Local Search for Bit-Precise Reasoning and Beyond

Aina Niemetz and Mathias Preiner

Shonan Meeting 180, October 2–5, 2023

Stanford University

Stanford | Center for Automated Reasoning

Bitwuzla

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

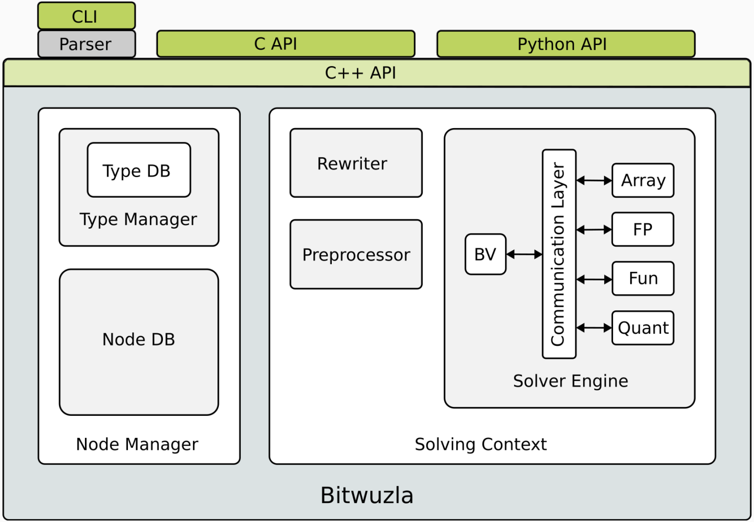
Boolector

- 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**

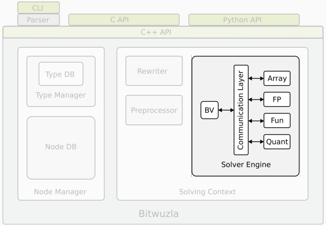
Bitwuzla

- 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 $\rightarrow$ AIG circuits (+rewriting [?]) $\rightarrow$ 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**)

Theory of Fixed-Size Bit-Vectors

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

$$\text{sat: } x = 001$$

- 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

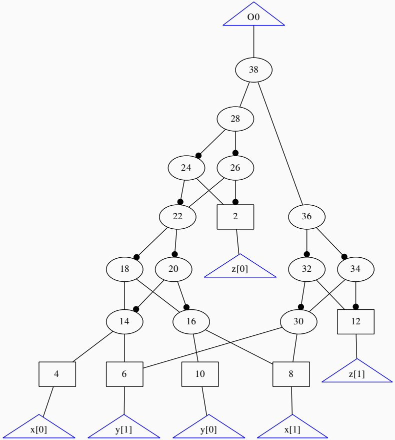
- current state-of-the-art for **quantifier-free** bit-vector formulas
- rewriting + simplifications + eager reduction to SAT

⇩

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



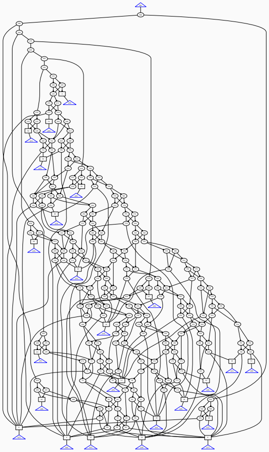
Bit-Blasting

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

- current state-of-the-art for **quantifier-free** bit-vector formulas
- rewriting + simplifications + eager reduction to SAT

circuit ⟹ CNF

- efficient in practice
- may suffer from an exponential blow-up in the formula size
- may not scale well with increasing bit-width



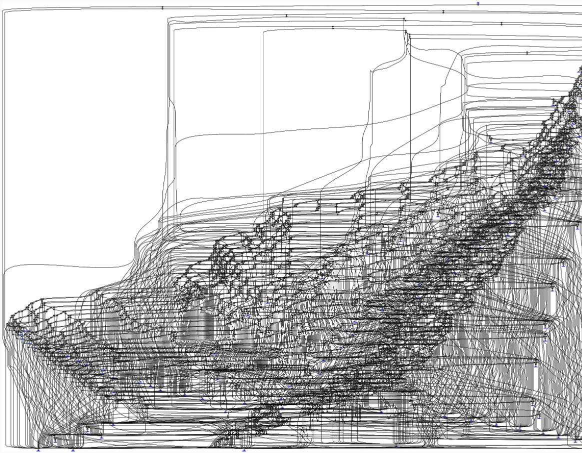
Bit-Blasting

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

- current state-of-the-art for **quantifier-free** bit-vector formulas
- rewriting + simplifications + eager reduction to SAT

