Open Source Formal Verification

engines

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

July 2025

Introduction

Model Checking

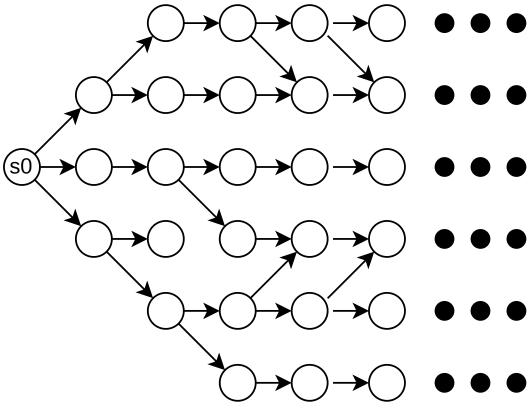
Model checking

Method for checking whether a finite-state model of a system meets a given specification

- Applied to state machines
  - A hardware model is by itself a state machine
    - State represented by the D flip-flops and memories
    - Inputs are simply the inputs of the module
  - The specification is given by properties and invariants

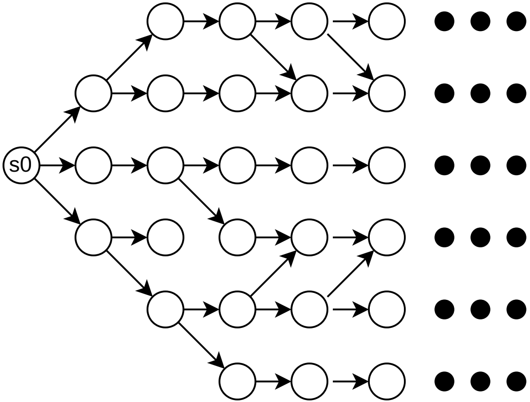
Model Checking

- Concept:
  - From a reset condition
  - Try to find a state where an assertion fails
- Process:
  1. Start from an initial state
  2. Check that assertions hold
  3. Generate the next states
  4. Back to 2
- *Success* if no failure is detected
- *Fail* if an assertion fails
- *Unconclusive* if the entire state space cannot be reached



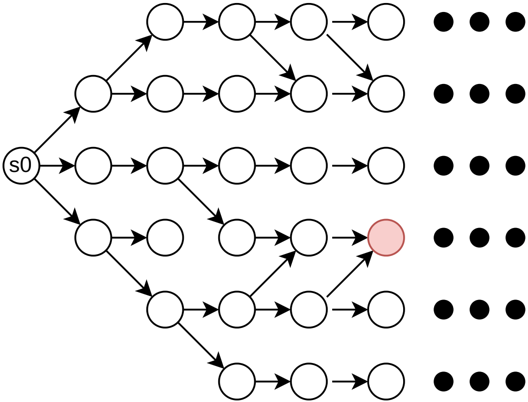
Bounded Model Checking

- Concept:
  - From a reset condition
  - Try to find a state where an assertion fails
  - For at most *N* steps (the depth)
- Process:
  1. Start from an initial state
  2. Check that assertions hold
  3. Generate the next states
  4. Back to 2
- *Success* if no failure is detected
- *Fail* if an assertion fails
- *Unconclusive* if the entire state space cannot be reached



Bounded Model Checking

- Red state corresponds to a failing assertion
- Found after 5 cycles
- If the depth is at least 5, then the failure will be detected
- If the depth is less that 5, then the checker will not detect it



What depth?

- Crucial question: What depth?
  - Too short $\Rightarrow$ Does not verify anything
  - Too big $\Rightarrow$ Too much processing time
- Depends on the system
- Examples:
  - For a FIFO, it seems fair not to check more writes than 4 times the FIFO size
  - For a counter, if it offers a load operation, 2-3 steps are sufficient
  - For a counter, without a load operation, should be at least the maximum value
    - A good argument for generic parameters

Proof

- BMC is not ideal because of the depth to be chosen
- Can we prove the correctness?
- Option: k-induction

k-induction

- Concept: Check correctness of K steps and then prove that if any K steps are correct, then the $(K+1)^{th}$ will be correct

