Theoretical and Practical Aspects of Bit-Vector Reasoning

Andreas Fröhlich
Institute for Formal Models and Verification
Johannes Kepler University

PhD Presentation

Thursday, April 28th, 2016

Linz, Austria

JOHANNES KEPLER
UNIVERSITY LINZ

- Research Areas: Bit-Vectors, SAT, DQBF, SMT, Local Search, . . .
- List of Contributions:
  - Total of **15 publications** (14 peer-reviewed, 1 benchmark description)
  - **2 solvers** (1 publicly available)
  - **2 translation tools** (both publicly available)
  - Several challenging **benchmark families** (publicly available)
- Thesis “Theoretical and Practical Aspects of Bit-Vector Reasoning”
  - Consisting of 9 publications
  - Some **additional** unpublished complexity results

Outline

- Preliminaries
- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT
- Conclusion

Outline

- Preliminaries
- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT
- Conclusion

JOHANNES KEPLER UNIVERSITY LINZ

Bit-Vector Reasoning: Motivation

- Bit-Vector: String of bits $\{0,1\}^n$ of **fixed length** n

- Practical Applications
  - Hardware Verification
    - Natural representation of RTL specifications (e.g., VHDL, Verilog)
    - Equivalence checking or property checking (e.g., used by Intel)
  - Software Verification
    - Natural representation of datatypes
    - SAGE: Large-scale project at Microsoft

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{NEXPTIME} \subseteq \text{EXPSPACE} \subseteq \text{2-NEXPTIME} \subseteq \ldots$$

- Bounds in regard to the **input size**:
  - P: problems can be solved in polynomial time
  - NP: solutions can be checked in polynomial time
  - PSPACE: problems can be solved with polynomial space
  - NEXPTIME: solutions can be checked in exponential time
- NEXPTIME: **more succinct** representations than NP
  - Can be solved by NP algorithms after **(exponential) expansion**

Propositional domain $\{0,1\}$:

- SAT $[\exists x_1, x_2, x_3\,.] \quad (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2)$

  NP-complete

- QBF $\forall u_1 \exists e_1 \forall u_2 \exists e_2\,. \quad (u_2 \vee \neg e_1) \wedge (\neg u_1 \vee e_1) \wedge (u_1 \vee \neg e_2) \wedge (\neg u_2 \vee e_2)$

  PSPACE-complete

- DQBF $\forall u_1, u_2 \exists e_1(u_1), e_2(u_2)\,. \quad (u_2 \vee \neg e_1) \wedge (\neg u_1 \vee e_1) \wedge (u_1 \vee \neg e_2) \wedge (\neg u_2 \vee e_2)$

  NEXPTIME-complete

First-order but no functions:

- EPR $\exists a, b \forall x, y\,. \quad (p(a,x,y) \vee \neg q(y,x,b)) \wedge (q(x,b,y) \vee \neg p(y,a,x))$
  (Bernays-Schönfinkel class) NEXPTIME-complete

- QF_BV: Included in SMT-LIB
- Bit-Vector Variables: $x^{[4]}, y^{[8]}, z^{[1]}, \ldots$
- Bit-Vector Constants: $1011^{[4]}, 10011010^{[8]}, 1^{[8]}, \ldots$
- Bit-Vector Operators:
  - Bitwise: $\sim \quad \& \quad | \quad \oplus \quad \ldots$
  - Arithmetic: $+ \quad - \quad \cdot \quad / \quad \ldots$
  - Relational: $= \quad < \quad \leq \quad \ldots$
  - Shifts: $\ll \quad \gg \quad \ldots$
  - …

JOHANNES KEPLER UNIVERSITY LINZ

Outline

- Preliminaries
- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT
- Conclusion

JOHANNES KEPLER UNIVERSITY LINZ

- Running example:

$$(z = x + y) \quad \wedge \quad (z = x \ll 1) \quad \wedge \quad (x \neq y)$$

- With bit-vectors of fixed bit-width n, e.g., $n = 32$:

$$(z^{[32]} = x^{[32]} + y^{[32]}) \wedge (z^{[32]} = x^{[32]} \ll 1^{[32]}) \wedge (x^{[32]} \neq y^{[32]})$$

- Satisfiability: Are there bit-vectors, so that the formula evaluates to *true*?
  - Common solving approach:
    - **Bit-blasting** (encoding the bit-vector formula as a circuit) ...
    - ... and then using a **SAT-solver**
  - Often assumed to be NP-complete:

    "This paper addresses the satisfiability problem for bit-vector formulas: [...] It is easy to see that this problem is NP-complete."

JOHANNES KEPLER UNIVERSITY LINZ

- Complexity actually depends on the **encoding of bit-widths**

- Consider the previous example, ...

$$(z^{[n]} = x^{[n]} + y^{[n]}) \wedge (z^{[n]} = x^{[n]} \ll 1^{[n]}) \wedge (x^{[n]} \neq y^{[n]})$$

  ... with large n, e.g., $n = 1,000,000$.

