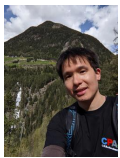
Exploring the Interplay of Hardware and Software Analysis

Nian-Ze Lee

EDA Group, GIEE, National Taiwan University

2024-09-27 @ EDA Group Information Session



About Me

- 2014: B.Sc. in Electrical Engineering, NTU
- 2021: Ph.D. in Electronics Engineering, NTU
- 2021-2024: PostDoc in Computer Science, LMU Munich, Germany
- From 2025: Assistant Professor, EDA Group, NTU

About Me

- 2014: B.Sc. in Electrical Engineering, NTU
- 2021: Ph.D. in Electronics Engineering, NTU
- 2021-2024: PostDoc in Computer Science, LMU Munich, Germany
- From 2025: Assistant Professor, EDA Group, NTU

Research interests: formal methods and verification for computational models (hardware, software, and more)

Formal Verification in a Nutshell

- Does a **computational model** satisfy a **specification**?
  - Model: control-flow automata (software), sequential circuits (hardware), etc.
  - Specification: temporal-logic formulas

Formal Verification in a Nutshell

- Does a **computational model** satisfy a **specification**?
  - Model: control-flow automata (software), sequential circuits (hardware), etc.
  - Specification: temporal-logic formulas
  - Mathematical rigor: prove absence of bugs (cf. testing)
  - Undecidable in theory

Facebook's Software Verifier Infer [1]

```
class Infer {

    String mayReturnNull(int i) {
        if (i > 0) {
            return "Hello, Infer!";
        }
        return null;
    }

    int mayCauseNPE() {
        String s = mayReturnNull(0);
        return s.length();
    }

}
```

Facebook's Software Verifier Infer [1]

```java
class Infer {

    String mayReturnNull(int i) {
        if (i > 0) {
            return "Hello, Infer!";
        }
        return null;
    }

    int mayCauseNPE() {
        String s = mayReturnNull(0);
        return s.length();
    }

}
```

```
Found 1 issue
Infer.java:12: error: NULL_DEREFERENCE
  object s last assigned on line 11 could be null and is dereferenced at line 12
  10.       int mayCauseNPE() {
  11.           String s = mayReturnNull(0);
  12. >         return s.length();
  13.       }
```

*Screenshots from https://fbinfer.com

Overview of Software Verification

- Automatically detect issues in source code
  - Null-pointer dereference
  - Assertion violation
  - Memory leak
  - ...

Overview of Software Verification

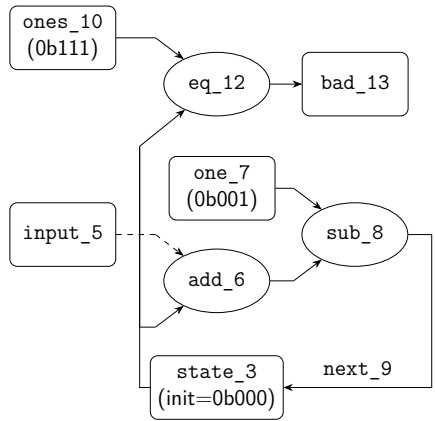
- Automatically detect issues in source code
  - Null-pointer dereference
  - Assertion violation
  - Memory leak
  - ...
- Annual tool competitions: SV-COMP
  - 59 tools on more than 30 K verification tasks in 2024 [2]

Overview of Software Verification

- Automatically detect issues in source code
  - Null-pointer dereference
  - Assertion violation
  - Memory leak
  - ...
- Annual tool competitions: SV-COMP
  - 59 tools on more than 30 K verification tasks in 2024 [2]
- CPAchecker [3]: found more than 100 bugs (confirmed and fixed) in Linux kernel modules [4]

Example: Sequential Circuit in 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 Circuits to C Programs

```
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
```

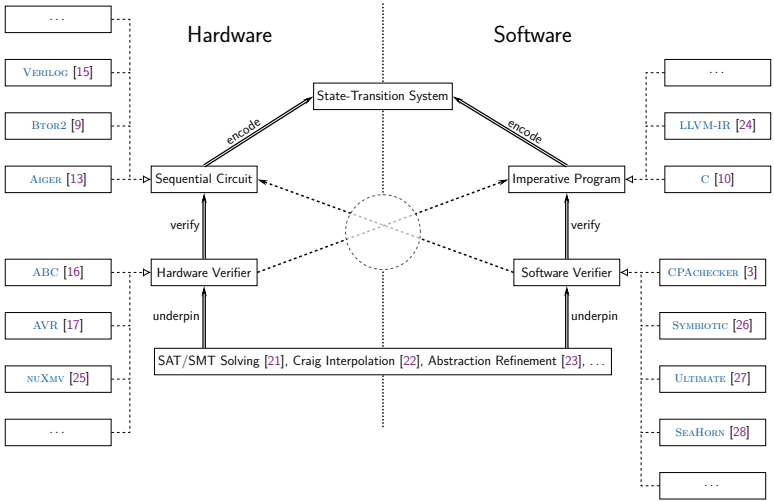
```c
void main() {
  typedef unsigned char SORT_1;
  typedef unsigned char SORT_11;
  const SORT_1 var_2 = 0b000;
  const SORT_1 var_7 = 0b001;
  const SORT_1 var_10 = 0b111;
  SORT_1 state_3 = var_2;
  for (;;) {
    SORT_1 input_5 = nondet_uchar();
    input_5 = input_5 & 0b111;
    SORT_11 var_12 = 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;
  }
}
```