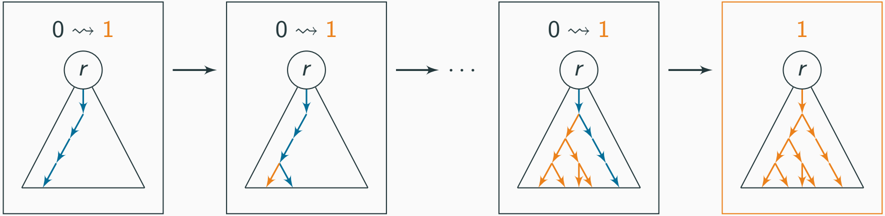
⇩

circuit ⟹ CNF



- efficient in practice
- may suffer from an exponential blow-up in the formula size
- may not scale well with increasing bit-width

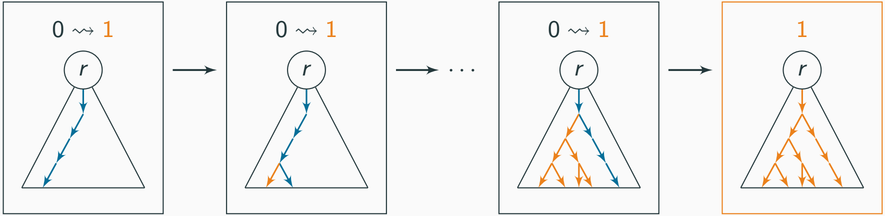
Propagation-Based Local Search



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

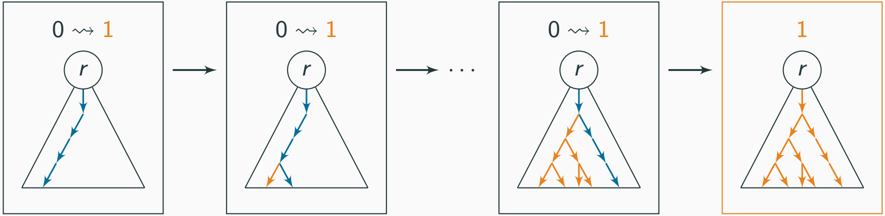
- **not able** to determine **unsatisfiability**
- Probabilistically Approximately Complete (PAC) [Hoos, AAAI'99]
  - guaranteed to find a solution if there is one

Propagation-Based Local Search

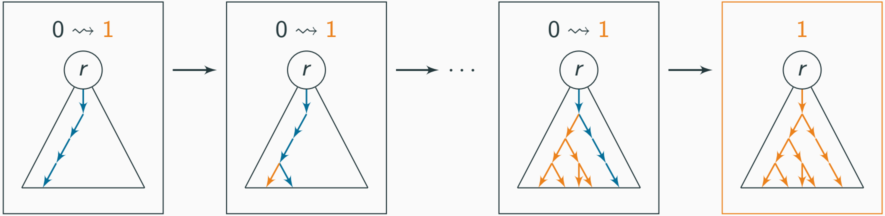


- assume satisfiability, start with **initial assignment**
- **propagate** target values towards inputs
  - invertibility conditions
  - inverse value computation
  - weaker notion: consistency condition, consistent value computation
- iteratively improve current state until **solution** is found

Propagation-Based Local Search



- **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



- **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**

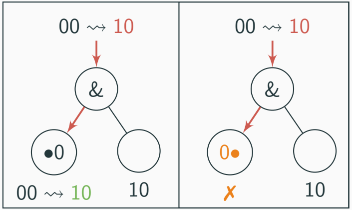
  ⇩

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

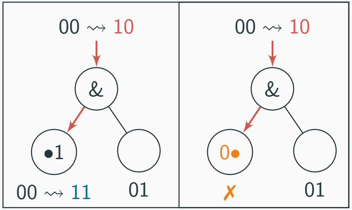
- **Down-propagation** of values wrt. **constant bits**

- 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:** $x_{[4]} = \bullet\bullet\bullet 0 = \langle 0000, 1110 \rangle$
    $x_{[4]} = \bullet\bullet 1 \bullet = \langle 0010, 1111 \rangle$



