Bit-Blasting Meets Local Search in Bitwuzla

Aina Niemetz and Mathias Preiner

Shonan Meeting 180, October 2–5, 2023

Stanford University

Stanford | Center for Automated Reasoning

Bitwuzla [CAV'23]

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

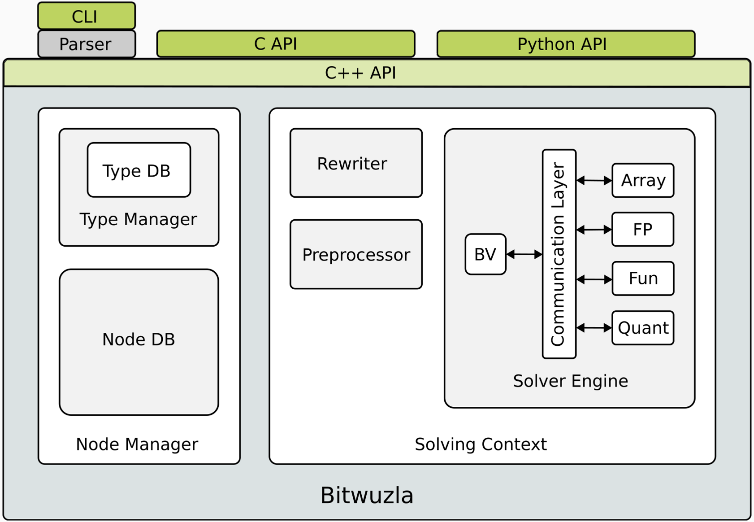
Boolector

- 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**

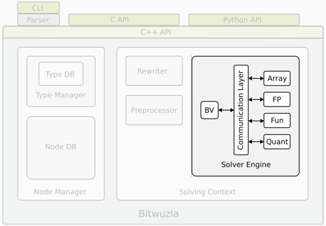
Bitwuzla

- 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

Architecture



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 » AIG circuits (+rewriting [BB'06]) » 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**)

Theory of Fixed-Size Bit-Vectors

$$(x \ll 001) \geq_s 000 \;\land\; x <_u 100 \;\land\; (x \cdot 010) \bmod 011 \;=\; x + 001$$

sat: $x = 001$

- constants, variables: $010,\;\; 2_{[3]},\;\; x_{[3]}$
- bit-vector operators: $<_u,\; >_s,\; \sim,\; \&,\; \gg,\; \gg_a,\; \circ,\; [:],\; +,\; \cdot,\; \div,\; \ldots$
- arithmetic operators modulo 2^n (overflow semantics!)

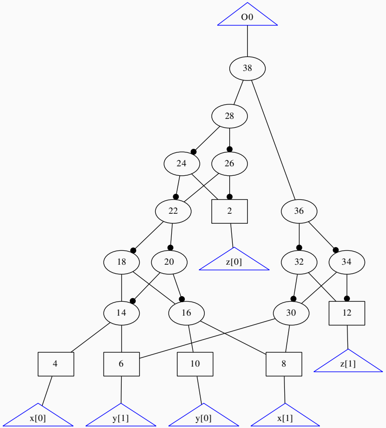
Bit-Blasting

- 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 $x_{[2]} * y_{[2]} = z_{[2]}$



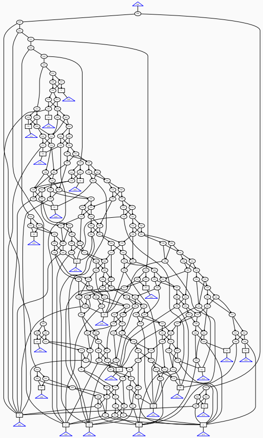
Bit-Blasting

- 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 $x_{[8]} * y_{[8]} = z_{[8]}$



