Cross-Application of Hardware and Software Verification

Po-Chun Chien
SoSy-Lab, LMU Munich
ConVeY Spring Workshop
2024-04-06 @ Luxembourg
Motivation

Sequential Circuit

Hardware Verifier

Hardware

State-Transition System

Imperative Program

Software Verifier

Software

encode

encode

verify

verify

underpin

underpin

SAT/SMT Solving, Craig Interpolation, Abstraction Refinement, Symbolic Representation, ...
Motivation

SoSy-Lab, LMU Munich

Cross-Application of HW/SW 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

Btor2 circuit [31]
(HWMCC [14])

C program [26]
(SV-COMP [3])

Hardware model checkers
ABC [17], AVR [24], ...

Software verifiers
CPAchecker [10], Esbmc [22], ...

SoSy-Lab, LMU Munich
Cross-Application of HW/SW Verification
Translating Verification Tasks

Btor2 circuit [31] (HWMCC [14])

C program [26] (SV-COMP [3])

Hardware model checkers
- ABC [17], AVR [24], ...

Software verifiers
- CPAchecker [10], Esbmc [22], ...

Translate
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
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
HW-Analysis Tool Chain with Btor2C

Our Contribution


HW-Analysis Tool Chain with Btor2C

Our Contribution

- Btor2C [4]
- C [26]
- SW Analyzer
  - ESBMC [22]
  - CPAchecker [10]
  - CBMC [19]

Word-Level HW Analyzer
- Btor2 [31]
- Btor2AIGER [31]
- Yosys [35]
- Verilog [1]
- AIGER [13]
- Bit-Level HW Analyzer
  - ABC [17]

Bit-Level HW Analyzer
- Btor2C [4]

SoSy-Lab, LMU Munich
Cross-Application of HW/SW Verification
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

SoSy-Lab, LMU Munich Cross-Application of HW/SW Verification
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

SoSy-Lab, LMU Munich
Cross-Application of HW/SW Verification
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
void main() {
  typedef unsigned char SORT_1;
  typedef unsigned char SORT_11;
  const SORT_1 var_2 = 0b000;
  const SORT_1 var_7 = 0b001;
  const SORT_1 var_10 = 0b111;
  SORT_1 state_3 = var_2;
  for (; ;) {
    SORT_1 input_5 = nondet_uchar();
    input_5 = input_5 & 0b111;
    SORT_11 var_12 = state_3 == var_10;
    SORT_11 bad_13 = var_12;
    if (bad_13) { ERROR: abort(); }
    SORT_1 var_6 = state_3 + input_5;
    var_6 = var_6 & 0b111;
    SORT_1 var_8 = var_6 - var_7;
    var_8 = var_8 & 0b111;
    state_3 = var_8;
  }
}
Translating Btor2 to C

```c
void main() {
    typedef unsigned char SORT_1;
    typedef unsigned char SORT_11;
    const SORT_1 var_2 = 0b000;
    const SORT_1 var_7 = 0b001;
    const SORT_1 var_10 = 0b111;
    SORT_1 state_3 = var_2;
    for (;;) {
        SORT_1 input_5 = nondet_uchar();
        input_5 = input_5 & 0b111;
        SORT_11 var_12 = state_3 == var_10;
        SORT_11 bad_13 = var_12;
        if (bad_13) { ERROR: abort(); }
        SORT_1 var_6 = state_3 + input_5;
        var_6 = var_6 & 0b111;
        SORT_1 var_8 = var_6 - var_7;
        var_8 = var_8 & 0b111;
        state_3 = var_8;
    }
}
```
void main() {
    typedef unsigned char SORT_1;
    typedef unsigned char SORT_11;
    const SORT_1 var_2 = 0b000;
    const SORT_1 var_7 = 0b001;
    const SORT_1 var_10 = 0b111;
    SORT_1 state_3 = var_2;
    for (;;) {
        SORT_1 input_5 = nondet_uchar();
        input_5 = input_5 & 0b111;
        SORT_11 var_12 = state_3 == var_10;
        SORT_11 bad_13 = var_12;
        if (bad_13) { ERROR: abort(); }
        SORT_1 var_6 = input_5 = state_3 + input_5;
        var_6 = var_6 & 0b111;
        SORT_1 var_8 = var_6 − var_7;
        var_8 = var_8 & 0b111;
        state_3 = var_8;
    }
}
void main() {
    typedef unsigned char SORT_1;
    typedef unsigned char SORT_11;
    const SORT_1 var_2 = 0b000;
    const SORT_1 var_7 = 0b001;
    const SORT_1 var_10 = 0b111;
    SORT_1 state_3 = var_2;
    for (;;) {
        SORT_1 input_5 = nondet_uchar();
        input_5 = input_5 & 0b111;
        SORT_11 var_12 = state_3 == var_10;
        SORT_11 bad_13 = var_12;
        if (bad_13) {
            ERROR: abort();
        }
        SORT_1 var_6 = state_3 + input_5;
        var_6 = var_6 & 0b111;
        SORT_1 var_8 = var_6 - var_7;
        var_8 = var_8 & 0b111;
        state_3 = var_8;
    }
}
On 1008 safe and 490 unsafe BTOR2 verification tasks:
Evaluation of SW Analyzers on HW Tasks