Circuit-Based Program Verification

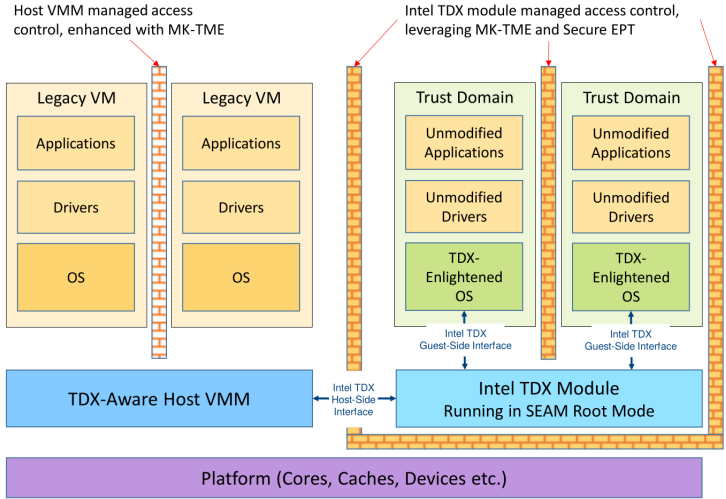
- Sequential circuit as an intermediate representation for program analysis
  - Leveraging hardware verification and logic synthesis

Our circuit-based program verifier, CPV, ranks 6$^{\text{th}}$ out of 26 in the category *ReachSafety* as a first-time participant in SV-COMP 2024!

Exploring the Interplay of Hardware and Software Verification

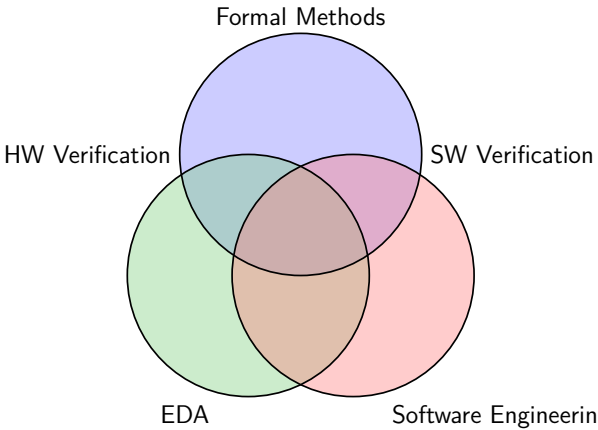


Emerging Paradigms: Confidential Computing in the Cloud



Source: Figure 2.1 in Intel TDX Module v1.5 Base Architecture Specification

We Are Looking for You!



Email:
nian-ze.lee@sosy.ifi.lmu.de

References I

[1] Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P.W., Papakonstantinou, I., Purbrick, J., Rodriguez, D.: Moving fast with software verification. In: Proc. NFM. pp. 3–11. LNCS 9058, Springer (2015). doi:10.1007/978-3-319-17524-9_1

[2] 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). doi:10.1007/978-3-031-57256-2_15

[3] 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

[4] Beyer, D., Petrenko, A.K.: Linux driver verification. In: Proc. ISoLA. pp. 1–6. LNCS 7610, Springer (2012). doi:10.1007/978-3-642-34032-1_1, https://www.sosy-lab.org/research/pub/2012-ISOLA.Linux_Driver_Verification.pdf

[5] 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 (2). pp. 152–172. LNCS 13994, Springer (2023). doi:10.1007/978-3-031-30820-8_12

[6] 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 (3). pp. 129–149. LNCS 14572, Springer (2024). doi:10.1007/978-3-031-57256-2_7

References II

[7] 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). doi:10.1007/978-3-031-57256-2_22

[8] Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. J. Autom. Reasoning (2024). doi:10.1007/s10817-024-09702-9, preprint: https://doi.org/10.48550/arXiv.2208.05046

[9] 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

[10] 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

[11] 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

[12] 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

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

References III

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

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

[16] 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

[17] 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

[18] 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

[19] 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

[20] 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

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

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

References IV

[23] 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

[24] 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

[25] 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

[26] 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

[27] 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

[28] 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

[29] Beyer, D., Chien, P.C., Jankola, M., Lee, N.Z.: A transferability study of interpolation-based hardware model checking for software verification. Proc. ACM Softw. Eng. 1(FSE) (2024). doi:10.1145/3660797

References V

[30] Manino, E., Menezes, R.S., Shmarov, F., Cordeiro, L.C.: NeuroCodeBench: a plain C neural network benchmark for software verification. arXiv/CoRR **2309**(03617) (September 2023). doi:10.48550/arXiv.2309.03617

[31] 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