Bit-Blasting Meets Local Search in Bitwuzla

Aina Niemetz and Mathias Preiner

Shonan Meeting 180, October 2–5, 2023
A new, specialized SMT Solver

- for the **quantified** and **quantifier-free** theories of
  - fixed-size bit-vectors, floating-point arithmetic, arrays, and uninterpreted functions
  - **Focus:** theories primarily used in **hardware verification**

**Selected Features:**

- **Full incremental** support
- Seamless interaction between **multiple solver instances**
- Models, unsat cores/assumptions
- Comprehensive and easy-to-use **APIs** (C++, C, Python, OCaml)
- **Input Formats:** SMT-LIBv2, BTOR2

**Pronounced as** “bitvootslah”
- Derived from an Austrian dialect expression for **someone who tinkers with bits**.

- Bitwuzla considered **superior successor** of Boolector
History

Boolector

- An award-winning SMT solver, but . . .
  - Specialized, tight integration of bit-vectors with arrays
  - Monolithic C code base, rigid architecture
- Cumbersome to maintain, adding new features difficult

Bitwuzla

- Started as an improved and extended fork of Boolector in 2018
  - Floating-point arithmetic, local search procedure, unsat cores, . . .
  - No official release, limitations of Boolector remained
- In 2022, code base discarded and rewritten from scratch
- Written in C++, inspired by techniques in Boolector
Solver Engine

- Maintains a theory solver for each supported theory
- **Quantifiers module** implemented as theory solver
- **Distributes relevant terms** to theory solvers
- **Processes lemmas** generated by theory solvers
- **Model-based** theory combination

- Implements lazy SMT paradigm **lemmas on demand**

- **Bit-vector abstraction** of formula (instead of **propositional**)
  - Bit-vector solver at its **core**
  - BV solver reasons about **Boolean and bit-vector terms**
  - Non-BV theory atoms abstracted as **Boolean constant**
  - BV terms with non-BV operator abstracted as **bit-vector constant**
Bit-Vector Solver

**Bit-Blast Solver**
- BV terms $\rightarrow$ AIG circuits (+rewriting [BB’06]) $\rightarrow$ CNF
- CaDiCaL (default), Kissat (non-incremental)
- SAT solver used as a black box (*no IPASIR-UP*)

**Propagation-Based Local Search Solver** (sat only)
- **Ternary propagation-based local search** [FMCAD’20]
- extension with **bound tightening**
- **no SAT solver**

**Three Configuration Modes**
- Bit-blasting only
- Local search only
- **Combination** of both approaches (**challenge: how to share information**)
\[(x \ll 001) \geq_s 000 \land x <_u 100 \land (x \cdot 010) \mod 011 = x + 001\]

sat: \(x = 001\)

- constants, variables: 010, 2\([3]\), \(x[3]\)
- bit-vector operators: \(<_u, >_s, \sim, \&, >>, >>_a, \circ, [:], +, :, \div, \ldots\)
- arithmetic operators modulo \(2^n\) (overflow semantics!)
Bit-Blasting

- current **state-of-the-art** for quantifier-free bit-vector formulas
- rewriting + simplifications + **eager** reduction to SAT
- BV terms » AIG circuit » CNF
- **efficient** in practice
- may suffer from an **exponential** blow-up in the formula size
- **may not scale well** with increasing bit-width

Example: 
Bit-Blasting

- current **state-of-the-art** for quantifier-free bit-vector formulas
- rewriting + simplifications + **eager reduction** to SAT
- BV terms ⇒ AIG circuit ⇒ CNF

- **efficient** in practice
- may suffer from an **exponential** blow-up in the formula size

- **may not scale well** with increasing bit-width

Bit-Blasting

- current **state-of-the-art** for **quantifier-free** bit-vector formulas
- rewriting + simplifications + **eager reduction** to SAT
- BV terms $\triangleright$ AIG circuit $\triangleright$ CNF
- efficient in practice
- may suffer from an **exponential** blow-up in the formula size
- may not scale well with increasing bit-width

Example \( x_{[32]} \times y_{[32]} = z_{[32]} \)
without bit-blasting (orthogonal approach)

lifts concept of backtracing from ATPG to the word-level

not able to determine unsatisfiability

Probabilistically Approximately Complete (PAC) [Hoos, AAAI'99]

guaranteed to find a solution if there is one
assume satisfiability, start with **initial assignment**

**propagate** target values towards inputs

- invertibility conditions
- inverse value computation
- weaker notion: **consistency** condition, **consistent** value computation

iteratively improve current state until **solution** is found
Main Weaknesses:

- oblivious to constant bits
  - propagates target values that can never be assumed
  - redundant work
- too many possible candidates for value selection
  - blindly picking a candidate is bad
  - disrespects bounds implied from top-level constraints
Ternary Propagation-Based Local Search  [FMCAD’20]

- **Non-deterministic algorithm**
  - propagation path and value selection
    - multiple possible paths and values