- In practice: **logarithmic encoding**, e.g., SMT-LIB format

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 1000000))
(declare-fun y () (_ BitVec 1000000))
(declare-fun z () (_ BitVec 1000000))
(assert (= z (bvadd x y)))
(assert (= z (bvshl x (_ bv1 1000000))))
(assert (distinct x y))
```


JOHANNES KEPLER UNIVERSITY LINZ

Bit-Blasting is exponential

- $x^{[n]}$ can be "written down" using $\log(n)$ bits, ...

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 1000000))
(declare-fun y () (_ BitVec 1000000))
(declare-fun z () (_ BitVec 1000000))
(assert (= z (bvadd x y)))
(assert (= z (bvshl x (_ bv1 1000000))))
(assert (distinct x y))
```

- ...but bit-blasting requires n separate variables $x_0, x_1, \ldots, x_{n-1}$

bit-width	input size	bit-blasting	output size
10	223 Byte	0.0s	4.1 kB
100	227 Byte	0.0s	51.7 kB
1,000	231 Byte	0.0s	610.3 kB
10,000	235 Byte	0.9s	7.0 MB
100,000	239 Byte	14.1s	79.3 MB
1,000,000	243 Byte	167.9s	883.6 MB
10,000,000	247 Byte	...	...

- Satisfiability for QF_BV is NEXPTIME-complete [SMT'12]
- Hardness: reduction from DQBF to QF_BV
  - Use the so-called **binary magic numbers** (e.g., in Knuth—TAOCP)

$$\forall\, u_0\, u_1\, u_2 \quad \rightarrow \quad U_0^{[8]} := \begin{bmatrix} 0\\1\\0\\1\\0\\1\\0\\1 \end{bmatrix} \quad U_1^{[8]} := \begin{bmatrix} 0\\0\\1\\1\\0\\0\\1\\1 \end{bmatrix} \quad U_2^{[8]} := \begin{bmatrix} 0\\0\\0\\0\\1\\1\\1\\1 \end{bmatrix}$$

  - Eliminate dependencies:

$$\exists\, e(u_0, u_2) \quad \rightarrow \quad E^{[8]} \;\&\; {\sim}U_1^{[8]} = (E^{[8]} \ll 2^{[8]}) \;\&\; {\sim}U_1^{[8]}$$

- Complexity depends on the encoding ...
- ...but also on the **set of operators**: [CSR'13]
  - QF_BV$_{\ll}$ (only bitwise operations, equality, and left shift)

    QF_BV$_{\ll}$ is NEXPTIME-complete
  - QF_BV$_{\ll_1}$ (only bitwise operations, equality, and left shift by one)

    QF_BV$_{\ll_1}$ is PSPACE-complete
  - QF_BV$_{bw}$ (only bitwise operations and equality)

    QF_BV$_{bw}$ is NP-complete

Outline

- Preliminaries
- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT
- Further research
- Conclusion (Summary, Impact, Future Work)

JOHANNES KEPLER UNIVERSITY LINZ

- Consider the previous example:

$$(z^{[n]} = x^{[n]} + y^{[n]}) \wedge (z^{[n]} = x^{[n]} \ll 1^{[n]}) \wedge (x^{[n]} \neq y^{[n]})$$

- Can we do better than bit-blasting?
  - $+$ can be expressed by $\oplus \quad | \quad \& \quad = \quad \ll_1$

$$(z^{[n]} = x^{[n]} \oplus y^{[n]} \oplus c_{in}^{[n]}) \wedge \Big(c_{out}^{[n]} = (x^{[n]} \ \& \ y^{[n]}) \mid \Big((c_{in}^{[n]} \ll 1^{[n]}) \ \& \ (x^{[n]} \mid y^{[n]})\Big)\Big)$$