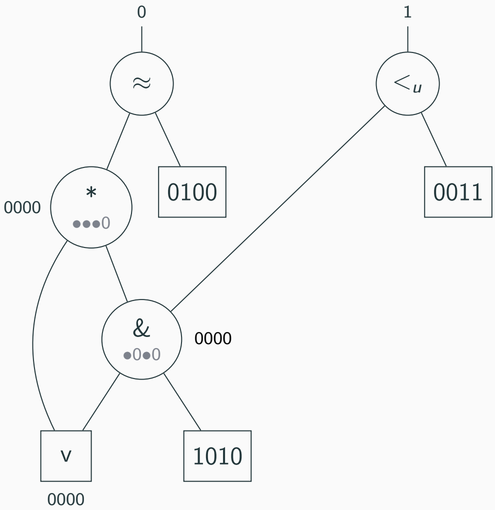
inverse value



consistent value

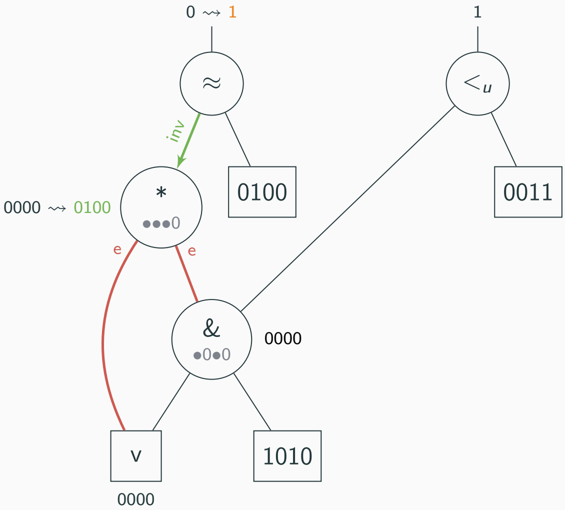
Ternary Propagation-Based Local Search++

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



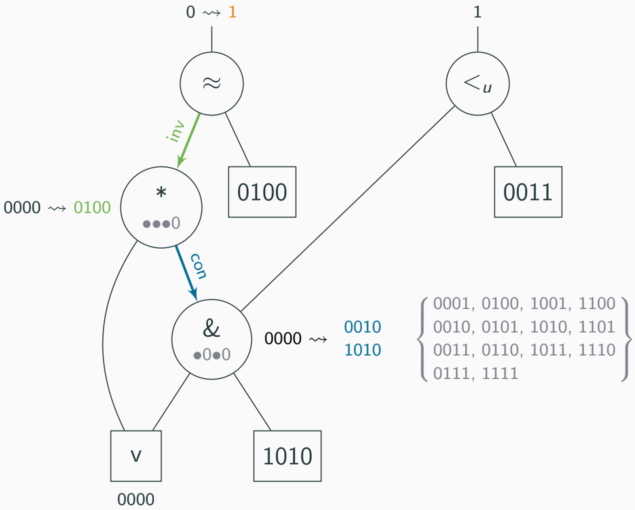
Ternary Propagation-Based Local Search++

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



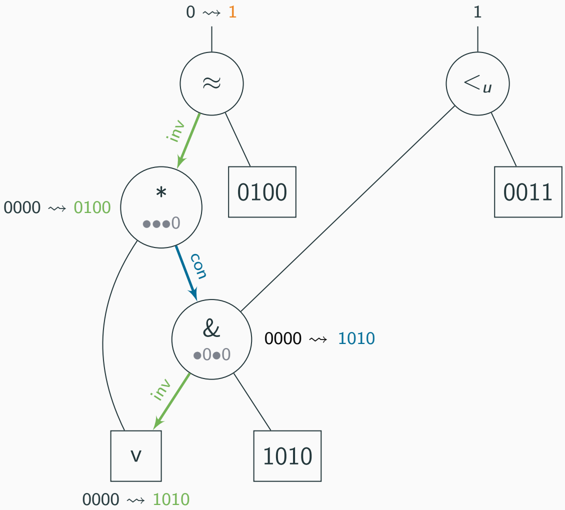
Ternary Propagation-Based Local Search++

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



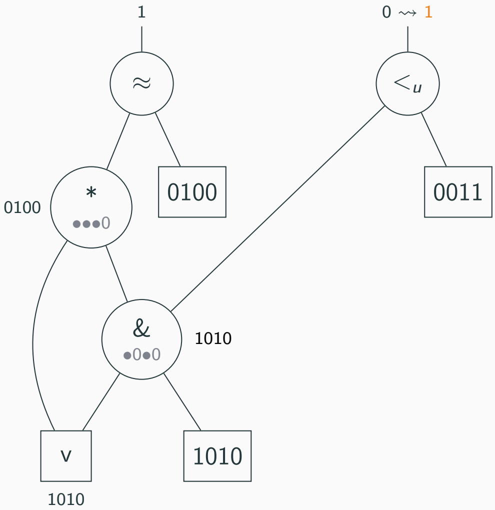
Ternary Propagation-Based Local Search^{++}

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



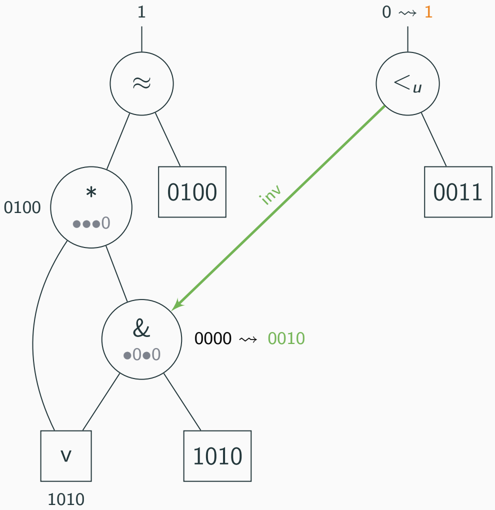
Ternary Propagation-Based Local Search++

