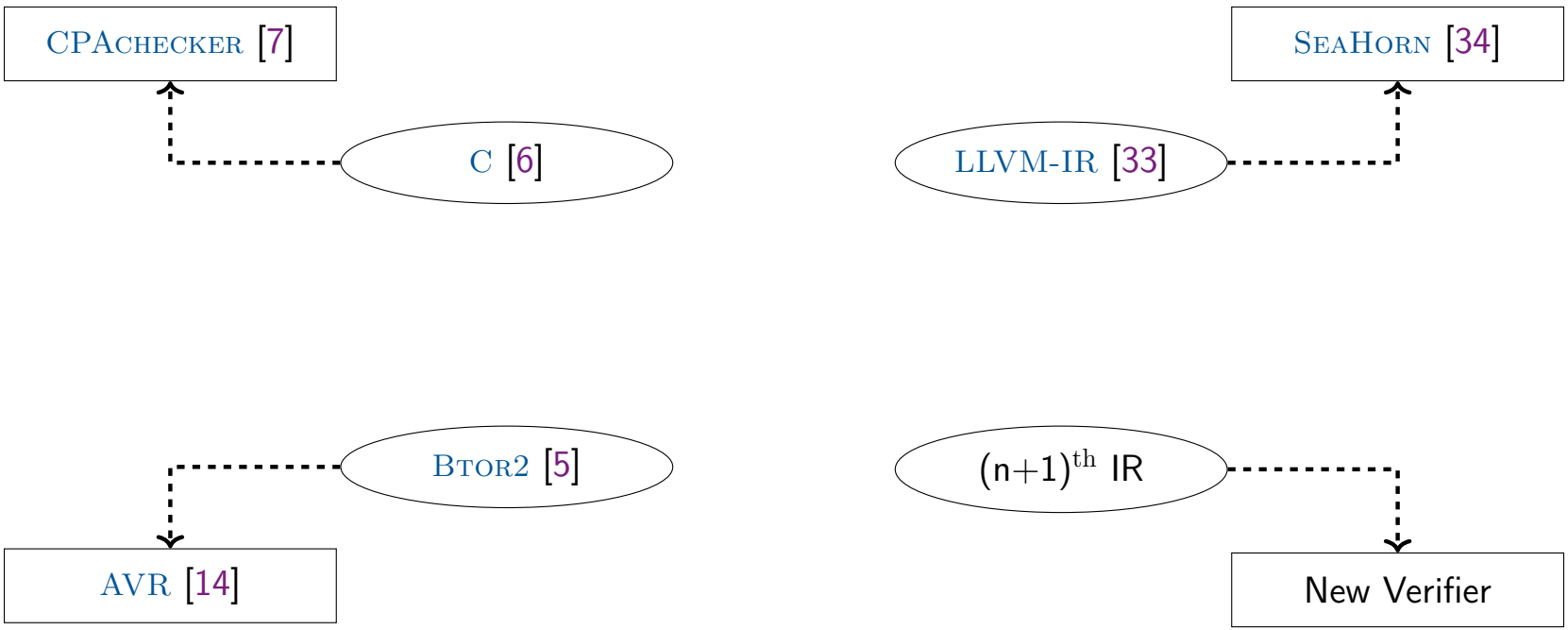
2. Knowledge consolidation of formal verification
2.1 Transferability of HW algorithms to SW verification