  - The example is in QF_BV$_{\ll_1}$

    $\rightarrow$ can be solved in PSPACE

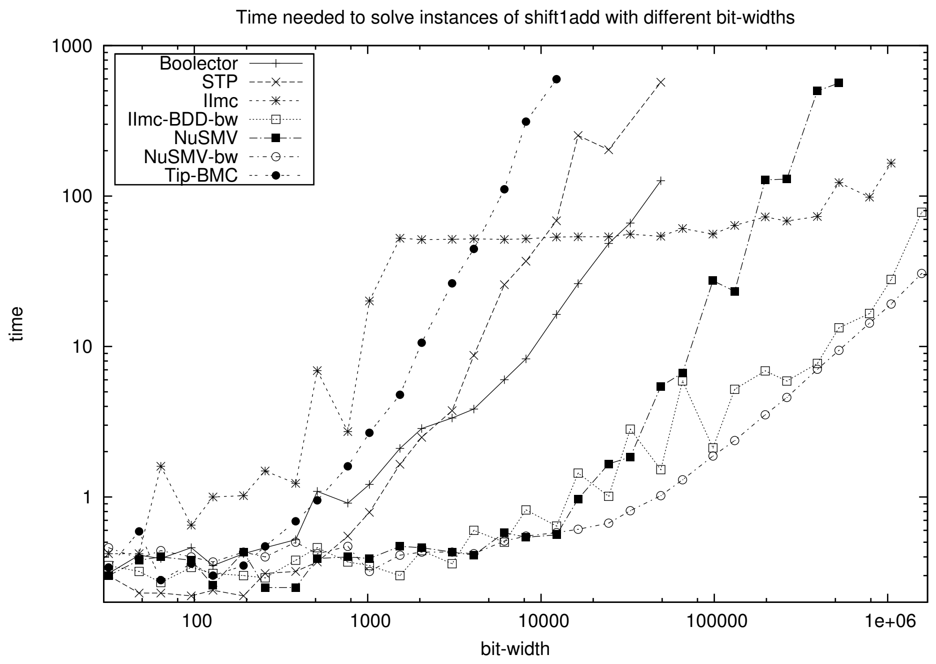
- **Polynomial** encoding as a **model checking** problem

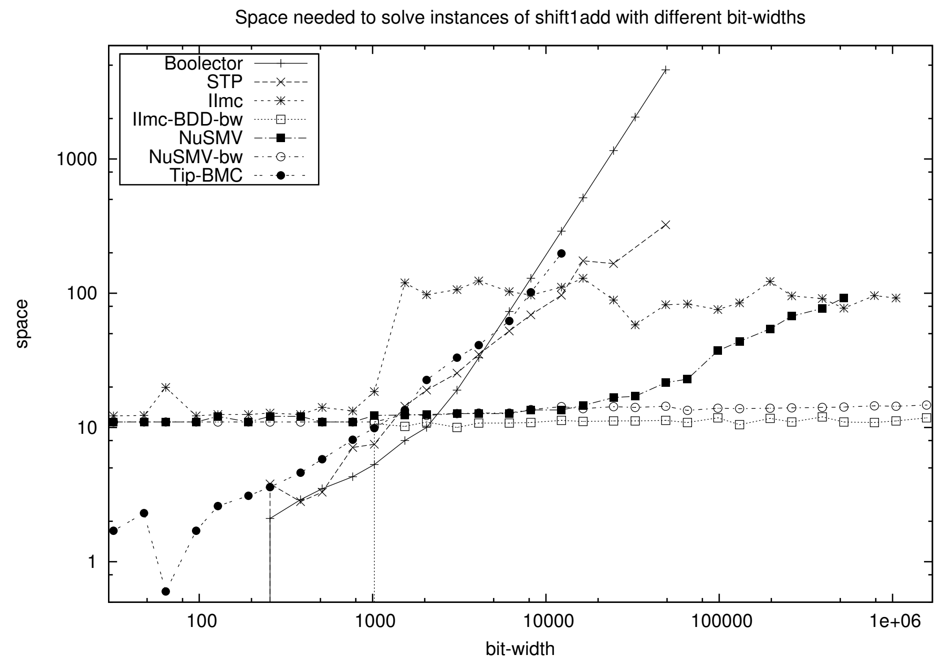
SMV Encoding of Addition


```
init(counter_bit0) := FALSE;
next(counter_bit0) := counter_bit0 xor (TRUE);
init(counter_bit1) := FALSE;
next(counter_bit1) := counter_bit1 xor (counter_bit0);
...
init(counter_bit19) := FALSE;
next(counter_bit19) := counter_bit19 xor
  (counter_bit0 & ... & counter_bit18);

init(counter_gte_1000000) := FALSE;
next(counter_gte_1000000) := counter_gte_1000000 |
  (counter_bit0 & counter_bit1 & ... & counter_bit19);

init(atom_add) := TRUE;
next(atom_add) := case
  counter_gte_1000000 : atom_add;
  TRUE : atom_add & (z <-> (x xor y xor atom_cin));
esac;

init(atom_cin) := FALSE;
next(atom_cin) := case
  counter_gte_1000000 : atom_cin;
  TRUE : (x & y) | (x & atom_cin) | (y & atom_cin);
esac;

AG(!counter_gte_1000000 | !atom_add)
```


JOHANNES KEPLER UNIVERSITY LINZ

- BV2SMV: Polynomial translation from QF_BV$_{\ll_1}$ to SMV [SMT'13]
- SMV formulas can be solved with model checkers
  - BDD based model checkers are most efficient