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

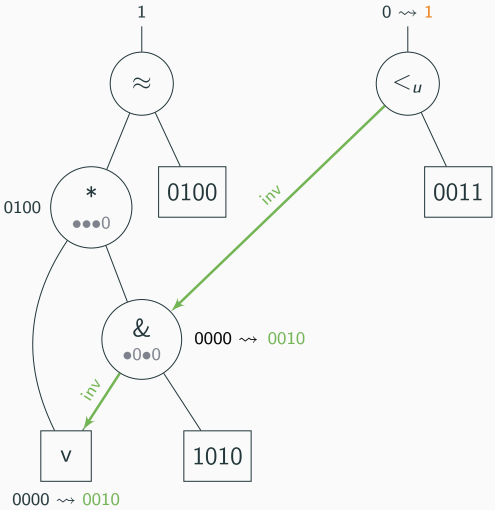


Ternary Propagation-Based Local Search^{++}

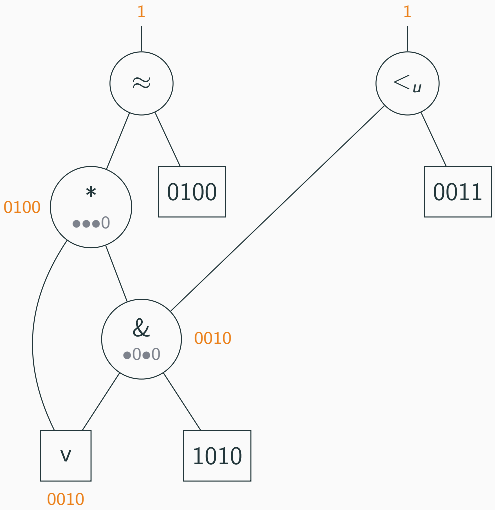
Example. $v_{[4]} \cdot (v_{[4]} \,\&\, 1010) \approx 0100 \wedge (v_{[4]} \,\&\, 1010) <_u 0011$



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

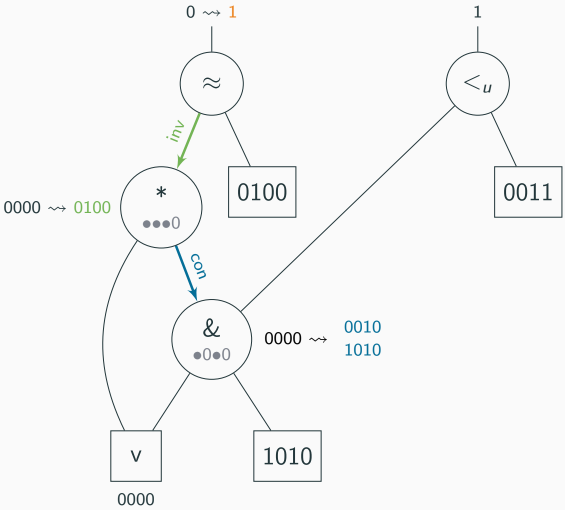


Example. $v_{[4]} \cdot (v_{[4]} \mathbin{\&} 1010) \approx 0100 \wedge (v_{[4]} \mathbin{\&} 1010) <_u 0011$



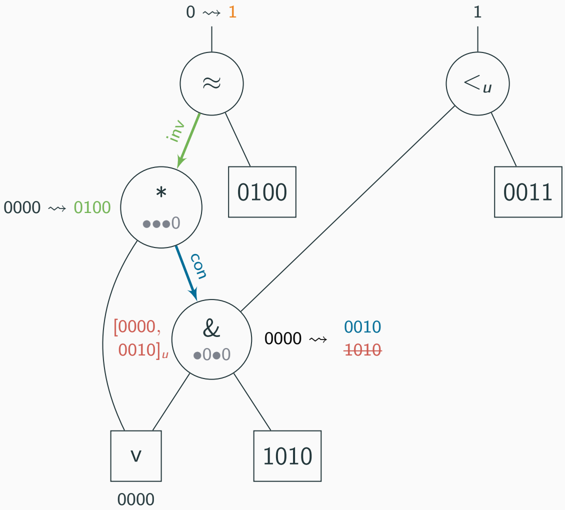
Ternary Propagation-Based Local Search^{++}

