Formal **M**ethods and **A**nalysis for **C**omputing and **E**ngineering

Prof. Nian-Ze Lee
nzlee@ntu.edu.tw

ForMACE Lab, National Taiwan University

February 24, EDA Seminar 2025

Agenda

Agenda

Who Am I?

- A new faculty member at NTUEE/GIEE (starting from February)
- A PostDoc at LMU Munich, Germany, from 2021 to 2024
- A PhD graduate from GIEE in 2021
- A Bachelor's graduate from NTUEE in 2014

What Do I Research and Teach?

- Formal Methods

What Do I Research and Teach?

- Formal Methods

A new course this semester: every Tuesday from 14:20 to 17:30 in EE2-225

Agenda

Problems with This Code

```
1       int binarySearch(int arr[], int left, int right, int target) {
2         while (left <= right) {
3             int mid = (left + right) / 2;
4             if (arr[mid] = target)
5                 return mid;
6             if (arr[mid] < target)
7                 left = mid;
8             else
9                 right = mid;
10        }
11      }
```

How would you debug the code?

Problems with This Code

```
1   int binarySearch(int arr[], int left, int right, int target) {
2     while (left <= right) {
3         // Bug 1: Potential integer overflow when computing mid
4         // Found in "java.util.Arrays" in 2006
5         int mid = (left + right) / 2;
6         // Bug 2: Incorrect comparison (assignment instead of comparison)
7         if (arr[mid] = target)
8             return mid;
9         // Bug 3: Incorrect updates to left and right may cause infinite loop
10        if (arr[mid] < target)
11            left = mid;
12        else
13            right = mid;
14    }
15    // Bug 4: Forgetting to return a value may cause undefined behavior
16  }
```

Problems with This Code

```
1   int binarySearch(int arr[], int left, int right, int target) {
2     while (left <= right) {
3         // Fix 1: Prevent overflow
4         int mid = left + (right - left) / 2;
5         // Fix 2: Use comparison
6         if (arr[mid] == target)
7             return mid;
8         // Fix 3: Update left and right correctly
9         if (arr[mid] < target)
10            left = mid + 1;
11        else
12            right = mid - 1;
13    }
14    // Fix 4: Return -1 to indicate target not found
15    return -1;
16  }
```

Problems with This Code

```
1   int binarySearch(int arr[], int left, int right, int target) {
2     while (left <= right) {
3         // Fix 1: Prevent overflow
4         int mid = left + (right - left) / 2;
5         // Fix 2: Use comparison
6         if (arr[mid] == target)
7             return mid;
8         // Fix 3: Update left and right correctly
9         if (arr[mid] < target)
10            left = mid + 1;
11        else
12            right = mid - 1;
13    }
14    // Fix 4: Return -1 to indicate target not found
15    return -1;
16  }
```

Q: Can we exhaustively debug the code without running test cases?
A: Formal Methods!

Demo: CPAchecker

- Automatic and static program analyzer for C programs
- Found more than 100 bugs (confirmed and fixed) in Linux kernel modules
- Top contender at annual competitions for software verifiers (SV-COMP)

“Formal” Methods in a Nutshell

- Modeling computing systems and specifying their expected behaviors so that we can (automatically) analyze their correctness with mathematical rigor
  - Automata, logic, constraint solving, etc.

“Formal” Methods in a Nutshell

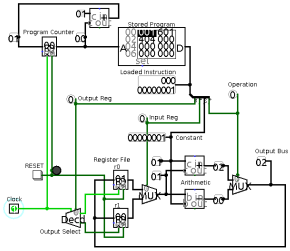
- Modeling computing systems and specifying their expected behaviors so that we can (automatically) analyze their correctness with mathematical rigor
  - Automata, logic, constraint solving, etc.
- What does “formal” mean?
  - “Form” is the key: $(A \Rightarrow B) \wedge (B \Rightarrow C) \Rightarrow (A \Rightarrow C)$

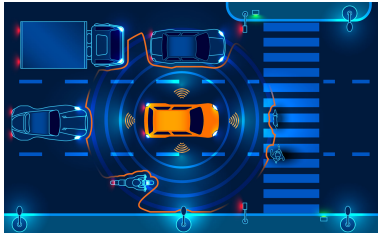
"Formal" Methods in a Nutshell

- Modeling computing systems and specifying their expected behaviors so that we can (automatically) analyze their correctness with mathematical rigor
  - Automata, logic, constraint solving, etc.
- What does "formal" mean?
  - "Form" is the key: $(A \Rightarrow B) \wedge (B \Rightarrow C) \Rightarrow (A \Rightarrow C)$
- Comparison with testing
  - Static vs. Dynamic
  - Symbolic vs. Concrete
  - All possibilities (hard!) vs. Some test cases