- Application benchmarks by Intel

Outline

- Preliminaries

- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT

- Conclusion

JOHANNES KEPLER UNIVERSITY LINZ

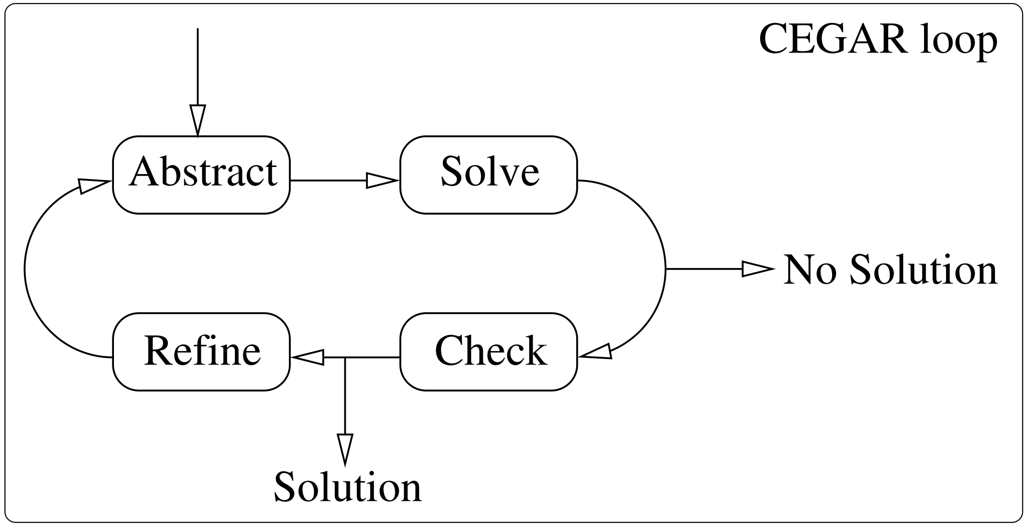
DQBF: Motivation

- Interesting in the context of QF_BV
  - DQBF is NEXPTIME-complete $\rightarrow$ possible target logic for QF_BV
- Succinct encodings of problems
  - Partial equivalence checking
  - Partial information games
- However: Not a lot of previous work
  - Mainly theoretic
  - No existing solver

DQBF: Solvers (1)

- DQDPLL [POS'12]
  - DPLL and QDPLL successful for SAT and QBF
  - Search-based approach
    - Requires **dependency constraints** to be respected
  - Many **techniques can be lifted** (bottom-up)
    - Unit Propagation, Pure Literal Reduction, Clause Learning
    - Universal Reduction, Cube Learning
  - Prototype: Not very efficient
- The first existing DQBF solver

- iDQ [POS'14]
  - iProver successful for EPR
  - Techniques can be reused and refined (top-down)
    - SAT overapproximations
    - **CEGAR loop**
  - More efficient than iProver
  - Can compete with QBF solvers



- First publicly available (complete) DQBF solver

JOHANNES KEPLER UNIVERSITY LINZ

Outline

- Preliminaries
- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT
- Conclusion

JOHANNES KEPLER UNIVERSITY LINZ

- Search on the space of full assignments $\alpha \in \{0,1\}^n$
  - Starting from an initial assignment
  - Local "improvement" in regard to a heuristic "score"
  - Typical score for SAT: Number of unsatisfied clauses

- Example:
  - $F = (x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2),$ with $\alpha = (0,0,0), F(\alpha) = 0 \wedge 1 \wedge 1$

    $\rightarrow \alpha(x_0) := \neg\alpha(x_0),$ with $\alpha = (1,0,0), F(\alpha) = 1 \wedge 0 \wedge 1$

    $\rightarrow \alpha(x_2) := \neg\alpha(x_2),$ with $\alpha = (1,0,1), F(\alpha) = 1 \wedge 1 \wedge 1$

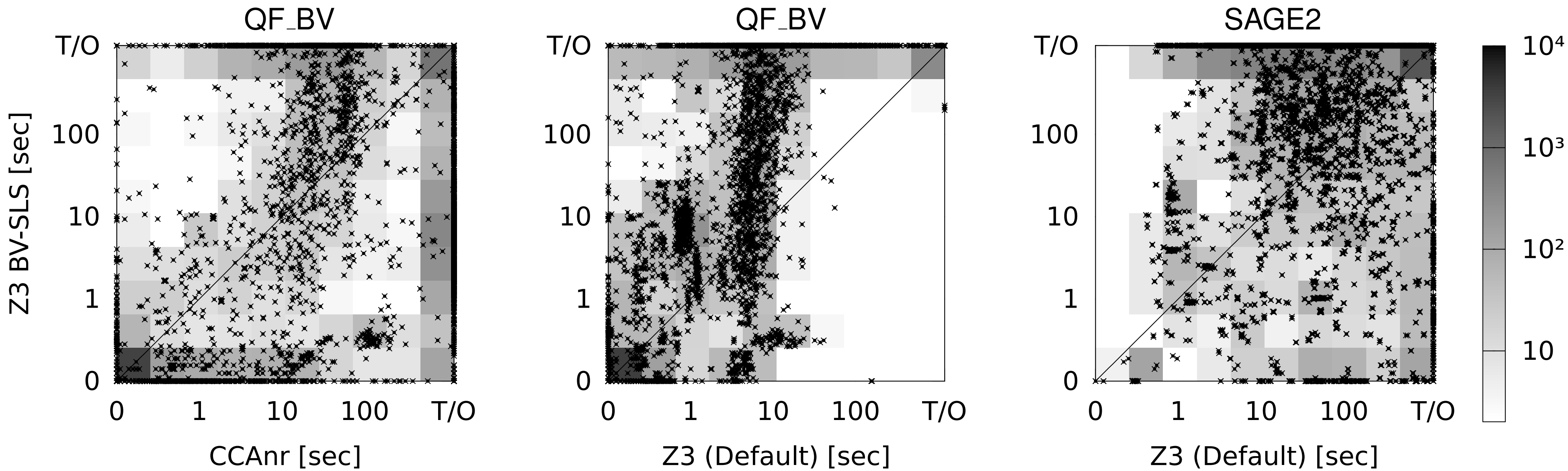
- Stochastic: Probabilistic component in choosing the next move

