Bridging Hardware and Software Analysis with Btor2C

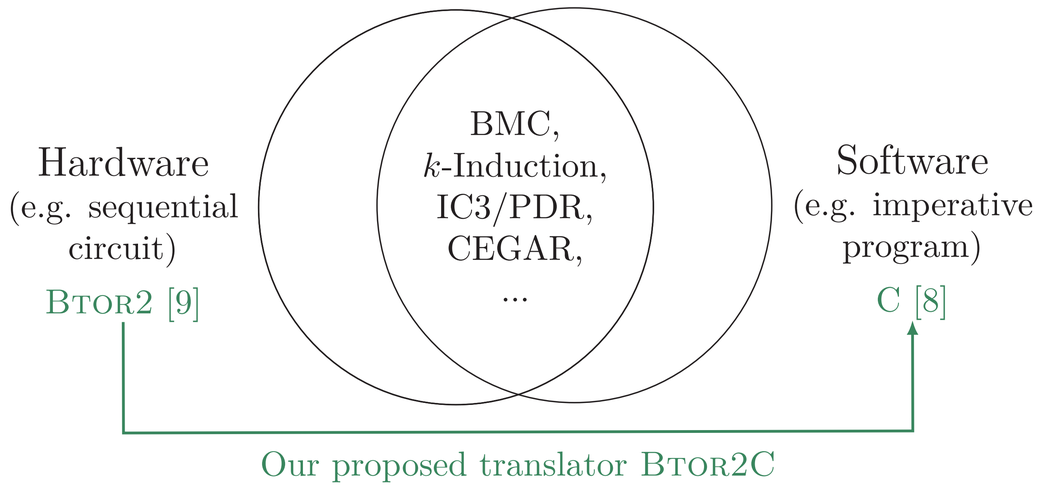
Dirk Beyer, Po-Chun Chien, and Nian-Ze Lee

{dirk.beyer, po-chun.chien, nian-ze.lee}@sosy.ifi.lmu.de

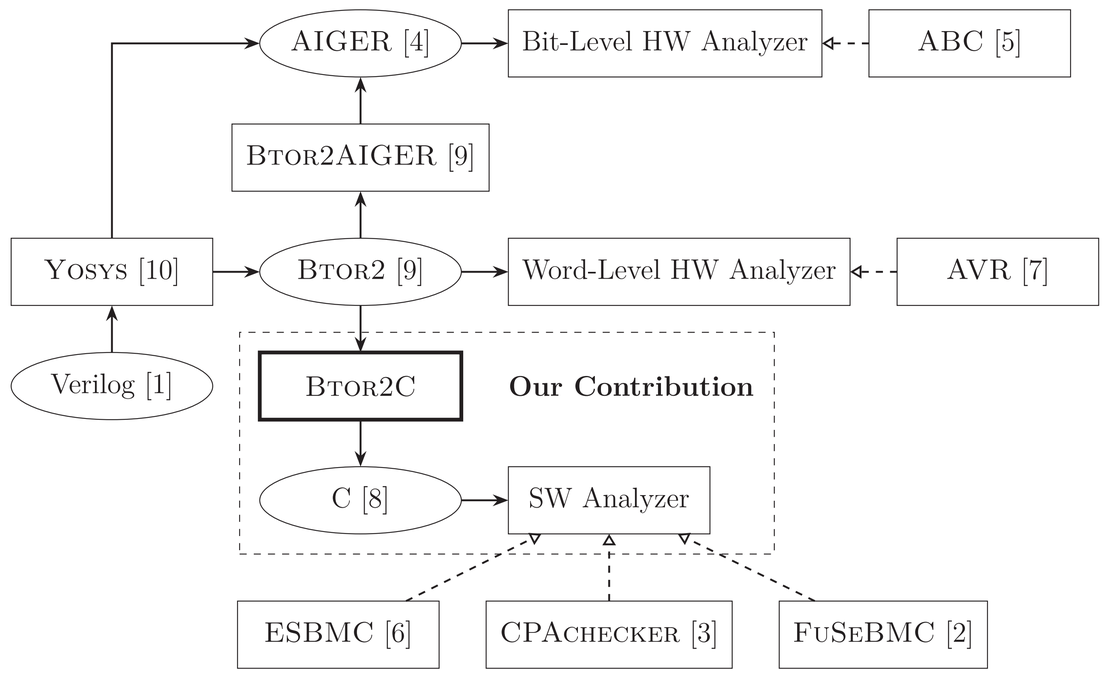
Attend our TACAS talk! 11:00 April 26 (Wednesday), Room: 44-45/108

Motivation

- Quality assurance is important for computational systems
- Common theoretical foundations are shared among different fields
- Analysis tools/algorithms are usually developed for a specific model
- Leveraging knowledge from one field to another is **difficult!**