Agenda

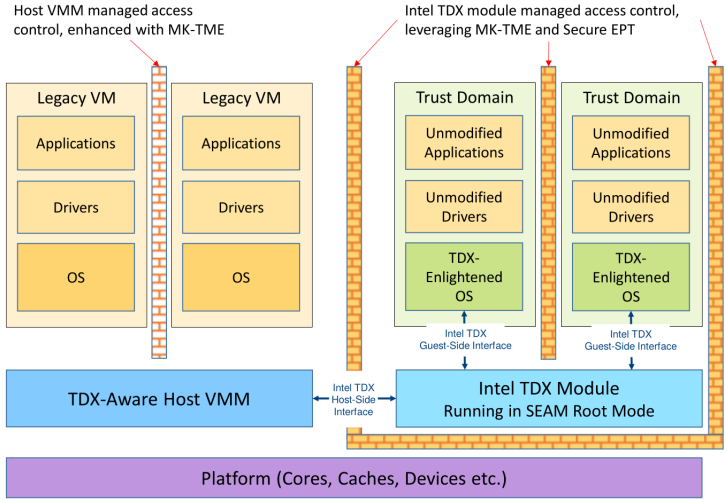
Engineering of Software, Hardware, and Cyber-Physical Systems





*Images from the Internet

Case Study: Confidential Computing for Data Security in Cloud



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

Agenda

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

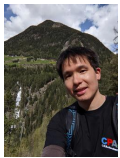
Zsófia Ádám[1,2], Dirk Beyer[2], Po-Chun Chien[2], Nian-Ze Lee[2], and Nils Sirrenberg[2]

[1]Budapest University of Technology and Economics, Hungary

[2]LMU Munich, Germany

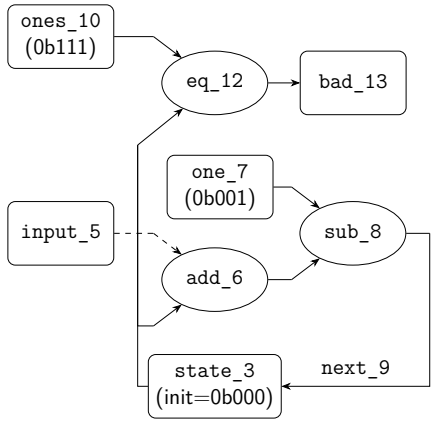
Received "Distinguished Artifact Award" at TACAS 2024



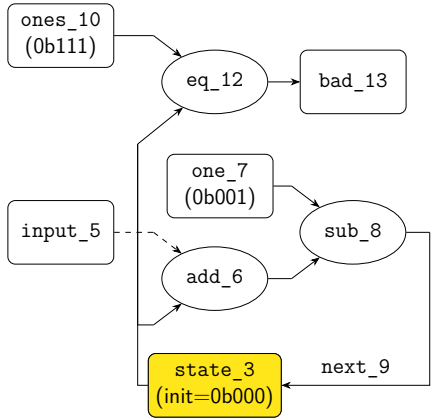

The BTOR2 Language

```
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
```



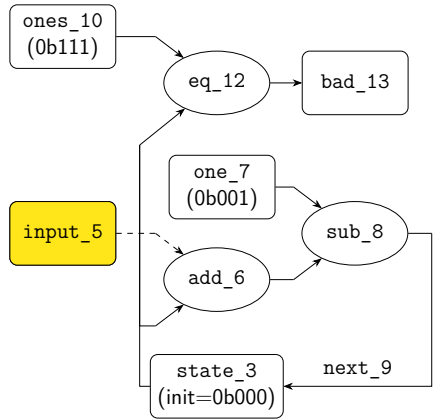
The BTOR2 Language

```
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
```