JOHANNES KEPLER UNIVERSITY LINZ

- Stochastic local search for SAT
  - Lots of previous work, but bad on application benchmarks
- BV-SLS: Stochastic local search for bit-vectors [AAAI'15]
  - **No bit-blasting**
  - Works on the theory representation of the formula
- Idea: Combine **techniques from SAT** with QF_BV **theory information**
  - Many techniques from SAT can successfully be lifted
  - Theory information allows to deal with structure efficiently

BV-SLS: Results

	solved instances	
	QF_BV	SAGE2
CCAnr	**5409**	64
CCASat	4461	8
probSAT	3816	10
Sparrow	3806	12
VW2	2954	4
PAWS	3331	**143**
YalSAT	3756	142
Z3 (Default)	7173	5821
Z3 BV-SLS	**6172**	**3719**



JOHANNES KEPLER UNIVERSITY LINZ

Outline

- Preliminaries
- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT
- Conclusion

Conclusion

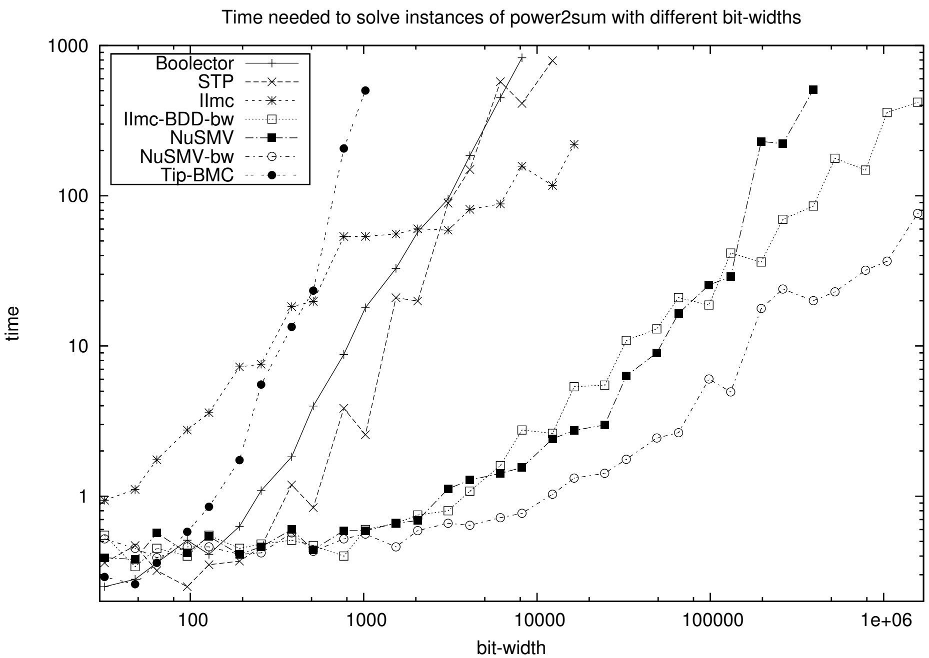
- Presented contributions:
  - Complexity of quantifier-free bit-vector logics **[SMT'12, CSR'13]**
  - Reencoding of QF_BV$_{\ll_1}$ to SMV **[SMT'13]**
  - 2 decision procedures for DQBF **[POS'12, POS'14]**
  - Lifting stochastic local search to the theory level **[AAAI'15]**
- Further results:
  - Reencoding of QF_BV to EPR **[CADE'13]**
  - More on the complexity of bit-vector logics [MFCS'14, **TOCS'15, Thesis'16**]
  - Improving state-of-the-art in SAT solving [SAT'14a, SAT'14b, POS'15, SAT'15]