Example: 1-induction

$$P(0) \wedge \forall n (P(n) \Rightarrow P(n+1)) \Rightarrow \forall n P(n)$$

Example: 2-induction

$$P(0) \wedge P(1) \wedge \forall n ((P(n) \wedge P(n+1)) \Rightarrow P(n+2)) \Rightarrow \forall n P(n)$$

Interesting: `http://www.ccs.neu.edu/home/wahl/Publications/k-induction.pdf`

k-induction

- Mathematically, k-induction is identical to 1-induction
- Why k?
- Because it helps the proof, starting with more initial conditions

Verification

- The proof exploits a solver
- The solvers are usually based on SAT or SMT solvers
- SAT (Boolean satisfiability problem)
  - Is there a combination of variables that satisfy a boolean equation in its conjunctive form?
  - $(x1 \wedge x3) \vee (\overline{x2} \wedge x3) \vee (x1 \wedge \overline{x3})$
  - NP-complete problem
- SMT (satisfiability modulo theories)
  - Generalizes SAT to more complex formulas involving:
    - Real numbers, integers
    - Lists, arrays, bit vectors
    - ...

Engines in SBY

Mode	Engine
bmc	smtbc [all solvers] btor btormc btor pono abc bmc3 abc sim3 aiger aigbmc
prove	smtbmc [all solvers] abc pdr aiger suprove
cover	smtbmc [all solvers] btor btormc

smtbmc solvers (1)

- The SMTBMC engine supports the following solvers:
  - yices (`https://yices.csl.sri.com/`)
    - Yices 2 is an SMT solver that decides the satisfiability of formulas containing uninterpreted function symbols with equality, real and integer arithmetic, bitvectors, scalar types, and tuples.
  - boolector (`https://boolector.github.io/`)
    - A Satisfiability Modulo Theories (SMT) solver for the theories of fixed-size bit-vectors, arrays and uninterpreted functions.
  - bitwuzla (`https://bitwuzla.github.io/`)
    - Bitwuzla is a Satisfiability Modulo Theories (SMT) solver for the theories of fixed-size bit-vectors, floating-point arithmetic, arrays and uninterpreted functions and their combinations.
  - z3 (`https://github.com/Z3Prover/z3`)
    - Z3 is a theorem prover from Microsoft Research.

smtbmc solvers (2)

- The SMTBMC engine supports the following solvers:
  - mathsat (`https://mathsat.fbk.eu/`)
    - MathSAT 5 is an efficient Satisfiability modulo theories (SMT) solver supporting a wide range of theories (including e.g. equality and uninterpreted functions, linear arithmetic, bit-vectors, and arrays) and functionalities.
  - cvc4 (`https://cvc4.github.io/`)
    - CVC4 is an efficient open-source automatic theorem prover for satisfiability modulo theories (SMT) problems. It can be used to prove the validity (or, dually, the satisfiability) of first-order formulas in a large number of built-in logical theories and their combination.
  - cvc5 (`https://cvc5.github.io/`)
    - cvc5 is an efficient open-source automatic theorem prover for Satisfiability Modulo Theories (SMT) problems. It can be used to prove the satisfiability (or, dually, the validity) of first-order formulas with respect to (combinations of) a variety of useful background theories.

btor

- The btor engine works with btor2 files. It supports the following solvers:
  - btormc: `https://github.com/Boolector/boolector`
  - pono: `https://github.com/stanford-centaur/pono`

aiger

- The aiger engine (`https://github.com/arminbiere/aiger`) offers two solvers:
  - aigbmc
  - suprove (quite good for proofs)

abc

- The abc engine (`https://github.com/berkeley-abc/abc`) offers three solvers:
  - pdr (most used)
  - bmc3
  - sim3

Prove vs BMC

- A proof can be faster than bounded model checking
- For BMC you need to setup the depth
  - Choice to be made
    - As small as possible, but not smaller
    - *"Aussi petite que possible, mais aussi grande que nécessaire"*

Conclusion

A proof is better than bounded model checking

$\Rightarrow$

Use a proof is possible, else go for model checking

- Do not hesitate to compare various engines to find the most efficient one
  - Can be done by hand
  - or with the `-autotune` option of `sby`
  - Reduces the computing time for regression tests