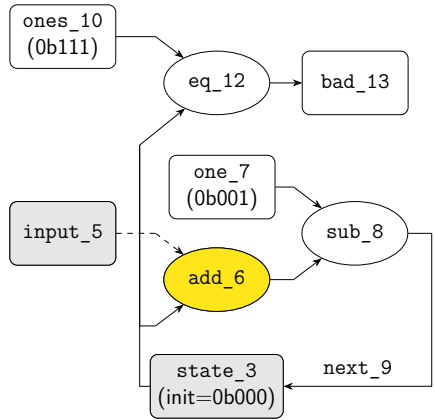
The BTOR2 Language

```
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
```



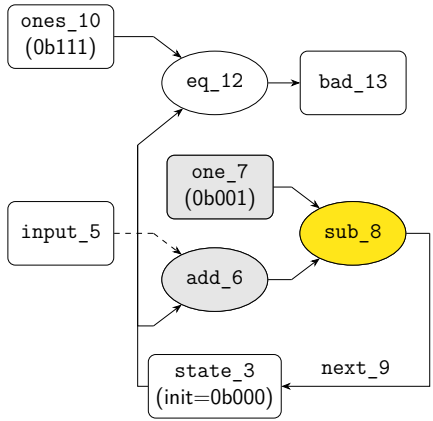
The BTOR2 Language

```
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
```



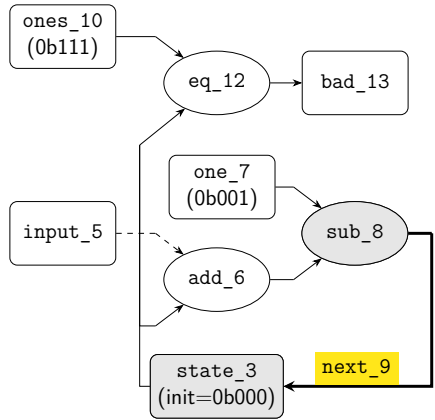
The BTOR2 Language

```
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
```



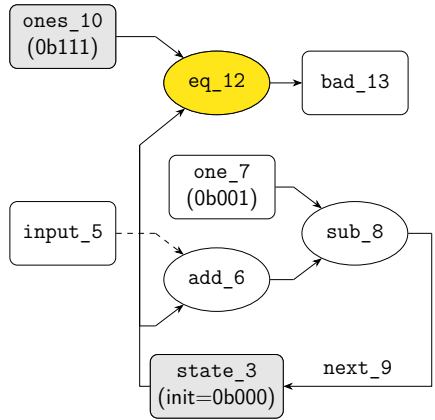
The BTOR2 Language

```
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
```



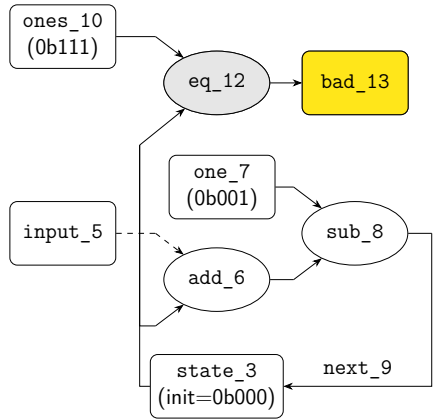
The BTOR2 Language

```
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
```

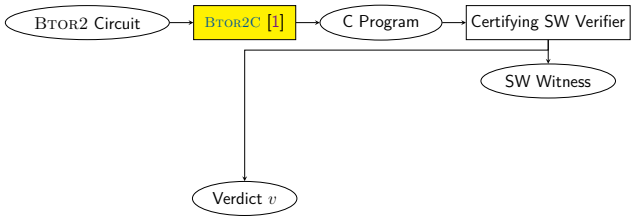


Translating BTOR2 Circuits to C Programs

```
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
```