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



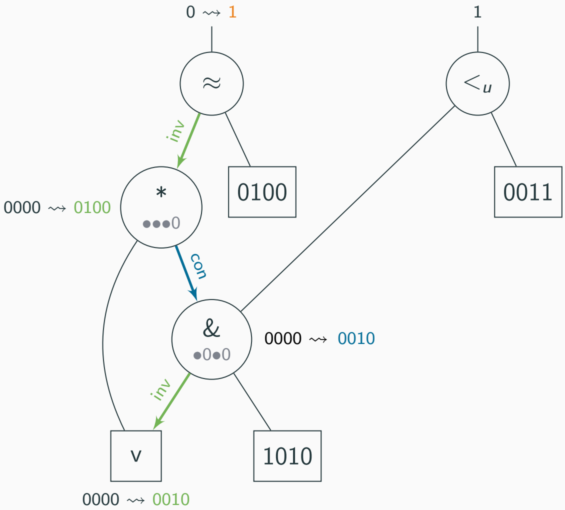
Ternary Propagation-Based Local Search++

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



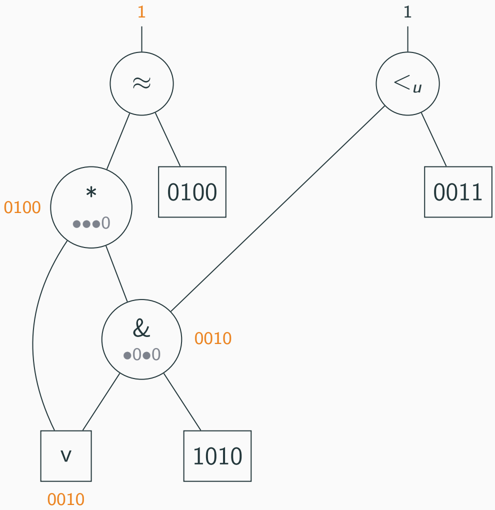
Ternary Propagation-Based Local Search^{++}

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



Ternary Propagation-Based Local Search++

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



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

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

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

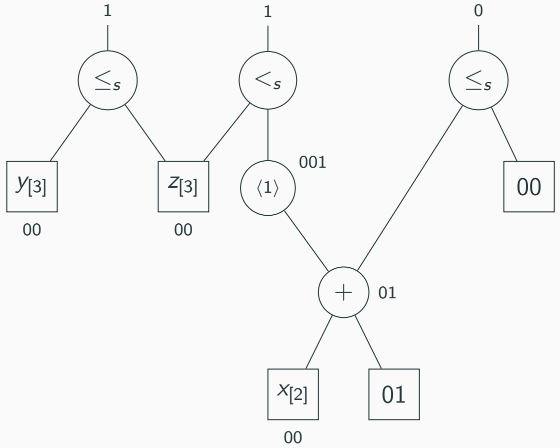
    $$\Downarrow$$

    $$t \approx 1 \Rightarrow (min_u(x) <_u s \wedge x^{lo} <_u s) \wedge t \approx 0 \Rightarrow (x^{hi} \geq_u s) \wedge max_u(x) \geq_u s$$
  - affects path selection (essential input condition)

- **consistency conditions** remain unchanged
  - IC **with respect to** current assignment
  - CC **independent** of the current assignment

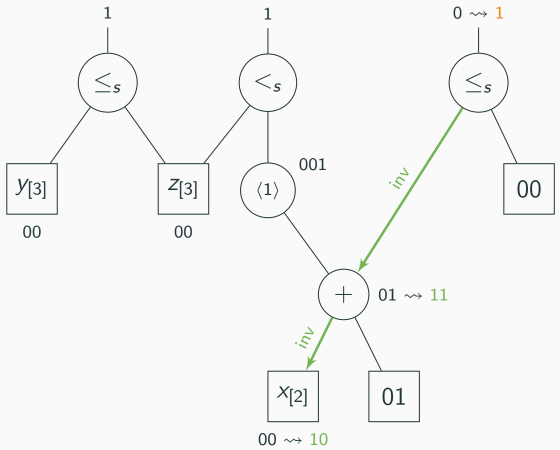
Bound Tightening: Why not always select essential paths?

Example. $y_{[3]} \leq_s z_{[3]} \wedge z_{[3]} <_s (x_{[2]} + 01)\langle 1 \rangle \wedge (x_{[2]} + 01) \leq_s 00$



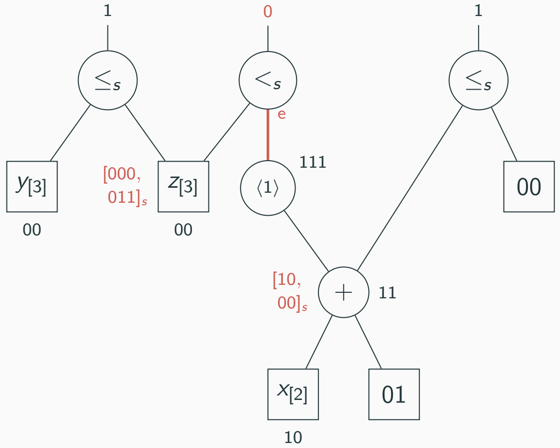
Bound Tightening: Why not always select essential paths?