3. Reflection and outlook

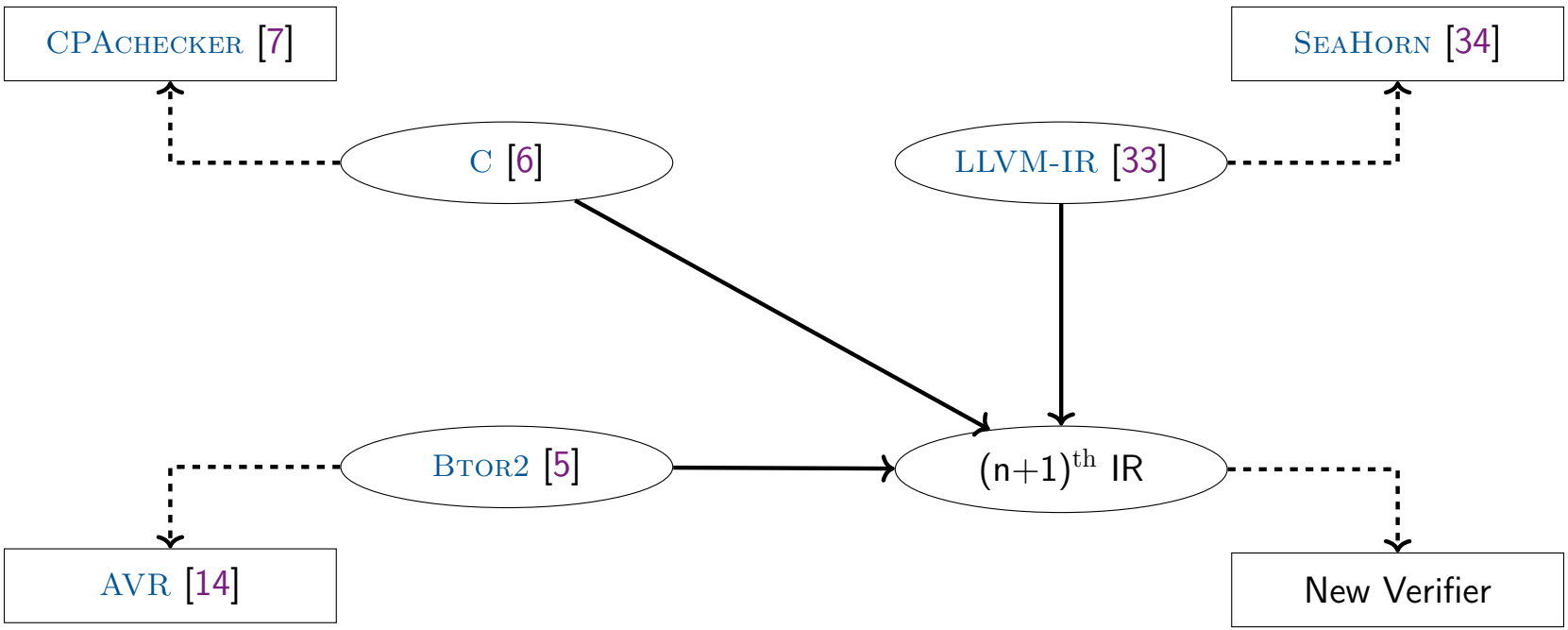
Bridging HW and SW Analysis: So Far

- SW tools uniquely solving HW tasks [1, 2]
- HV algorithms [4] and tools [3] improving SV
- Transferability of algorithmic characteristics [29]