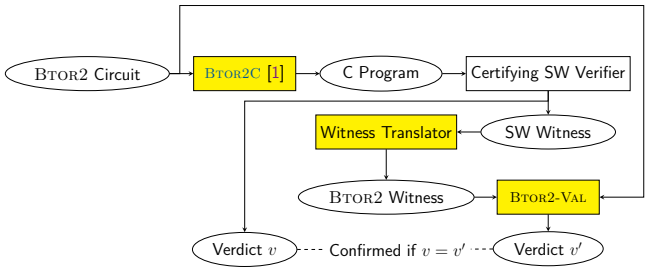
```c
void main() {
  typedef unsigned char SORT_1;
  typedef unsigned char SORT_11;
  const SORT_1 var_2 = 0b000;
  const SORT_1 var_7 = 0b001;
  const SORT_1 var_10 = 0b111;
  SORT_1 state_3 = var_2;
  for (;;) {
    SORT_1 input_5 = nondet_uchar();
    input_5 = input_5 & 0b111;
    SORT_11 var_12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

Certifying Verification for BTOR2 with SV Tools



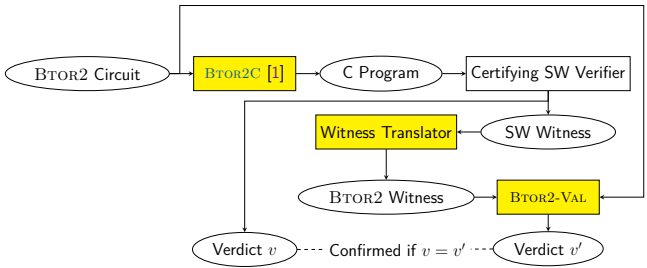
- BTOR2 [5] word-level circuits and translator BTOR2C [1]
- Software verifiers in SV-COMP [18]

Certifying Verification for BTOR2 with SV Tools



- BTOR2 [5] word-level circuits and translator BTOR2C [1]
- Software verifiers in SV-COMP [18]
- Software-to-hardware 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 [18]
- Software-to-hardware witness translation and BTOR2-VAL
- On 1214 BTOR2 circuits, BTOR2-CERT
  - found 37 bugs that ABC [13] missed using CBMC [19]
  - derived invariants to accelerate ABC using CPACHECKER [7]

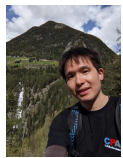
Summary

- Btor2-Cert: certifying and validating hardware verifier using software verifiers
- Reproduction artifact [20] available on Zenodo
  - **Distinguished Artifact Award** at TACAS 2024

CPV: A Circuit-Based Program Verifier

Po-Chun Chien and Nian-Ze Lee

LMU Munich, Germany



Research Question

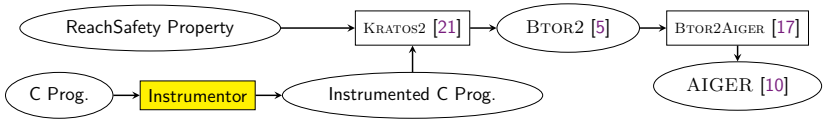
- Sequential circuit as an intermediate representation for program analysis
  - Leveraging hardware model checkers as backend

Research Question

- Sequential circuit as an intermediate representation for program analysis
  - Leveraging hardware model checkers as backend

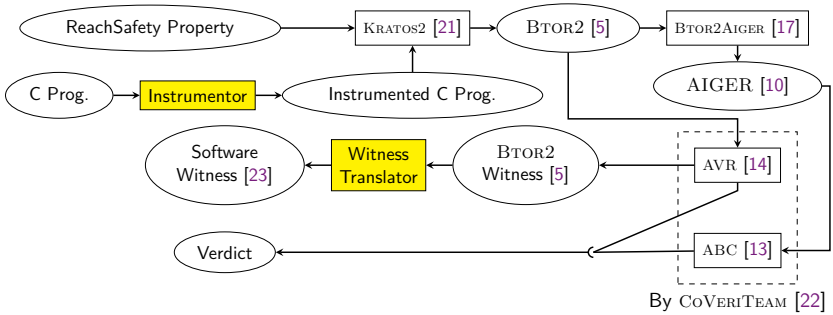
CPV ranked 6th out of 26 in the category *ReachSafety* as a first-time participant in SV-COMP 2024. (and a higher position in 2025!)

Software Architecture of CPV



- Program instrumentation for retrieving witness information
- Software-to-hardware translation by KRATOS2 [21]

Software Architecture of CPV



- Program instrumentation for retrieving witness information
- Software-to-hardware translation by Kratos2 [21]
- Translated circuits verified by hardware model checkers
- Hardware-to-software witness translation

Agenda

Confidential Computing

- Scenario: sensitive data to train ML models using cloud services

Confidential Computing

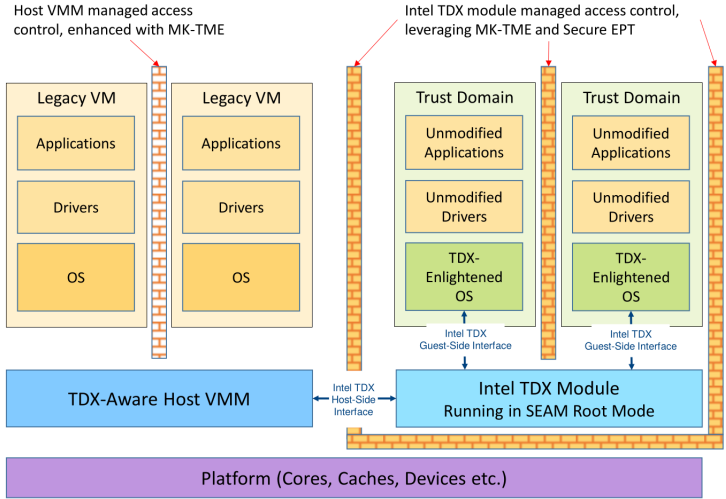
- Scenario: sensitive data to train ML models using cloud services
  $\rightarrow$ Do not want to trust cloud operators

Confidential Computing

- Scenario: sensitive data to train ML models using cloud services
  $\rightarrow$ Do not want to trust cloud operators

How can we protect data when they are in use, especially in a remote execution environment, e.g., cloud?

Intel Trust Domain Extensions (TDX)



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

Intel TDX: Components

- Hardware extensions: new CPU mode, highest privilege
- Firmware components: TDX module
  - Loaded in special memory range
  - Executed with highest privilege