Example. $y_{[3]} \leq_s z_{[3]} \wedge z_{[3]} <_s (x_{[2]} + 01)\langle 1 \rangle \wedge (x_{[2]} + 01) \leq_s 00$



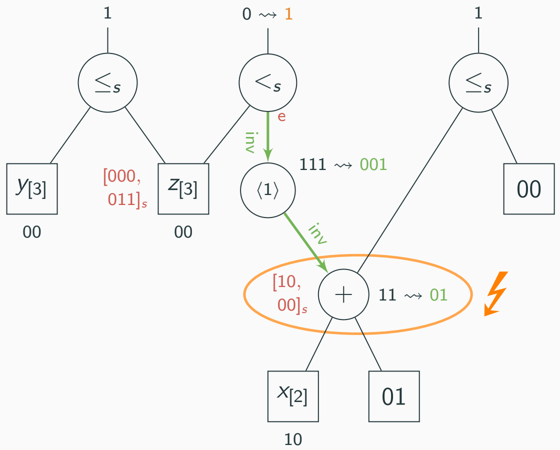
Bound Tightening: Why not always select essential paths?

Example. $y_{[3]} \leq_s z_{[3]} \wedge z_{[3]} <_s (x_{[2]} + 01)\langle 1 \rangle \wedge (x_{[2]} + 01) \leq_s 00$



Bound Tightening: Why not always select essential paths?

Example. $y_{[3]} \leq_s z_{[3]} \land z_{[3]} <_s (x_{[2]} + 01)\langle 1 \rangle \land (x_{[2]} + 01) \leq_s 00$

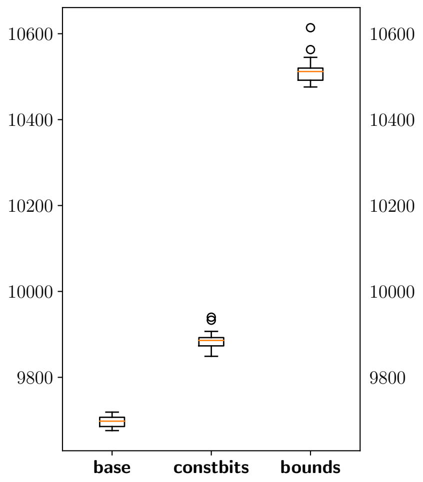


misc

- with bounds incomplete with only essential paths selected
- no bounds path sel bad, essential check does not match with relatively
- future:
  - cc modulo const bits in x?
  - ic modulo const bits in s!

Results

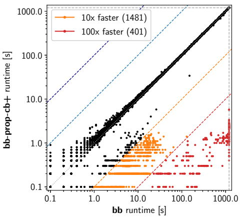
- implemented in our **new** LS library, integrated in **Bitwuzla**



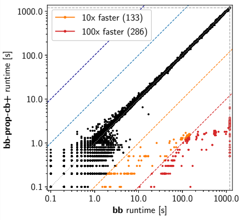
- **base** Prop.-based LS [CAV'16]
- **constbits** Ternary prop.-based LS [FMCAD'20]
  - + 188 (median) instances vs. **base**
- **bounds** **constbits** with bound tightening
  - for operators $<_s$, $<_u$, &
  - + 626 (median) instances vs. **constbits**

- 14,639 QF_BV **sat** instances in SMT-LIB
- 20 runs with different seeds for RNG
- 60s time limit, 8GB memory limit

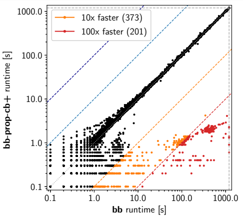
Results



Lingeling SAT back end

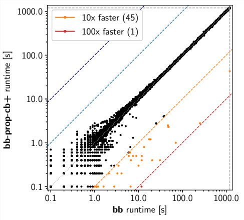


CryptoMiniSat SAT back end



Kissat SAT back end

SAT competition winner 2020



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
- winner of division QF_BV
  in the SMT-COMP 2020

Conclusion

- 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
- new **local search library** (under development)
  - allows solver-independent integration
- **Future work:** Hybrid approach
  - share information between bit-blasting and local search

Beyond Local Search

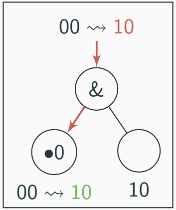
- **symbolic invertibility and consistency conditions**
  - synthesized manually and with SyGuS
    - w/o const bits: 50% with SyGuS (only IC)
    - with const bits: 5% with SyGuS (IC + CC)
  - for a representative set of bit-vector operators
  - verified up to size 65
  - **Future work:** proofs of correctness
- **invertibility conditions** for **quantifier instantiation** [CAV'18]
  - default approach for BV in CVC4/cvc5
  - SMT-COMP winner 2018, 2020, 2021
- **invertibility conditions** for **FP** [CAV'19]
  - **Future Work:** propagation-based LS for FP

References

- 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.
- 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.

