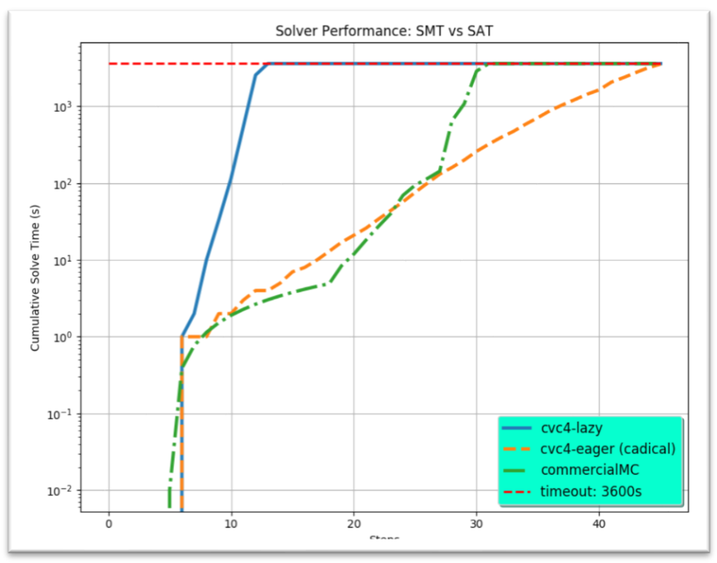
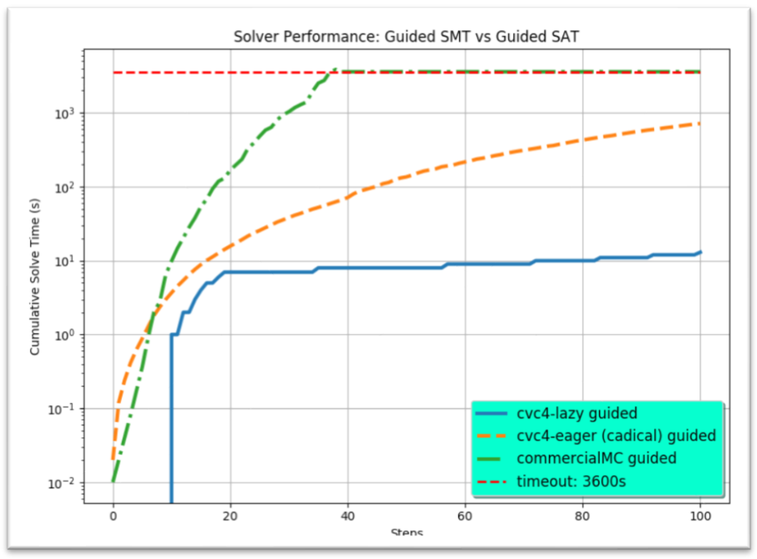
Finding Critical Clauses in SMT-based Hardware Verification

Evidence of Hope for SMT BMC

Guiding solvers:

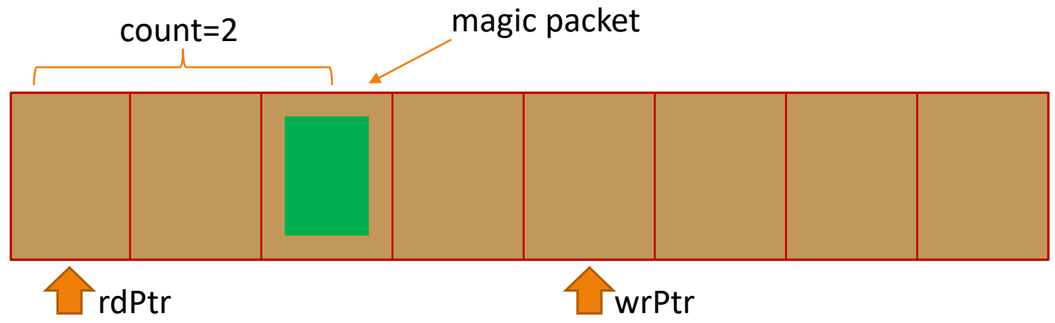
- Lemma mining
- Proof lifting
  - See *Transition Relation Techniques*
- Partial Order Reduction
- Encoding tricks: booleans vs. bit-vectors
- Much bigger impact on lazy SMT solving





Data Integrity Proof

- Non-deterministically choose a "magic packet"
- Keep track of the location
- Expect it to match when exits



3 Major Approaches

Design Info

- Configured Processing Element (PE)
  - Essentially an ALU
  - Configured for multiplication by 2

