**How Efficient are Software Verifiers for Hardware?**

Rajdeep Mukherjee, Daniel Kroening, Tom Melham

### HARDWARE MODEL CHECKING

**Transition system:** \( T = (I, x, δ) \), where \( x = x_1, x_2, ..., x_n \) is the set of variables over \( B = \{true, false\} \), \( I(x) \) is the initial state and \( δ(x, x') \) represents the transition relation.

**State:** A state \( s \) of \( T \) is an assignment of values to variable \( x \).

**Trace:** A trace \( γ : s_0, s_1, ... \) is an infinite sequence of states such that \( s_0 = I \), and for each \( i \geq 0 \), \( (s_i, s_{i+1}) = δ \).

**Reachable State:** A state \( s \) is reachable in \( T \) if \( ∃ γ : γ \in T \) s.t. \( γ \). We denote the reachable state space as \( Q \).

**Safety Property:** A safety property \( P \) of \( T \) is a first-order formula over the variables \( X \) of \( T \), which asserts that certain states \( s \) of \( T \) cannot be reached during the execution of \( T \), often known as bad states, \( B(s) \).

**Problem Statement:** Given a state-space over \( n \) boolean variables, the problem is to decide whether \( T \models P \), that is, starting from initial state \( I(x) \), whether a state in \( B(x) \) can be reached following only transitions in \( T(x, x') \).

### TECHNIQUES

**Bounded Model Checking:**

\[
I(x_0) \bigwedge_{i=1}^{k-1} (T(x_i, x_{i+1}) \land (v_i \leq i B(x_i)))
\]

**BMC with K-induction:**

\[
P(x_0) \bigwedge_{i=0}^{k-1} (T_i \land P_i) \Rightarrow P_k
\]

**Interpolation-based Model Checking:**

\[
Q \bigwedge_{i=0}^{k-1} (T_i \land P_i) \Rightarrow P_k
\]

**IC3/Property Directed Reachability:**

1. Transition Relation: \( T := \{(b, b') \mid \exists x. x' \in S : (T(x, x') \land \alpha(x) = b \land \alpha(x') = b') \land \alpha \text{ is abstraction function}, x = \{x_1, x_2, ..., x_n\}, b = \{b_1, ..., b_n\}, b_i = \pi_i(x), \pi_i \text{ is the predicate on concrete variable } x_i \}

2. Initial State: \( \hat{I}(b) := \exists x \in S : (\alpha(x) = b) \land I(x) \)

3. Safety Property: \( \hat{P}(b) := \forall x \in S : (\alpha(x) = b) \Rightarrow P(x) \)

### FROM BITS TO WORD SOFTWARE NETLIST

**Software Netlist:** A Software Netlist is defined as the six tuple, \( SN = (In, Out, Seq, Comb, Init, Asgn) \), where In, Out, Seq, Comb, Init are inputs, output, sequential/state-holding, combinational/stateless signals and initial states respectively. Asgn is a finite set of assignments to Out, Seq and Comb.

- \( Asgn := \{Cassign|Isassign\} \)
- \( Cassign := \{V = \text{eval}(\text{VAR})\} | \{V = \text{bool}\} \), where \( V \in \text{Const} \cup \text{Out} \)
- \( Isassign := \{V = \text{eval}(\text{VAR})\} | \{V = \text{seq}\} \)
- \( \text{eval} := \text{eval}_{\text{op}}(\{\text{eval}_{\text{op}}(\text{expr}_{\text{op}})\}) \)
- \( \text{expr}_{\text{op}} := \text{var} | \text{const} | \text{op} \text{func}(\text{expr}_{\text{op}}) \)
- \( \text{op} \text{func} := \{\text{add}, \text{sub}, \text{and}, \text{or}, \text{not}\} \)
- \( \text{var} := \{x_1, x_2, ..., x_n\} \)
- \( \text{const} := \{0, 1\} \)
- \( \text{func} := \{\text{+}, \text{-}, \text{&}, \text{\&\&}, \text{\|}, \text{\|\|}\} \)

**Tool Flow:**

1. BMC on software-netlist is on average \( >2 \times \) faster than BMC on bit-level netlist and word-level RTL model.

2. For unbounded verification, software k-induction is faster and solves more safe instances than k-induction for bit-level netlist.

3. Software PDR and bit-level PDR times are comparable for detecting deep bugs.

### PROPOSED TECHNIQUE

**Software Verification Algorithms**

- BMC
- Symbolic Execution
- SAT
- SMT
- Interpolation
- Interpretation

**Abstract Interpretation view of CDCL**

**REFERENCES**

[1] R. Mukherjee, D. Kroening, T. Melham. Hardware Verifi-


driven learning. In POPL '13

[4] P. Boese. A practical approach to word level model check-
ing of instruction netlist. In CAV '08

**RESULTS**

1. BMC on software-netlist is on average \( >2 \times \) faster than BMC on bit-level netlist and word-level RTL model.

2. For unbounded verification, software k-induction is faster and solves more safe instances than k-induction for bit-level netlist.

3. Software PDR and bit-level PDR times are comparable for detecting deep bugs.