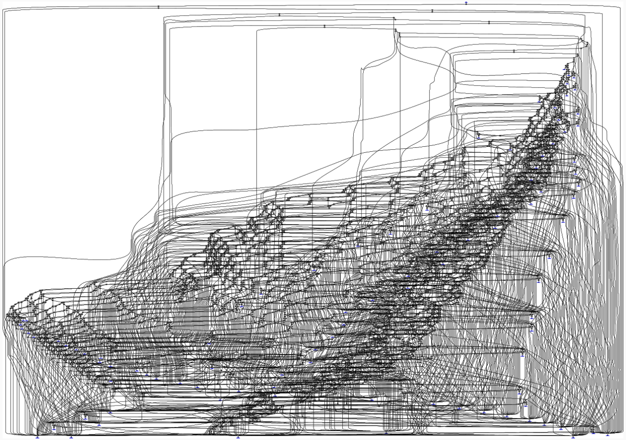
Bit-Blasting

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

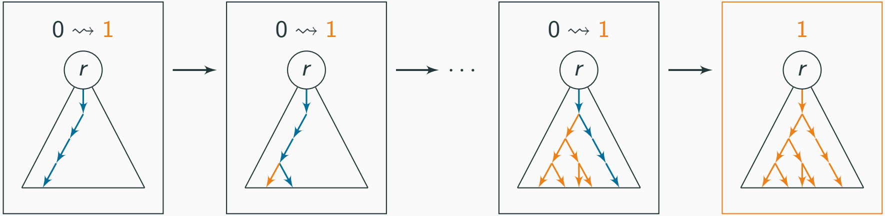
- **efficient** in practice
- may suffer from an **exponential** blow-up in the formula size

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

Example $x_{[32]} * y_{[32]} = z_{[32]}$



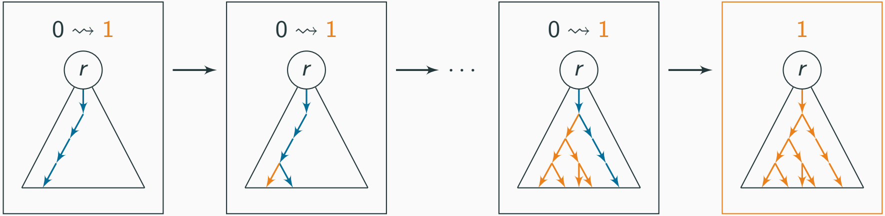
Propagation-Based Local Search [CAV'16]



- **without** bit-blasting (**orthogonal** approach)
- lifts concept of **backtracing** from ATPG to the **word-level**

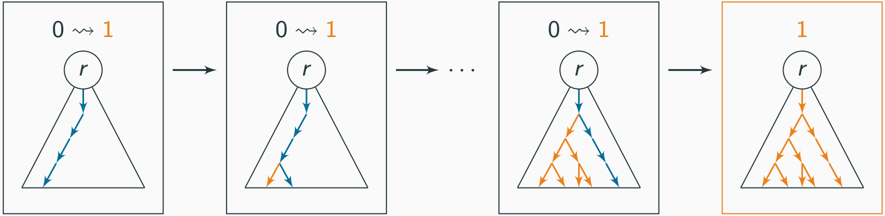
- **not able** to determine **unsatisfiability**
- **Probabilistically Approximately Complete (PAC)** [Hoos, AAAI'99]
  - guaranteed to find a solution if there is one

Propagation-Based Local Search [CAV'16]



- 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

Propagation-Based Local Search [CAV'16]



- **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]

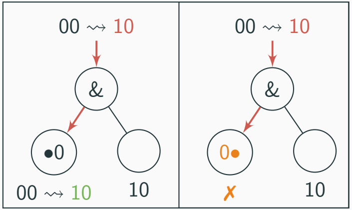
- **Non-deterministic algorithm**

  ⇩

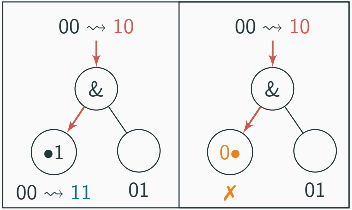
  - **propagation path** and **value selection**
    - multiple possible paths and values

- **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 $(\sim x^{lo} \mid x^{hi}) \approx$ ones (validity condition)
  - **Example:** $x_{[4]} = \bullet\bullet\bullet 0 = \langle 0000, 1110 \rangle$
    $x_{[4]} = \bullet\bullet 1 \bullet = \langle 0010, 1111 \rangle$