Verification

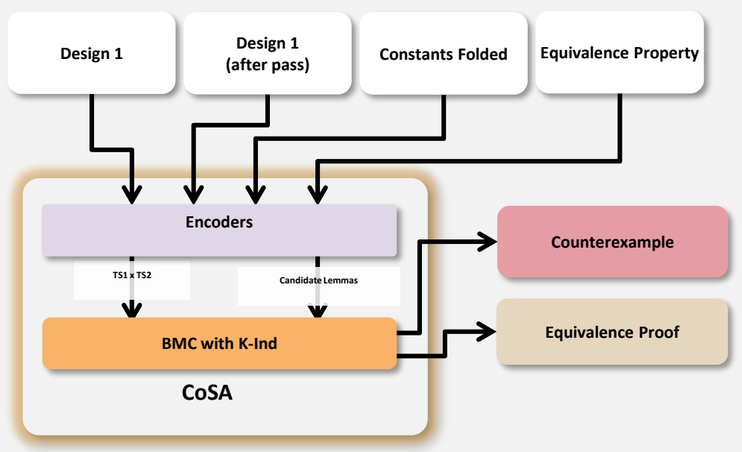
- Design before and after optimization pass

CoerIR Pass: Fold-Constants

- Replace any sub-circuit with constant output with a constant
- Can change state elements

Modular (Structural) Techniques

- Automated lemma extraction
  - **Nothing** in the pass is trusted
  - But it can provide hints
    - Check substitutions
    - Then add as lemmas
- Significant performance improvement
  - 1.3 min vs. T.O. at 2 hours
  - Found bug in reduction-and



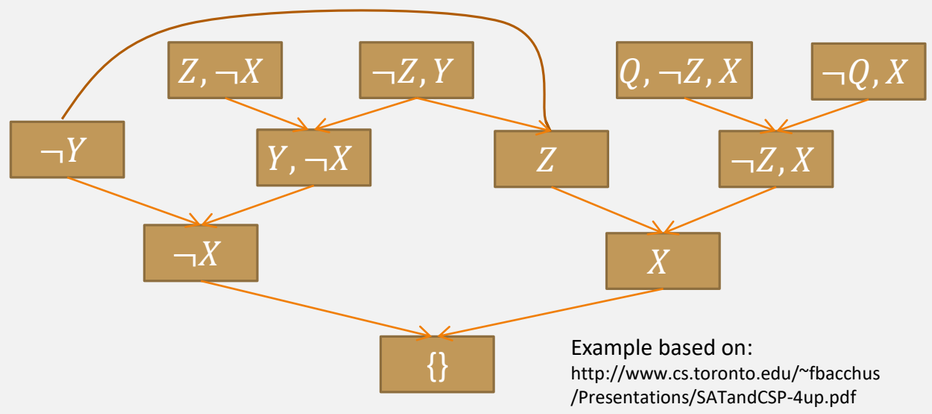
Statistical Techniques

Learning From Previous Bounds

- Bounded Model Checking
  - Sequence of unsat calls, potentially followed by sat
- Resolution Refutation Proof in SAT
  - Vector encoding of clauses
  - Literals mapped to representative variable from circuit
  - Heuristically assign scores to clauses
  - Train ML algorithm to score clauses

Applying Model

- Use learned model from bound j < k
- Generate many resolvents and keep high scoring ones
  - Achieved 40% reduction in solve time at bound 11
- Currently dominated by learning
- Next Steps:
  - Permutation invariant encoding
  - Learning good splits



Example Resolution Refutation Proof

Transition Relation Techniques

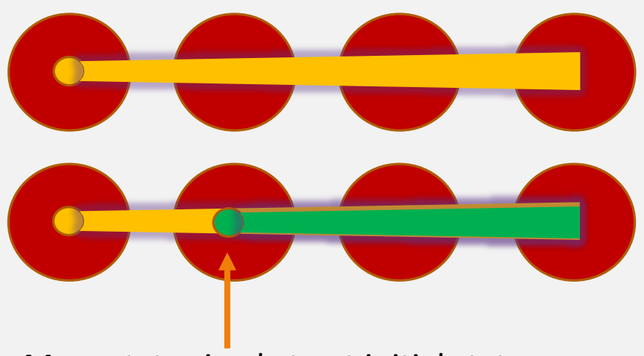
Partial Order Reduction

- Simplest: No-stutter
  - Disallow "no-ops"
  - More efficient to make assumption on inputs e.g. assume(**push** | **pop**)
- Requires side condition proof
  - Show that the partial order reduction is safe

Proof lifting (modulo circuit unrolling)

Init[0] |-> Prop[k] ➡ Init[1] |-> Prop[k+1]

- Weak initial state, strong lifted proof
- 2 Requirements on initial state:
  - Contain concrete initial state
  - Preserve property (if it holds)
- Add lifted proof at next bound



MAKAI MANN, CLARK BARRETT

Stanford University