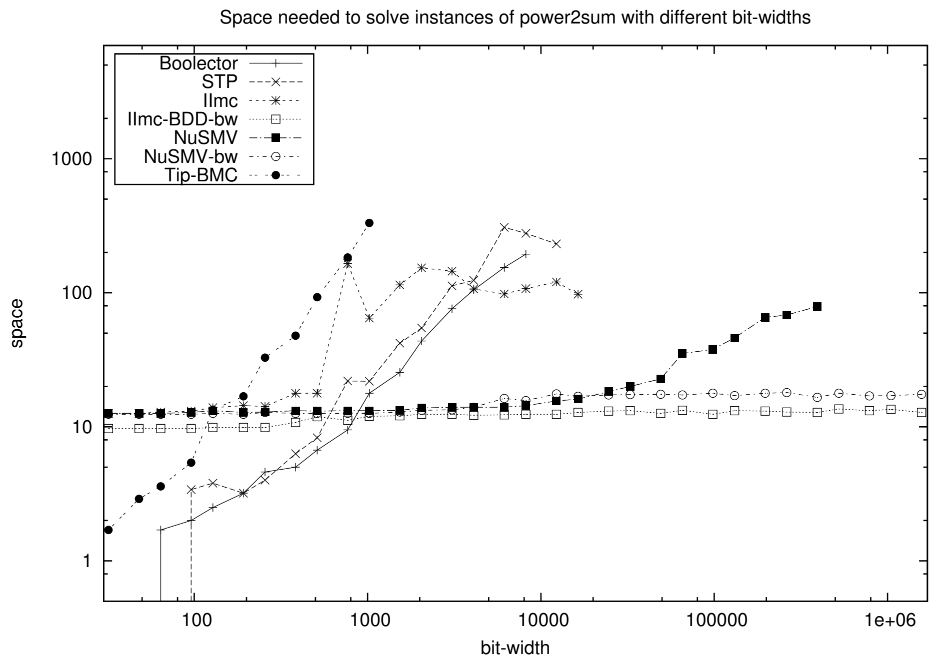
JOHANNES KEPLER UNIVERSITY LINZ

- Andreas Fröhlich, Gergely Kovásznai, Armin Biere. A DPLL Algorithm for Solving DQBF. [POS'12]
- Gergely Kovásznai, Andreas Fröhlich, Armin Biere. On the Complexity of Fixed-Size Bit-Vector Logics with Binary Encoded Bit-Width. [SMT'12]
- Gergely Kovásznai, Andreas Fröhlich, Armin Biere. BV2EPR: A Tool for Polynomially Translating Quantifier-free Bit-Vector Formulas into EPR. [CADE'13]
- Andreas Fröhlich, Gergely Kovásznai, Armin Biere. More on the Complexity of Quantifier-Free Fixed-Size Bit-Vector Logics with Binary Encoding. [CSR'13]
- Andreas Fröhlich, Gergely Kovásznai, Armin Biere. Efficiently Solving Bit-Vector Problems Using Model Checkers. [SMT'13]
- Gergely Kovásznai, Helmut Veith, Andreas Fröhlich, Armin Biere. On the Complexity of Symbolic Verification and Decision Problems in Bit-Vector Logic. [MFCS'14]
- Tomáš Balyo, Andreas Fröhlich, Marijn Heule, Armin Biere. Everything You Always Wanted to Know about Blocked Sets (But Were Afraid to Ask). [SAT'14a]

- Adrian Balint, Armin Biere, Andreas Fröhlich, Uwe Schöning. Improving implementation of SLS solvers for SAT and new heuristics for k-SAT with long clauses. [SAT'14b]
- Andreas Fröhlich, Gergely Kovásznai, Armin Biere, Helmut Veith. iDQ: Instantiation-Based DQBF Solving. [POS'14]
- Andreas Fröhlich, Armin Biere, Christoph M. Wintersteiger, Youssef Hamadi. Stochastic Local Search for Satisfiability Modulo Theories. [AAAI'15]
- Gergely Kovásznai, Andreas Fröhlich, Armin Biere. Complexity of Fixed-Size Bit-Vector Logics. [TOCS'15]
- Armin Biere, Andreas Fröhlich. Evaluating CDCL Variable Scoring Schemes. [SAT'15]
- Armin Biere, Andreas Fröhlich. Evaluating CDCL Restart Schemes. [POS'15]
- Andreas Fröhlich. Theoretical and Practical Aspects of Bit-Vector Reasoning. [Thesis'16]

- BV2SMV: Polynomial translation from QF_BV$_{\ll_1}$ to SMV [SMT'13]
- SMV formulas can be solved with model checkers
  - BDD based model checkers are most efficient