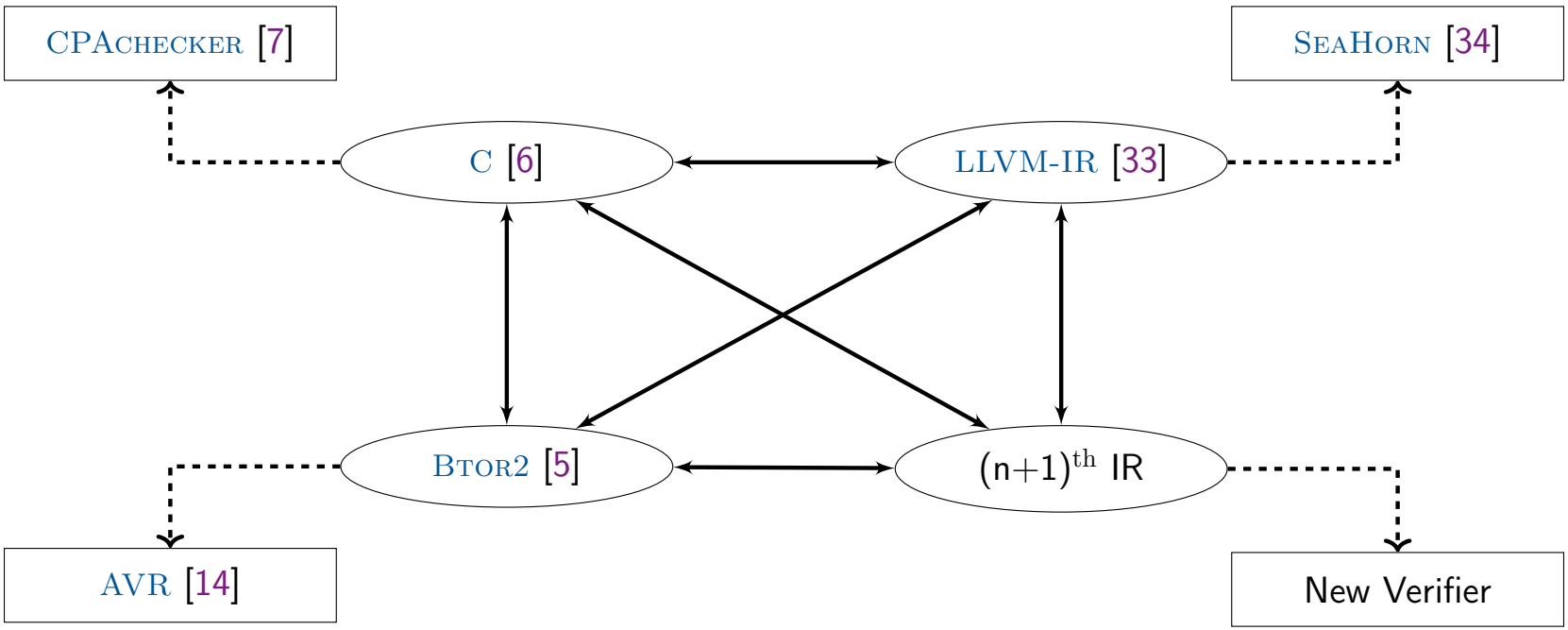
Critical Reflection



Critical Reflection



Critical Reflection



Transformation between different representations to leverage their unique strengths

Challenges of Heterogeneous Systems

- Firmware
  - Ex: data privacy of smart devices
  - Specific constructs (e.g., assembly), hardware modeling
  - Current practices: manual review or testing
  - Idea: circuit-based program verification