On 1008 safe and 490 unsafe B\texttt{TOR}2 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 $\texttt{Btor2}$ 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

2. SW Witness
3. Verdict $v$

- Btor2-Val: witness validator for Btor2
- Correctness: circuit instrumentation and validation via verification
- Violation: execution-based validation via BtorSim

Cross-Application of HW/SW Verification
- Translate verification witness: automata [7] to \texttt{Btor2} [31]

\textbf{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
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

---

https://www.sosy-lab.org/research/btor2-cert/
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

https://www.sosy-lab.org/research/btor2-cert/
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\(^1\)

\(^1\url{https://www.sosy-lab.org/research/btor2-cert/}\)
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
Utilizing 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

[Link to CPV repository on GitLab]
System Architecture

- ReachSafety property
  - C prog.
    - Instrumentor
    - Instrumented C prog.
    - Software witness [7]
    - Witness translator
    - BTOR2 [31]
    - BTOR2 witness [31]
    - AVR [24]
    - ABC [17]
  - Kratos2 [25]
  - BTOR2Aiger [30]
  - AIGER [13]
  - Verdict

By CoVeriTeam [9]
1. Instrument the input program
2. Translate the program to a circuit
3. Verify the translated circuit with hardware model checkers
4. Translate the \texttt{BTOR2} witness back to software domain
Results in SV-COMP

6th place in ReachSafety
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

Sequential circuit

HW model checking
- BMC [15],
- $k$-induction [32],
- IC3/PDR [16],
- ...

Imperative program

SW analysis
- fuzz testing [28],
- symbolic execution [27],
- abstract interpretation [20],
- ...

SoSy-Lab, LMU Munich
Cross-Application of HW/SW Verification
Transferring Verification Techniques Across Domains

Sequential circuit

- HW model checking
  - BMC [15],
  - $k$-induction [32],
  - IC3/PDR [16],
  - ...

Imperative program

- SW analysis
  - fuzz testing [28],
  - symbolic execution [27],
  - abstract interpretation [20],
  - ...

Transfer
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 Verification

- 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]

Our study shows:
- Characteristics of these algorithms transferrable
- These HW algorithms can tackle tasks unsolvable by existing methods
  → cross-disciplinary adoption is beneficial
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

IMC [29] ↔ Date-flow analysis [5]

effective but expensive  auxiliary invariants  efficient but imprecise
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
Try our tools!

- **Btor2-Cert** [2]
- **CPV** [18]
- **CPAchecker** [10]
Try our tools!

- **Btor2-Cert** [2]
- **CPV** [18]
- **CPAchecker** [10]

Join our talks at ETAPS!

- **COOP**: Sun. 10:30 (more on our work)
- **SV-COMP**: Mon. 14:00 (**CPV**)
- **SPIN**: Wed. 11:30 (**IMC** + inv.)
- **TACAS**: Thu. 12:00 (**Btor2-Cert**)

SoSy-Lab, LMU Munich  
Cross-Application of HW/SW Verification
References


References


References