- **Down-propagation** of values wrt. constant bits

- constant bits are **precomputed** upfront

- represented as ternary bit-vectors \( x = \langle x^{lo}, x^{hi} \rangle \)
  - \( x^{lo} \) ... minimum (unsigned) value of \( x \)
  - \( x^{hi} \) ... maximum (unsigned) value of \( x \)
  - with \((\sim x^{lo} | x^{hi}) \approx \text{ones}\) (validity condition)

  - **Example**: \( x[4] = \bullet \bullet 0 = \langle 0000, 1110 \rangle \)
    \( x[4] = \bullet \bullet 1 = \langle 0010, 1111 \rangle \)
Example. $v_4 \cdot (v_4 \& 1010) \approx 0100 \land (v_4 \& 1010) <_u 0011$
Example. \( v_4 \cdot (v_4 \& 1010) \approx 0100 \land (v_4 \& 1010) <_u 0011 \)
Example. $v_{[4]} \cdot (v_{[4]} \& 1010) \approx 0100 \land (v_{[4]} \& 1010) <_u 0011$
Bound Tightening

- **too many** possible candidates for value selection
  - especially for disequality, inequalities, bit-wise operators
  - especially for large(r) bit-widths

- **compute bounds**
  - for $x$ in $x \odot s$ ($s \odot x$)
  - implied by satisfied **top-level** inequalities $\{<s, \geq s, <u, \geq u\}$

- **define invertibility conditions** wrt. to min/max bounds
  - $IC(x, x <_u s \approx t) =$
    - $t \approx 1 \Rightarrow (s \not\approx 0 \land x^{lo} <_u s) \land t \approx 0 \Rightarrow (x^{hi} \geq_u s)$
    - $t \approx 1 \Rightarrow (\text{min}_u(x) <_u s \land x^{lo} <_u s) \land t \approx 0 \Rightarrow (x^{hi} \geq_u s) \land \text{max}_u(x) \geq_u s$
  - affects path selection (essential input condition)

- **consistency conditions** remain unchanged
  - IC with respect to current assignment
  - CC independent of the current assignment
Ternary Propagation-Based Local Search + Bound Tightening

- implemented in our **new** LS library, integrated in **Bitwuzla**

- **base** Prop.-based LS [CAV’16]

- **constbits** Ternary prop.-based LS [FMCAD’20]
  - + 223 (median) instances vs. **base**

- **bounds** **constbits** with **bound tightening**
  - for majority of operators
  - + 239 (median) instances vs. **constbits**

- 14,639 QF.BV sat instances in SMT-LIB
- 10 runs with different seeds for RNG
- 30s time limit, 8GB memory limit
Sequential Portfolio Combination

- Lingeling SAT back end
- CryptoMiniSat SAT back end
- Kissat SAT back end
- CaDiCaL SAT back end

- **sequential portfolio**
  (first run LS, then fall back to bit-blasting)

- **all** 41,713 benchmarks in SMT-LIB QF_BV
- **1200s** time limit, **8GB** memory limit
Challenge: Hybrid Combination

Local Search Solver » Bit-Blast Solver

- **Utilize assignment** of local search solver to give the SAT solver a **head start**
- **Which assignment** should we use?
  - **Last** assignment (before the LS solver gave up)
  - **Best** assignment (largest number of roots satisfied)
- **Which parts** of the assignment?
  - Assignment of **all** inputs
  - Assignment of inputs **under sat roots**
  - Assignment of inputs that **only appear under sat roots**
- Use API function `phase()` to set assignment of input bits
- **Problem:** seems to ”**lock in**” the phase too much
- We would need an API level ”**saved phase**” (only use the phase as a per literal starting phase, not for every decision)
Challenge: Hybrid Combination

Bit-Blast Solver » Local Search Solver

- **Seed local search solver** with last sat assignment of bit-blasting solver
- Especially interesting for **lemmas on demand**
  - Bit-vector abstraction is iteratively refined until unsat or sat and theory consistent
  - **Successful** application of sequential portfolio combination on problems where the bit-vector abstraction is already hard
- **Significantly** helps the local search engine
- **Problem:** Potentially worse than sequential portfolio in combination with other direction
- Why is the combination of both directions worse than sequential portfolio?
  - **Suspicion:** "too many" iterative calls are solved by local search and SAT solver has to "catch up" without the benefit of small incremental calls
  - We need to be able to **seed the bit-blasting solver** with the local search assignment **without** decreasing performance of the SAT solver
Conclusion

Bitwuzla

▶ A new state-of-the-art SMT solver for all things bits (and more)
▶ Source code: https://github.com/bitwuzla/bitwuzla
▶ Website and Documentation: https://bitwuzla.github.io

Ternary Propagation-Based Local Search

▶ great complementary technique to bit-blasting
  ▶ constant bits information helps avoid redundant work
  ▶ bound tightening extremely promising
    ○ work in progress
    ○ current (limited) support yields significant improvement
▶ implemented as local search library
  ▶ allows solver-independent integration
▶ Challenge: Hybrid approach
  ▶ share information between bit-blasting and local search
A. Niemetz and M. Preiner. *Bitwuzla*.

A. Niemetz and M. Preiner. *Ternary Propagation-Based Local Search for more Bit-Precise Reasoning*.