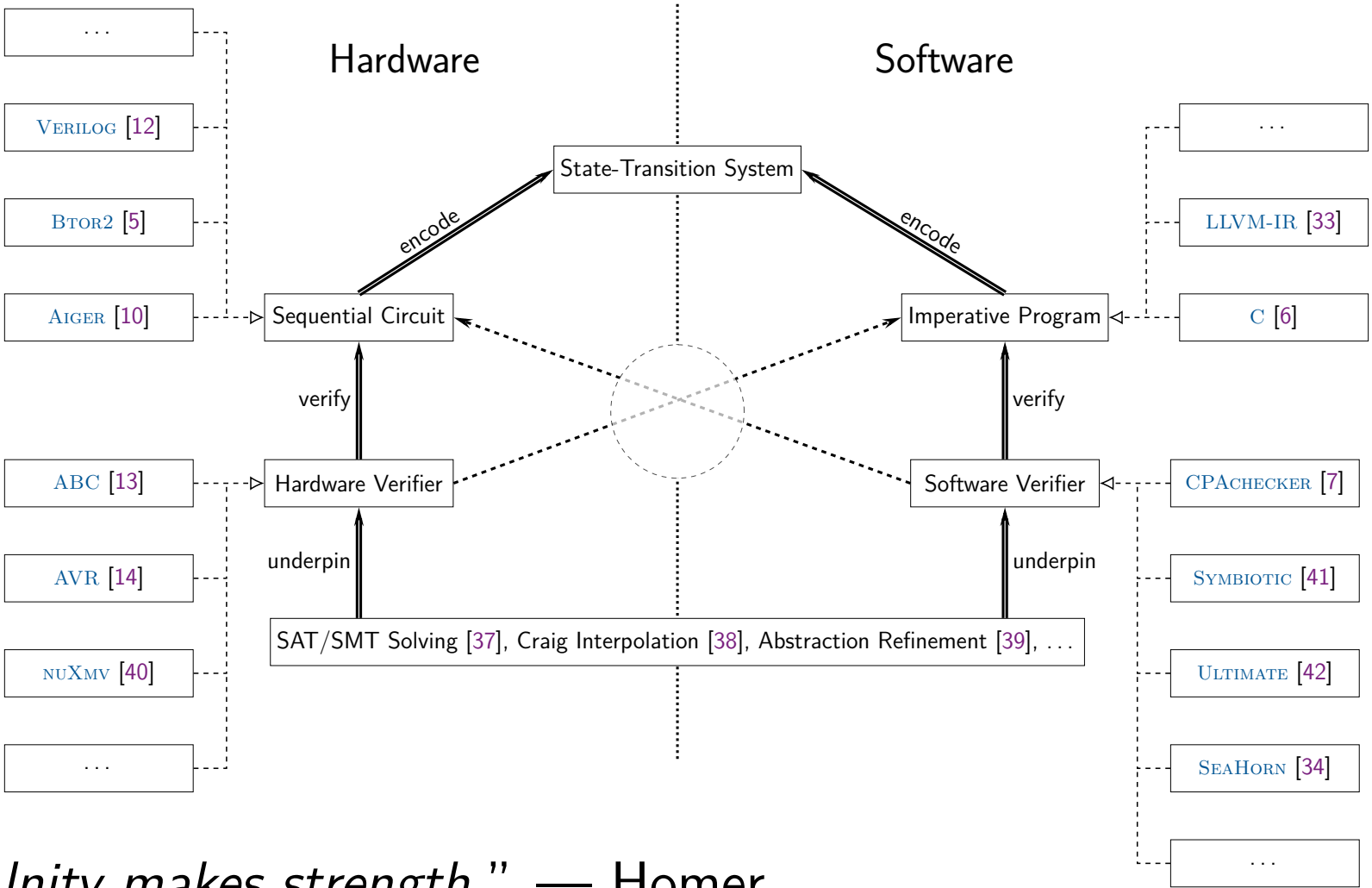
Challenges of Heterogeneous Systems

- Firmware
  - Ex: data privacy of smart devices
  - Specific constructs (e.g., assembly), hardware modeling
  - Current practices: manual review or testing
  - Idea: circuit-based program verification
- Neural networks
  - Ex: autonomous driving
  - Mathematical operations
  - Current practices: mostly on models (cf. VNN-COMP)
    - Less on C implementations (e.g., NeuroCodeBench [35])
  - Idea: symbolic computer algebra