Propagation Value Selection

Given $x \diamond s \approx t$ with $x = \langle x^{lo}, x^{hi} \rangle$ **ternary**, and s, t **binary** bit-vectors.



inverse value

Invertibility Condition (IC)

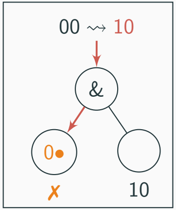
$$\forall x, s, t. \, (IC(x, x \diamond s \approx t) \Leftrightarrow \exists y. \, (y \diamond s \approx t \wedge mcb(x, y)))$$

$$IC(x, x \,\&\, s \approx t) = t \,\&\, s \approx t$$
$$\wedge((s \,\&\, x^{hi}) \,\&\, \sim(x^{lo} \oplus x^{hi})) \approx (t \,\&\, \sim(x^{lo} \oplus x^{hi}))$$

$mcb(x, y)$ . . . **true** if constant bits in x match corresponding bits in y (x ternary, y binary)

Propagation Value Selection

Given $x \diamond s \approx t$ with $x = \langle x^{lo}, x^{hi} \rangle$ **ternary**, and s, t **binary** bit-vectors.



inverse **value**

Invertibility Condition (IC)

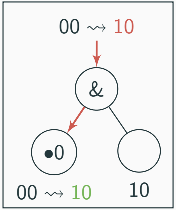
$$\forall x, s, t. \, (IC(x, x \diamond s \approx t) \Leftrightarrow \exists y. \, (y \diamond s \approx t \wedge mcb(x, y)))$$

$$IC(x, x \,\&\, s \approx t) = t \,\&\, s \approx t$$
$$\wedge((s \,\&\, x^{hi}) \,\&\, {\sim}(x^{lo} \oplus x^{hi})) \approx (t \,\&\, {\sim}(x^{lo} \oplus x^{hi}))$$

$mcb(x, y)$ . . . **true** if constant bits in x match corresponding bits in y (x ternary, y binary)

Propagation Value Selection

Given $x \diamond s \approx t$ with $x = \langle x^{lo}, x^{hi} \rangle$ **ternary**, and s, t **binary** bit-vectors.

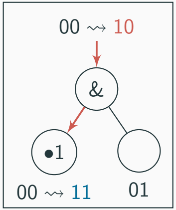


inverse value

Invertibility Condition (IC)

$$\forall x, s, t.\, (IC(x, x \diamond s \approx t) \Leftrightarrow \exists y.\, (y \diamond s \approx t \wedge mcb(x, y)))$$

$$IC(x, x \,\&\, s \approx t) = t \,\&\, s \approx t$$
$$\wedge ((s \,\&\, x^{hi}) \,\&\, {\sim}(x^{lo} \oplus x^{hi})) \approx (t \,\&\, {\sim}(x^{lo} \oplus x^{hi}))$$



consistent value

Consistency Condition (CC)

$$\forall x, t.\, (CC(x, x \diamond s \approx t) \Leftrightarrow \exists y, s.\, (y \diamond s \approx t \wedge mcb(x, y)))$$

$$CC(x, x \,\&\, s \approx t) = t \,\&\, x^{hi} \approx t$$

$mcb(x, y)$ . . . **true** if constant bits in x match corresponding bits in y (x ternary, y binary)

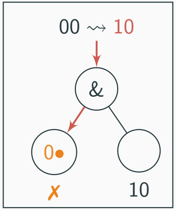
Propagation Value Selection

Given $x \diamond s \approx t$ with $x = \langle x^{lo}, x^{hi} \rangle$ **ternary**, and s, t **binary** bit-vectors.

Invertibility Condition (IC)

$$\forall x, s, t.\, (IC(x, x \diamond s \approx t) \Leftrightarrow \exists y.\, (y \diamond s \approx t \wedge mcb(x, y)))$$

$$IC(x, x \,\&\, s \approx t) = t \,\&\, s \approx t$$
$$\wedge((s \,\&\, x^{hi}) \,\&\, \sim(x^{lo} \oplus x^{hi})) \approx (t \,\&\, \sim(x^{lo} \oplus x^{hi}))$$