- Application benchmarks by Intel

JOHANNES KEPLER UNIVERSITY LINZ

$$x + 3 = \sim x \,,$$

where x is a bit-vector of $n = 6$. If we initialize the search:

$$x = [0,0,0,0,0,0]$$

$$\rightarrow [0,0,0,0,1,1] = [1,1,1,1,1,1]$$

Best improvement by negating x:

$$x = [1,1,1,1,1,1]$$

$$\rightarrow [0,0,0,0,1,0] = [0,0,0,0,0,0]$$

Flipping the least significant bit is the only move that will further increase the score:

$$x = [1,1,1,1,1,0]$$

$$\rightarrow [0,0,0,0,0,1] = [0,0,0,0,0,1]$$

$$\psi \;=\; \forall u_1, u_2 \exists e_1(u_1, u_2), e_2(u_2)\;.\;(u_1 \vee e_1) \wedge (\overline{u}_2 \vee \overline{e}_1 \vee e_2)$$

Initial set of clause instances:

$$(e_1)_{\overline{u}_1} \wedge (\overline{e}_1 \vee e_2)_{u_2}$$

Propositional abstraction:

$$(x_1) \wedge (\overline{x}_2 \vee x_3)$$

$\rightarrow \alpha = \{x_1 \rightarrow 1, x_2 \rightarrow 0, x_3 \rightarrow 0\}$

Refinement:

$$(e_1)_{\overline{u}_1} \wedge (e_1)_{\overline{u}_1 u_2} \wedge (\overline{e}_1 \vee e_2)_{u_2} \wedge (\overline{e}_1 \vee e_2)_{\overline{u}_1 u_2}$$

Propositional abstraction:

$$(x_1) \wedge (x_2) \wedge (\overline{x}_3 \vee x_4) \wedge (\overline{x}_2 \vee x_4)$$

$\rightarrow \alpha = \{x_1 \rightarrow 1, x_2 \rightarrow 1, x_3 \rightarrow 0, x_4 \rightarrow 1\}$

Bitwise: $z^{[2^n]} = x^{[2^n]} \mid y^{[2^n]}$

$$p_z(i_{n-1},\dots,i_0) \;\leftrightarrow\; p_x(i_{n-1},\dots,i_0) \vee p_y(i_{n-1},\dots,i_0)$$

Shift by one: $z^{[2^n]} = x^{[2^n]} \ll 1^{[2^n]}$

$$succ(i_{n-1},\dots,i_3,i_2,i_1,0,i_{n-1},\dots,i_3,i_2,i_1,1)$$

$$succ(i_{n-1},\dots,i_3,i_2,0,1,i_{n-1},\dots,i_3,i_2,1,0)$$

$$succ(i_{n-1},\dots,i_3,0,1,1,i_{n-1},\dots,i_3,1,0,0)$$

$$\vdots$$

$$succ(0,1,\dots,1,\;1,0,\dots,0)$$

$$\neg p_z(0,\dots,0) \;\wedge\; (succ(i_{n-1},\dots,i_0,\;j_{n-1},\dots,j_0) \;\rightarrow\; (p_z(j_{n-1},\dots,j_0) \;\leftrightarrow\; p_x(i_{n-1},\dots,i_0)))$$

- State-of-the-art solvers for QF_BV rely on **bit-blasting** and SAT solvers
  - Bit-blasting can be exponential
  - Is it possible to solve QF_BV without bit-blasting?
  - Can we profit from knowing the complexity of certain bit-vector classes?

- Some **alternative approaches** (and optimizations) exist
  - Translation to EPR [CADE'13]
  - Translation to SMV [SMT'13]
  - Bit-width reduction (by Johannsen)
  - SLS for SMT [AAAI'15]

JOHANNES KEPLER UNIVERSITY LINZ

- BV2EPR: Polynomial translation from QF_BV to EPR [CADE'13]
- EPR formulas can be solved with iProver (by Korovin)
  - CEGAR approach
- Performance worse than bit-blasting for most instances
  - Beneficial on some instances (0.1s instead of T/O)
  - Less memory used (can be several orders of magnitude)

JOHANNES KEPLER
UNIVERSITY LINZ

- For QF_BV_{bw}, bit-width reduction can be applied
  - There is a solution iff there is a solution with smaller bit-width, e.g.

$$\left(X^{[32]} \neq Y^{[32]} | Z^{[32]}\right) \wedge \left(Y^{[32]} \neq Z^{[32]} \& X^{[32]}\right)$$

$$\rightarrow \qquad \left(X^{[2]} \neq Y^{[2]} | Z^{[2]}\right) \wedge \left(Y^{[2]} \neq Z^{[2]} \& X^{[2]}\right)$$

  - Can be extended to allow certain cases of other operators

- Existing work for RTL Property Checking (by Johannsen)
  - Reduces size of design model to up to 30%
  - Reduces runtimes to up to 5%

- **Upgrading Theorem**: If a problem is complete for a complexity class C, it is complete for a ν-exponentially harder complexity class than C when succinctly encoded by bit-vectors with ν-logarithmic scalars. [MFCS'14]
  - Implication: Word-Level Model Checking and Reachability for bit-vectors with binary encoded bit-widths are EXPSPACE-complete.