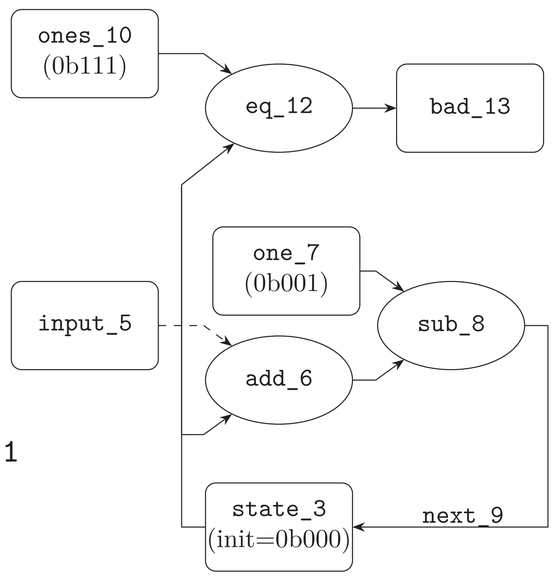


Analysis Tool Chain



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
```



(a) Example Btor2 circuit

- Each line represents a node and has a unique id
- A typical Btor2 operation is written as
  `<nid0> <op> <sid> <nid1> [<nid2 [nid3]>]`

Btor2	C
`sort bitvec`	unsigned integers
`sort array`	2D static array
`input`	non-deterministic function
`bad`	if guard w/ an `ERROR` label
`a add x b c`	`var_a = var_b + var_c;`
`a and x b c`	`var_a = var_b & var_c;`
`a ite x b c d`	`var_a = var_b ? var_c : var_d;`
`a concat x b c`	`var_a = var_b << N | var_c;` (`N` is the bit-width of `var_c`)
`a udiv x b c`	`var_a = (var_c == 0) ? MAX_X : (var_b / var_c);` (`MAX_X` is the max value of `SORT_x`)

(b) Model Btor2 constructs with C

```c
void main() {
  // Define sorts
  typedef unsigned char SORT_1;
  typedef unsigned char SORT_11;
  // Initialize states
  SORT_1 state_3 = 0b000;
  for (;;) { // model circuit behavior
    // Evaluate safety properties
    SORT_1 input_5 = nondet_uchar();
    SORT_11 var_12 = state_3 == 0b111;
    SORT_11 bad_13 = var_12;
    if (bad_13) { ERROR: abort(); }
    // Compute and assign next states
    SORT_1 var_6 = state_3 + input_5;
    SORT_1 var_8 = var_6 - 0b001;
    var_8 = var_8 & 0b111;
    state_3 = var_8;
  }
}
```

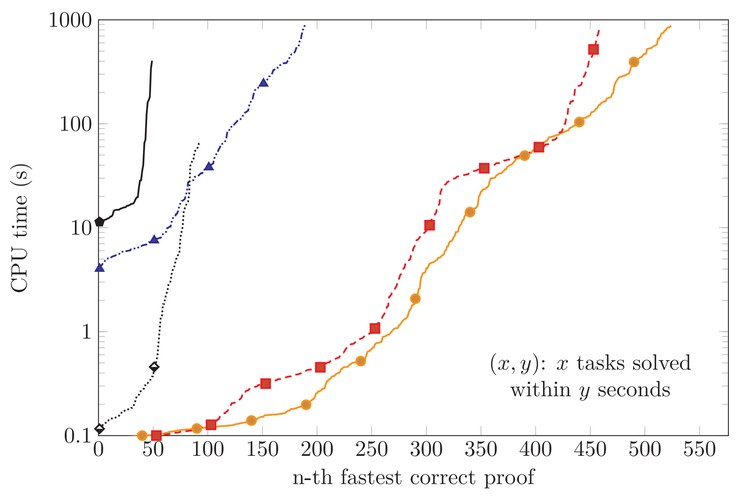
(c) Translated C program

Try Btor2C!

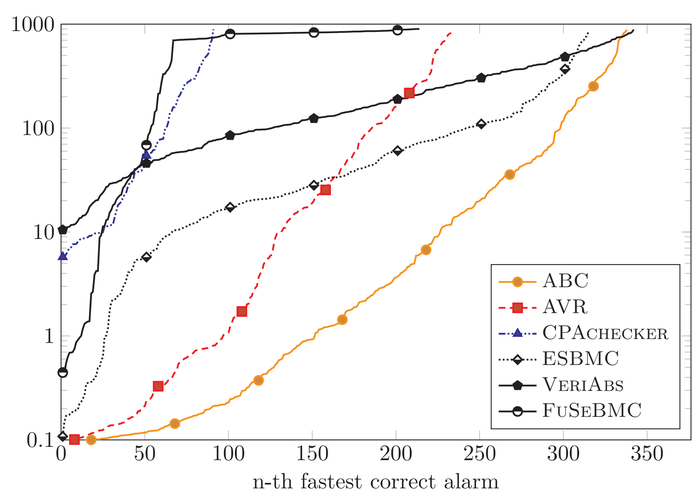
- Btor2C is a lightweight tool written in C
- Open-source under Apache 2.0 License

gitlab.com/sosy-lab/software/btor2c

Software vs. Hardware Analyzers on HWMCC Tasks



(d) Correct Proofs



(e) Correct Alarms

Summary

- Btor2C: a translator from word-level Btor2 sequential circuits to C programs
- **43** HW-verification tasks were uniquely solved by SW analyzers in our evaluation
- With Btor2C, we can
  - verify HW designs with off-the-shelf SW analyzers, and
  - improve quality assurance for HW systems
- In the future, we plan to bridge the gap from the other direction, i.e., translate SW programs into HW circuits.

References

[1] IEEE Standard for Verilog Hardware Description Language (2006)
[2] 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 (2021)
[3] Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806 (2011)
[4] 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)
[5] Brayton, R., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: Proc. CAV. pp. 24–40. LNCS 6174 (2010)
[6] 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 (2018)
[7] Goel, A., Sakallah, K.: AVR: Abstractly verifying reachability. In: Proc. TACAS. pp. 413–422. LNCS 12078 (2020)
[8] ISO/IEC JTC 1/SC 22: ISO/IEC 9899-2018: Information technology — Programming Languages — C (2018), https://www.iso.org/standard/74528.html
[9] Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2, BtorMC, and Boolector 3.0. In: Proc. CAV. pp. 587–595. LNCS 10981 (2018)
[10] Wolf, C.: Yosys open synthesis suite. https://yosyshq.net/yosys/, accessed: 2023-01-29