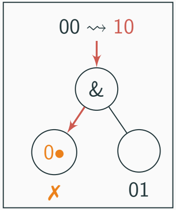


inverse value

Consistency Condition (CC)

$$\forall x, t.\, (CC(x, x \diamond s \approx t) \Leftrightarrow \exists y, s.\, (y \diamond s \approx t \wedge mcb(x, y)))$$

$$CC(x, x \,\&\, s \approx t) = t \,\&\, x^{hi} \approx t$$

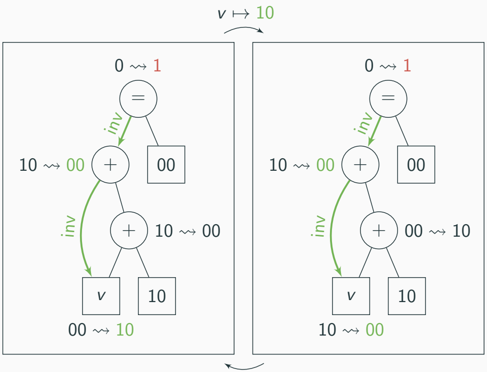


consistent value

$mcb(x, y)$ . . . **true** if constant bits in x match corresponding bits in y (x ternary, y binary)

Value Selection: Why not always select inverse values?

Example. $v_{[2]} + (v_{[3]} + 10) \approx 00$

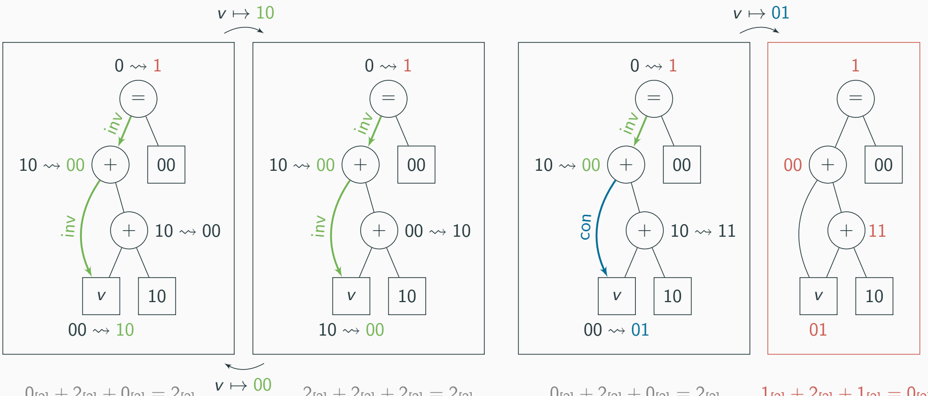


$0_{[2]} + 2_{[2]} + 0_{[2]} = 2_{[2]}$

$2_{[2]} + 2_{[2]} + 2_{[2]} = 2_{[2]}$

Value Selection: Why not always select inverse values?

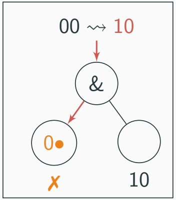
Example. $v_{[2]} + (v_{[3]} + 10) \approx 00$



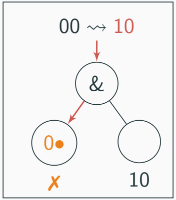
$0_{[2]} + 2_{[2]} + 0_{[2]} = 2_{[2]}$ $2_{[2]} + 2_{[2]} + 2_{[2]} = 2_{[2]}$ $0_{[2]} + 2_{[2]} + 0_{[2]} = 2_{[2]}$ $1_{[2]} + 2_{[2]} + 1_{[2]} = 0_{[2]}$

Propagation Value Selection

immediately yields the target value



inverse **value**

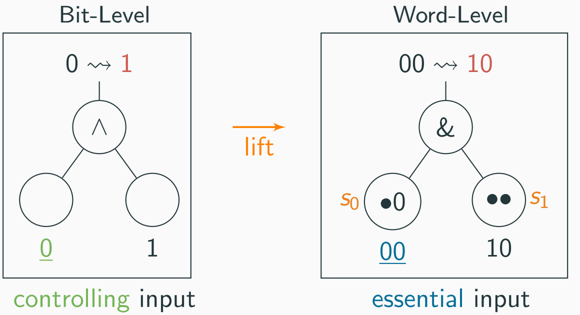


consistent **value**

yields the target value after changing other inputs

- inverse and consistent value **not always possible**
- **symbolic** invertibility/consistency conditions
  - IC: $t \,\&\, s \approx t \wedge ((s \,\&\, x^{hi}) \,\&\, \sim(x^{lo} \oplus x^{hi})) \approx (t \,\&\, \sim(x^{lo} \oplus x^{hi}))$
    $10 \,\&\, 10 \approx 10 \wedge ((10 \,\&\, 01) \,\&\, \sim(00 \oplus 01)) \approx (10 \,\&\, \sim(00 \oplus 01)) \quad = \bot$
  - CC: $t \,\&\, x^{hi} \approx t$
    $10 \,\&\, 01 \approx 10 \quad = \bot$
- **break and restart** propagation if consistency condition **false**

Propagation Path Selection



- s_0 is essential if $IC(s_1, s_0 \diamond s_1 \approx t) = \bot$
- s_1 is essential if $IC(s_0, s_0 \diamond s_1 \approx t) = \bot$

- always select essential input (if any)