Challenges of Heterogeneous Systems

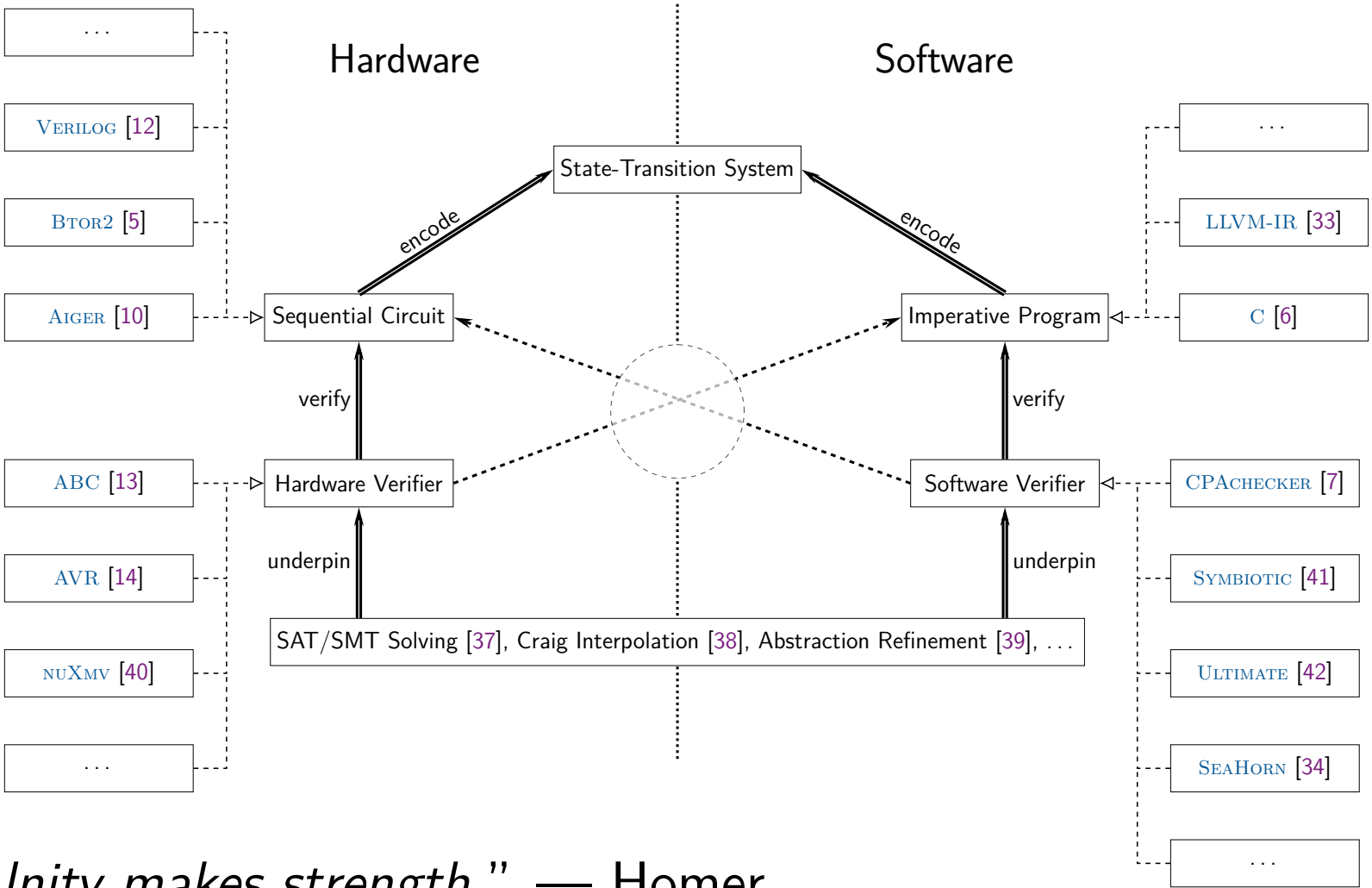
- Firmware
  - Ex: data privacy of smart devices
  - Specific constructs (e.g., assembly), hardware modeling
  - Current practices: manual review or testing
  - Idea: circuit-based program verification
- Neural networks
  - Ex: autonomous driving
  - Mathematical operations
  - Current practices: mostly on models (cf. VNN-COMP)
    - Less on C implementations (e.g., NeuroCodeBench [35])
  - Idea: symbolic computer algebra
- Hardware/software co-design (embedded systems)
  - Current practices: lower level verification conditions [36]
  - Idea: on-the-fly translation and abstraction as HW or SW

Closing the Gap between HW and SW Analysis



"*Unity makes strength.*" — Homer

Closing the Gap between HW and SW Analysis



"*Unity makes strength.*" — Homer

We are hiring! (A full Ph.D. position funded by DFG)

References I

[1] Beyer, D., Chien, P.C., Lee, N.Z.: Bridging hardware and software analysis with Btor2C: A word-level-circuit-to-C translator. In: Proc. TACAS. pp. 1–21. LNCS 13994, Springer (2023). doi:10.1007/978-3-031-30820-8_12

[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. Springer (2024)

[3] Chien, P.C., Lee, N.Z.: CPV: A circuit-based program verifier (competition contribution). In: Proc. TACAS. LNCS , Springer (2024)

[4] 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

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

[6] 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 II

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

[8] 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). doi:10.1145/3238147.3240481

[9] Alshmrany, K.M., Aldughaim, M., Bhayat, A., Cordeiro, L.C.: FuSeBMC: An energy-efficient test generator for finding security vulnerabilities in C programs. In: Proc. TAP. pp. 85–105. Springer (2021). doi:10.1007/978-3-030-79379-1_6