Intel TDX: Components

- Hardware extensions: new CPU mode, highest privilege
- Firmware components: TDX module
  - Loaded in special memory range
  - Executed with highest privilege
- Host VMM manages trust domains (TDs) via TDX module
  - Using application binary interfaces (ABIs), no direct access

Intel TDX: Components

- Hardware extensions: new CPU mode, highest privilege
- Firmware components: TDX module
  - Loaded in special memory range
  - Executed with highest privilege
- Host VMM manages trust domains (TDs) via TDX module
  - Using application binary interfaces (ABIs), no direct access

Goal: verify ABIs of TDX module (implemented as C code plus assembly), assuming VMM and TDs can call any ABI with any inputs

Intel TDX: Specification for ABI Function `TDG.SYS.RD`

Table 5.324: TDG.SYS.RD Input Operands Definition

Operand	Description		
RAX	TDCALL instruction leaf number and version, see 5.4.1		
	Bits	**Field**	**Description**
	15:0	Leaf Number	Selects the TDCALL interface function
	23:16	Version Number	Selects the TDCALL interface function version Must be 0
	63:24	Reserved	Must be 0
RDX	Field identifier – see 3.10 The LAST_ELEMENT_IN_FIELD and LAST_FIELD_IN_SEQUENCE components of the field identifier must be 0. WRITE_MASK_VALID, INC_SIZE, CONTEXT_CODE and ELEMENT_SIZE_CODE components of the field identifier are ignored. A value of -1 is a special case: it is not a valid field identifier; in this case the first readable field identifier is returned in RDX.		

Table 5.325: TDG.SYS.RD Output Operands Definition

Operand	Description
RAX	TDCALL instruction return code – see 5.4.1
RDX	If the input field identifier was -1, RDX returns the first readable field identifier. Else, in case of an error, RDX returns -1. On success, RDX returns the next readable field identifier. A value of -1 indicates no next field identifier is available.
R8	Contents of the field In case of no success, as indicated by RAX, R8 returns 0.
Other	Unmodified

Source: Intel TDX Module v1.5 ABI Specification

Firmware-Specific Constructs

- Byte/Bit-precise modeling of memory layouts (type punning)
- Inline assembly
- Externally defined variables

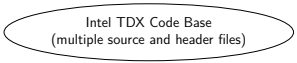
Example of Inline Assembly: Access Loader-Defined Variables

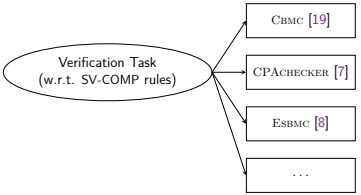
```
1  _STATIC_INLINE_ tdx_module_local_t* get_local_data(void) {
2    uint64_t local_data_addr;
3    _ASM_("movq %%gs:%c[local_data], %0\n\t"
4          : "=r"(local_data_addr)
5          : [local_data] "i"(
6              offsetof(tdx_module_local_t, local_data_fast_ref_ptr)));
7    return (tdx_module_local_t*)local_data_addr;
8  }
```

Modeling Inline Assembly via Program Instrumentation

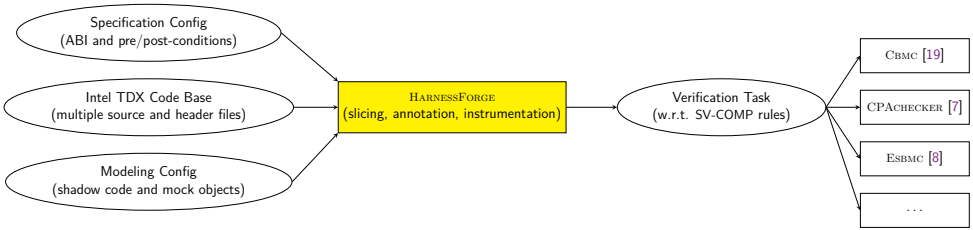
```
1  _STATIC_INLINE_ tdx_module_local_t* get_local_data(void) {
2  #ifdef TDXFV_NO_ASM
3    return &local_data_fv;
4  #else
5    uint64_t local_data_addr;
6    _ASM_("movq %%gs:%c[local_data], %0\n\t"
7          : "=r"(local_data_addr)
8          : [local_data] "i"(