- **Upgrading SAT**: Satisfiability for quantifier-free bit-vector formulas with ν-logarithmic encoded scalars is ν-NEXPTIME-complete. [Thesis'16]
  - Proof: Reduction from Turing machines or domino tiling problems.

Complexity of Bit-Vector Logics

<table>
<tr><td colspan="2" rowspan="4"></td><td colspan="4">quantifiers</td></tr>
<tr><td colspan="2">no</td><td colspan="2">yes</td></tr>
<tr><td colspan="2">uninterpreted functions</td><td colspan="2">uninterpreted functions</td></tr>
<tr><td>no</td><td>yes</td><td>no</td><td>yes</td></tr>
<tr><td rowspan="2">encoding</td><td>unary</td><td>NP</td><td>NP</td><td>PSPACE</td><td>NEXPTIME</td></tr>
<tr><td>binary</td><td>NEXPTIME</td><td>NEXPTIME</td><td>?</td><td>2-NEXPTIME</td></tr>
</table>

JOHANNES KEPLER UNIVERSITY LINZ

Encoding Turing Machines (1)

- The head initially is at position 0:

$$H^{[N]} \wedge lo^{[N]} = 1^{[N]} \ll mid^{[N]}$$

- M initially is in state s:

$$Q_s^{[N]} \wedge lo^{[N]} = 1^{[N]} \ll mid^{[N]}$$

- In each computation step, there is at most one symbol per tape cell, i.e., $\forall \sigma, \sigma' \in \Sigma$, with $\sigma \neq \sigma'$, we add:

$$\neg T_{\sigma}^{[N]} \vee \neg T_{\sigma'}^{[N]} = \neg 0^{[N]}$$

- In each computation step, there is at least one symbol per tape cell:

$$\bigvee_{\sigma \in \Sigma} T_{\sigma}^{[N]} = \neg 0^{[N]}$$

- In each computation step, there is at most one state at a time, i.e., $\forall q, q' \in Q$, with $q \neq q'$, we add:

$$\neg Q_q^{[N]} \vee \neg Q_{q'}^{[N]} = \neg 0^{[N]}$$

- The bits of the state variables can only be set at the head positions:

$$\bigvee_{q \in Q} Q_q^{[N]} \wedge \neg H^{[N]} = 0^{[N]}$$

- The tape does not change at positions different from those of the head, i.e., $\forall \sigma \in \Sigma$, we add:

$$(T_\sigma^{[N]} \ll size^{[N]} \leftrightarrow T_\sigma^{[N]}) \vee (H^{[N]} \ll size^{[N]}) \vee lo^{[N]} = \neg 0^{[N]}$$

- The transition relation, i.e., $\forall q \in Q, \sigma \in \Sigma$, we add:

$$(H^{[N]} \wedge Q_q^{[N]} \wedge T_\sigma^{[N]}) \ll size^{[N]} \rightarrow$$

$$\bigvee_{(q,\sigma,q',\sigma',d)\in\delta} (H^{[N]} \circ_d 1^{[N]} \wedge Q_{q'}^{[N]} \wedge T_{\sigma'}^{[N]}) = \neg 0^{[N]}$$

- M must reach a final state at one point:

$$\bigvee_{q\in F} Q_q^{[N]} \wedge H^{[N]} \neq 0^{[N]}$$

- Helper variables: $size^{[N]} = 2 \cdot \exp_\nu(n) + 1, \quad mid^{[N]} = \exp_\nu(n),$

$$hi^{[N]} = \neg 0^{[N]} \ll size^{[N]}, \quad lo^{[N]} = \neg(\neg 0^{[N]} \ll size^{[N]})$$

JOHANNES KEPLER UNIVERSITY LINZ

Encoding Turing Machines (4)

- If the head is in a certain position in a computation step, it cannot be at any position other than left or right of the current one in the next step:

$$lo^{[N]} \vee \neg H^{[N]} \vee H^{[N]} \ll (size+1)^{[N]} \vee H^{[N]} \ll (size-1)^{[N]} = \neg 0^{[N]}$$

- If the head is in a certain position in a computation step, it has to be at position left or right (non-exclusive) of the current one in the next step:

$$\neg(H^{[N]} \ll size^{[N]}) \vee H^{[N]} \ll 1^{[N]} \vee H^{[N]} \gg_{\mathbf{u}} 1^{[N]} = \neg 0^{[N]}$$

- In any computation step, the head will never be at two distinct positions exactly two indices apart from each other:

$$H^{[N]} \ll 1^{[N]} \wedge H^{[N]} \gg_{\mathbf{u}} 1^{[N]} = 0^{[N]}$$