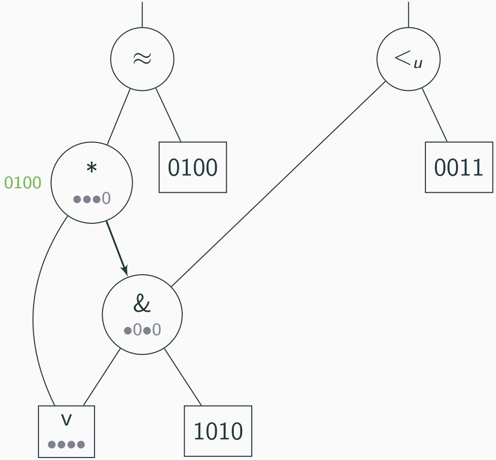
inverse value



consistent value

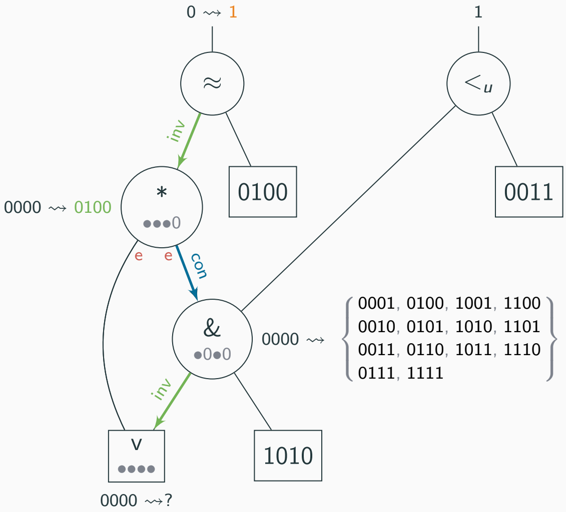
Ternary Propagation-Based Local Search + Bound Tightening

Example. $v_{[4]} \cdot (v_{[4]} \,\&\, 1010) \approx 0100 \wedge (v_{[4]} \,\&\, 1010) <_u 0011$



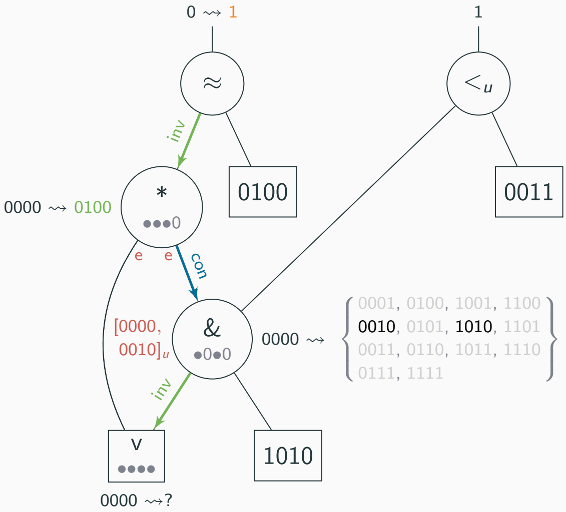
Ternary Propagation-Based Local Search + Bound Tightening

Example. $v_{[4]} \cdot (v_{[4]} \mathbin{\&} 1010) \approx 0100 \land (v_{[4]} \mathbin{\&} 1010) <_u 0011$



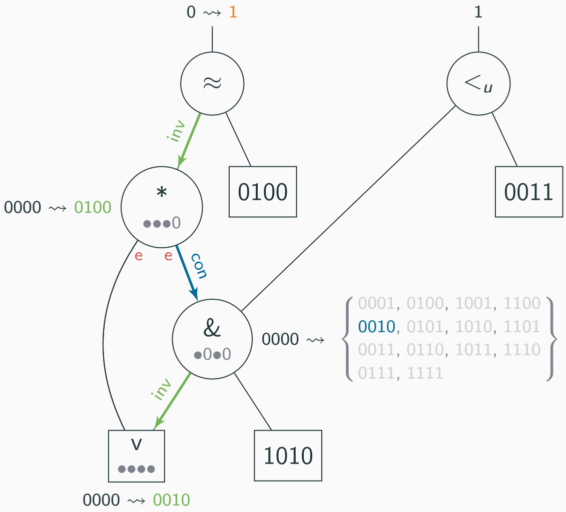
Ternary Propagation-Based Local Search + Bound Tightening