9              offsetof(tdx_module_local_t, local_data_fast_ref_ptr)));
10   return (tdx_module_local_t*)local_data_addr;
11 #endif
12 }
```

HarnessForge: Generating Verification Tasks Intel TDX



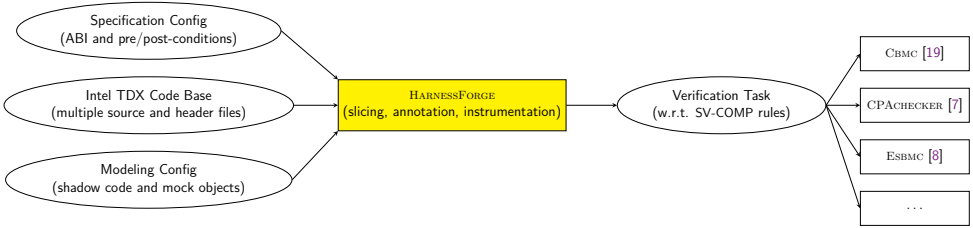


HarnessForge: Generating Verification Tasks Intel TDX



- Slice off irrelevant code
- Annotate pre/post-conditions
- Instrument shadow C code to model inline assembly

HarnessForge: Generating Verification Tasks Intel TDX



- Slice off irrelevant code
- Annotate pre/post-conditions
- Instrument shadow C code to model inline assembly

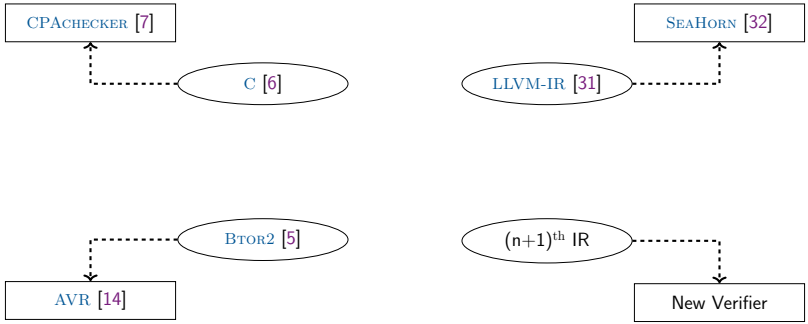
Vision: extend HarnessForge to arbitrary code base
(integrated into the build process, like common testing frameworks)

Agenda

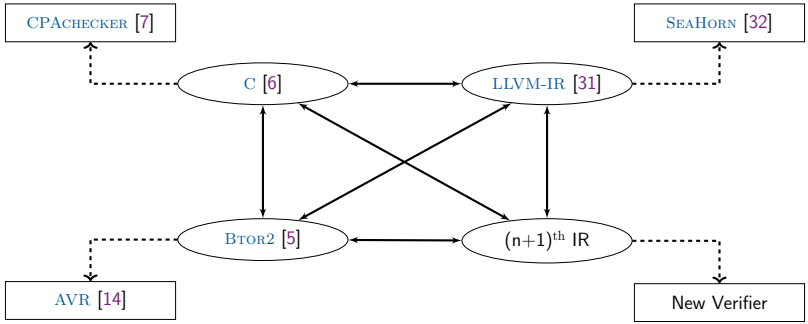
Intersection of Hardware and Software Verification: So Far

- Software analyzers uniquely solving hardware tasks [1, 2]
- Hardware-verification algorithms [4] and tools [3] improving software analysis
- Transferability of algorithmic characteristics [24]
- Firmware verification requiring expertise of both sides

Critical Reflection: The Transformation Game [30]



Critical Reflection: The Transformation Game [30]



Developing a transformation network between different representations to leverage their unique strengths

Practical Verification Challenges

- Firmware
  - Ex: used in confidential computing to protect data security in the cloud
  - Current practices: manual review or testing
  - Gap: specific constructs (e.g., inline assembly)

Practical Verification Challenges

- Firmware
  - Ex: used in confidential computing to protect data security in the cloud
  - Current practices: manual review or testing
  - Gap: specific constructs (e.g., inline assembly)
- Neural networks
  - Ex: used in autonomous driving for object detection
  - Current practices: mostly on models (cf. VNN-COMP)
    - Less on C implementations (e.g., NeuroCodeBench [33])
  - Gap: mathematical operations (e.g., sigmoid function)

Practical Verification Challenges

- Firmware
  - Ex: used in confidential computing to protect data security in the cloud
  - Current practices: manual review or testing
  - Gap: specific constructs (e.g., inline assembly)
- Neural networks
  - Ex: used in autonomous driving for object detection
  - Current practices: mostly on models (cf. VNN-COMP)
    - Less on C implementations (e.g., NeuroCodeBench [33])
  - Gap: mathematical operations (e.g., sigmoid function)
- Hardware/software co-design (embedded systems)
  - Ex: hardware accelerators
  - Current practices: lower level verification conditions [34]
  - Gap: scalability and lack of modularity

Welcome to ForMACE Lab!

- Multiple positions at NTU and LMU Munich
- Working at the intersection of
  - Hardware vs. Software (and more!)
  - Academic tools vs. Industrial applications
- New course "Formal Methods" (EE2-225, every Tuesday)
- Contact: nzlee@ntu.edu.tw (Office: EE2-349)

https://formace-lab.gitlab.io/webpage/