[10] 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). doi:10.35011/fmvtr.2007-1

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

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

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

References III

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

[15] Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS (2). pp. 375–402. LNCS 13244, Springer (2022). doi:10.1007/978-3-030-99527-0_20

[16] Beyer, D.: Advances in automatic software testing: Test-Comp 2022. In: Proc. FASE. pp. 321–335. LNCS 13241, Springer (2022). doi:10.1007/978-3-030-99429-7_18

[17] 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

[18] Afzal, M., Asia, A., Chauhan, A., Chimdyalwar, B., Darke, P., Datar, A., Kumar, S., Venkatesh, R.: VeriAbs: Verification by abstraction and test generation. In: Proc. ASE. pp. 1138–1141. IEEE (2019). doi:10.1109/ASE.2019.00121

[19] Biere, A., van Dijk, T., Heljanko, K.: Hardware model checking competition 2017. In: Proc. FMCAD. p. 9. IEEE (2017). doi:10.23919/FMCAD.2017.8102233

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

References IV

[21] Dutertre, B.: Yices 2.2. In: Proc. CAV. pp. 737–744. LNCS 8559, Springer (2014). doi:10.1007/978-3-319-08867-9_49

[22] Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Proc. TACAS. pp. 93–107. LNCS 7795, Springer (2013). doi:10.1007/978-3-642-36742-7_7

[23] Beyer, D., Chien, P.C., Lee, N.Z.: Reproduction package for TACAS 2023 article 'Bridging hardware and software analysis with Btor2C: A word-level-circuit-to-C translator'. Zenodo (2023). doi:10.5281/zenodo.7551707

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

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

[26] Griggio, A., Jonáš, M.: Kratos2: An SMT-based model checker for imperative programs. In: Proc. CAV. pp. 423–436. Springer (2023). doi:10.1007/978-3-031-37709-9_20

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

References V

[28] 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). doi:10.1145/3477579

[29] Beyer, D., Chien, P.C., Jankola, M., Lee, N.Z.: A transferability study of interpolation-based hardware model checking to software verification (2024), conditionally accepted (major revision) at the ACM International Conference on the Foundations of Software Engineering

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

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

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

[33] Lattner, C., Adve, V.S.: LLVM: A compilation framework for lifelong program analysis and transformation. In: Proc. CGO. pp. 75–88. IEEE (2004). doi:10.1109/CGO.2004.1281665

References VI

[34] Gurfinkel, A., Kahsai, T., Komuravelli, A., Navas, J.A.: The SeaHorn verification framework. In: Proc. CAV. pp. 343–361. LNCS 9206, Springer (2015). doi:10.1007/978-3-319-21690-4_20

[35] Manino, E., Menezes, R.S., Shmarov, F., Cordeiro, L.C.: NeuroCodeBench: a plain C neural network benchmark for software verification (arXiv:2309.03617) (2023). doi:10.48550/arXiv.2309.03617

[36] Mukherjee, R., Purandare, M., Polig, R., Kroening, D.: Formal techniques for effective co-verification of hardware/software co-designs. In: Proc. DAC. pp. 1–6. ACM (2017). doi:10.1145/3061639.3062253

[37] Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)

[38] Craig, W.: Linear reasoning. A new form of the Herbrand-Gentzen theorem. J. Symb. Log. **22**(3), 250–268 (1957). doi:10.2307/2963593

[39] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Proc. CAV. pp. 154–169. LNCS 1855, Springer (2000). doi:10.1007/10722167_15

References VII

[40] Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The NUXMV symbolic model checker. In: Proc. CAV. pp. 334–342. LNCS 8559, Springer (2014). doi:10.1007/978-3-319-08867-9_22

[41] Chalupa, M., Strejček, J., Vitovská, M.: Joint forces for memory safety checking. In: Proc. SPIN. pp. 115–132. Springer (2018). doi:10.1007/978-3-319-94111-0_7

[42] Heizmann, M., Hoenicke, J., Podelski, A.: Software model checking for people who love automata. In: Proc. CAV. pp. 36–52. LNCS 8044, Springer (2013). doi:10.1007/978-3-642-39799-8_2