Example. $v_{[4]} \cdot (v_{[4]} \,\&\, 1010) \approx 0100 \wedge (v_{[4]} \,\&\, 1010) <_u 0011$



Ternary Propagation-Based Local Search + Bound Tightening

Example. $v_{[4]} \cdot (v_{[4]} \mathbin{\&} 1010) \approx 0100 \wedge (v_{[4]} \mathbin{\&} 1010) <_u 0011$



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 $x \diamond s$ ($s \diamond x$)
  - implied by **satisfied** **top-level** inequalities $\{<_s, \geq_s, <_u, \geq_u\}$

- define **invertibility conditions** wrt. to min/max bounds
  - $IC(x, x <_u s \approx t) =$

    $t \approx 1 \Rightarrow (s \not\approx 0 \qquad \wedge x^{lo} <_u s) \wedge t \approx 0 \Rightarrow (x^{hi} \geq_u s)$

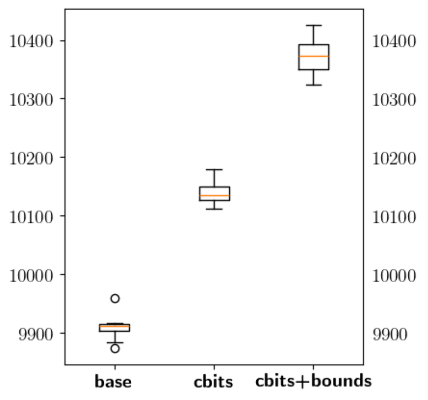
    $\Downarrow$

    $t \approx 1 \Rightarrow (min_u(x) <_u s \wedge x^{lo} <_u s) \wedge t \approx 0 \Rightarrow (x^{hi} \geq_u s) \wedge max_u(x) \geq_u s$
  - 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

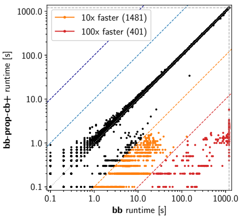
- implemented in our **new** LS library, integrated in **Bitwuzla**



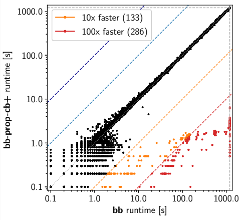
- **base** Prop.-based LS [CAV'16]
- **constbits** Ternary prop.-based LS [FMCAD'20]
  - + 223 (median) instances vs. **base**
- **bounds** **constbits** with bound tightening
  - for majority of operators
  - + 239 (median) instances vs. **constbits**

- 14,639 QF_BV **sat** instances in SMT-LIB
- 10 runs with different seeds for RNG
- 30s time limit, 8GB memory limit

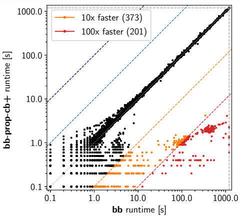
Sequential Portfolio Combination



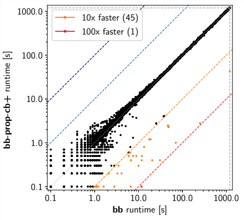
Lingeling SAT back end



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 satsified)
- **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

References i

- A. Niemetz and M. Preiner. *Bitwuzla*.
- A. Niemetz and M. Preiner. *Ternary Propagation-Based Local Search for more Bit-Precise Reasoning.*
- M. Brain, A. Niemetz, M. Preiner, A. Reynolds, C. Barrett and C. Tinelli. Invertibility Conditions for Floating-Point Formulas. In Proc. of CAV'19, pages 116–136, Springer, 2019.
- A. Niemetz, M. Preiner, A. Reynolds, C. Barrett and C. Tinelli. Solving Quantified Bit-Vectors Using Invertibility Conditions. In Proc. of CAV'18, pages 236–255, Springer, 2018.
- A. Niemetz, M. Preiner and A. Biere. *Precise and Complete Propagation Based Local Search for Satisfiability Modulo Theories.* In Proc. of CAV'16, pages 199–217, Springer, 2016.

References ii

- R. Brummayer, A. Biere. *Local Two-Level And-Inverter Graph Minimization without Blowup.* In Proc. of MEMICS'06, 2006.
- H. H. Hoos. *On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT.* In Proc. of AAAI/IAAI'99, pages 661–666, AAAI Press / The MIT Press, 1999.