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 (2). pp. 152–172. 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 (3). pp. 129–149. LNCS 14572, Springer (2024). doi:10.1007/978-3-031-57256-2_7

[3] 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

[4] Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. J. Autom. Reasoning **69** (2025). doi:10.1007/s10817-024-09702-9, preprint: 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

[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

References II

[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

[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

References III

[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] 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

[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). doi:10.1007/978-3-540-24730-2_15

[20] Ádám, Z., Beyer, D., Chien, P.C., Lee, N.Z., Sirrenberg, N.: Reproduction package for TACAS 2024 article 'Btor2-Cert: A certifying hardware-verification framework using software analyzers'. Zenodo (2024). doi:10.5281/zenodo.10548597

[21] 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

[22] 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

[23] 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

References IV

[24] 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

[25] Intel Trust Domain Extensions, `https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html`, accessed: 2024-05-01

[26] Arm Confidential Compute Architecture, `https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture`, accessed: 2024-09-02

[27] Li, X., Li, X., Dall, C., Gu, R., Nieh, J., Sait, Y., Stockwell, G.: Design and verification of the Arm confidential compute architecture. In: Proc. OSDI. pp. 465–484. USENIX Association (2022), `https://www.usenix.org/system/files/osdi22-li.pdf`

[28] Fox, A.C.J., Stockwell, G., Xiong, S., Becker, H., Mulligan, D.P., Petri, G., Chong, N.: A verification methodology for the Arm confidential computing architecture: From a secure specification to safe implementations. Proc. ACM Program. Lang. **7**(OOPSLA1), 376–405 (2023). doi:10.1145/3586040

[29] AMD Infinity Guard, `https://www.amd.com/en/products/processors/server/epyc/infinity-guard.html`, accessed: 2024-09-02

References V

[30] Beyer, D., Lee, N.Z.: The transformation game: Joining forces for verification. In: Principles of Verification: Cycling the Probabilistic Landscape. pp. 175–205. LNCS 15262, Springer (2024). doi:10.1007/978-3-031-75778-5_9

[31] 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

[32] 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

[33] 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

[34] 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

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

[36] 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 VI

[37] 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

[38] 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

[39] 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

[40] 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