How Hard is Bit-Precise Reasoning?

Gergely Kovásznai

IoT Research Institute,
Eszterhazy Karoly University,
Eger, Hungary
kovasznai.gergely@uni-eszterhazy.hu

ICAI 2017
January 31, 2017
Eger, Hungary
Why So Many Bugs Nowadays?

So many bugs in hardware/software! But why?

- Complexity increases.

- Length of development cycles decreases.

Firefox’s current version is 51.
Historical incidents due to bit-level hardware/software bugs
The need for hardware and software verification
Bit-vector logics
Precise computation complexity of bit-vector logics
Incidents – Mars Orbiter Disappears (1998)

NASA’s Mars Climate Orbiter disappeared in space after being launched. The engineering team used English units, while the software employed the metric system.

$193$ million were lost.
A launcher Ariane 5 exploded after 37 seconds of flight due to an overflow in its software.\footnote{A 16-bit number was reused inside a 32-bit number, causing an overflow and making the launcher uncontrollable.}

$370$ million were lost.
The US air safety authority issued a warning on a software bug of Boeing 787. The plane’s electric generators switch off if kept continuously powered on for 248 days, due to an overflow.

No crashes and no casualties.
In the National Cancer Institute, Panama City, a software miscalculated the proper dosage of radiation for patients. The system calculated twice the necessary dose when entering the input data in a very specific order.

8 patients died, 20 received overdoses.
Incorrect decimal results when dividing floating point numbers within a specific range. The bug was located inside the floating-point division circuitry.

Intel first offered to replace chips only for consumers who could prove that they needed high accuracy. Later the company agreed to replace the chips for anyone who complained. The bug costed $475 million for Intel.
Formal verification of systems.

A priori. Verifying the system before releasing.

Guarantee. Not the same as testing. Verification guarantees that the system meets the (safety) specification.

- E.g., no overflow, no division by zero, no data outside of suitable domain, no infinite loop, etc.

Formal. Given a mathematical model of the system and specifications, formal verification provides a mathematical proof.
Formal Verification of Hardware and Software

- **Hardware** = logical circuits

- **Software**

```java
@override void CollectAllRules(EngineNode engineNode)
{
    var rules = GetRuleCollection(engineNode);
    rules.Clear();

    foreach (var companion in AllCompanions)
    {
        var info = companion.GetType().GetTypeInfo();
        foreach (var methodinfo in info.DeclaredMethods)
        {
            foreach (var attribute in methodinfo.GetCustomAttributes)
            {
                var lns = attribute as LnAttribute;
                if (lns != null && lns.EngineNode == engineNode)
                {
                    var rule = AddRule(engineNode, lns);
                    rule.Companion = companion;
                    rule.Salience = lns.Salience;
                    rule.Lhs = methodInfo.CreateDelegate<
```
Bit-Precise Reasoning – SAT solving

Boolean formulas, DIMACS format, SAT solvers.

SAT problem: NP-complete.

Broad applications, e.g.:

- Intel Core i7 CPUs designed with the help of SAT solvers [le Berre et al., 2011]
- Cryptographics cyphers were analyzed by the use of SAT solvers [Soos et al., 2009]

Millions of variables and clauses. GBs in DIMACS.
SAT Competitions

Number of solved instances within a given amount of CPU time

[http://satcompetition.org/2014]
SMT = SAT Modulo Theory: checking SAT w.r.t. background theories such as integer arithmetic, real arithmetic, bit-vectors, arrays, etc.

SMT-LIB format: variables of different types, lots of operators (e.g., arithmetic, relational, shift, etc.), functions, quantifiers, etc.

```
(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))
```

Broad applications, e.g.:

- Windows device drivers verified by Microsoft’s SMT solver, Z3 [de Moura, Bjørner, 2010]
- Communication protocols of wireless sensor networks verified [Duan et al., 2012]
## Results: QF_BV (Main Track)

<table>
<thead>
<tr>
<th>Solver</th>
<th>Error Score</th>
<th>Solved Score (Parallel)</th>
<th>Unsolved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolector (pre)</td>
<td>0.000</td>
<td>24473.995</td>
<td>149</td>
</tr>
<tr>
<td>Boolector</td>
<td>0.000</td>
<td>24468.395</td>
<td>150</td>
</tr>
<tr>
<td>Minkeyrink</td>
<td>0.000</td>
<td>24434.194</td>
<td>193</td>
</tr>
<tr>
<td>smt-cms-mt</td>
<td>0.000</td>
<td>24244.599</td>
<td>216</td>
</tr>
<tr>
<td>smt-cms-st</td>
<td>0.000</td>
<td>24165.007</td>
<td>214</td>
</tr>
<tr>
<td>CVC4</td>
<td>0.000</td>
<td>23820.707</td>
<td>231</td>
</tr>
<tr>
<td>Z3</td>
<td>0.000</td>
<td>23732.215</td>
<td>304</td>
</tr>
<tr>
<td>smt-cms-exp</td>
<td>0.000</td>
<td>23640.669</td>
<td>270</td>
</tr>
<tr>
<td>ABC_glucose</td>
<td>0.000</td>
<td>23078.931</td>
<td>477</td>
</tr>
<tr>
<td>Yices2</td>
<td>0.000</td>
<td>22687.777</td>
<td>638</td>
</tr>
<tr>
<td>MathSat5</td>
<td>0.000</td>
<td>22496.779</td>
<td>544</td>
</tr>
<tr>
<td>MapleSTP-mt</td>
<td>0.000</td>
<td>22487.264</td>
<td>395</td>
</tr>
<tr>
<td>MapleSTP</td>
<td>0.000</td>
<td>21764.885</td>
<td>450</td>
</tr>
<tr>
<td>smt-minisat-st</td>
<td>0.000</td>
<td>20582.614</td>
<td>1058</td>
</tr>
<tr>
<td>ABC_default</td>
<td>0.000</td>
<td>18528.788</td>
<td>1354</td>
</tr>
<tr>
<td>Q3B</td>
<td>719.723</td>
<td>10397.757</td>
<td>4430</td>
</tr>
</tbody>
</table>

[http://smtcomp.sourceforge.net/2016]
Bit-vector Logics

- Bit-vector variables of given bit-width:
  
  \[\text{declare-fun x () (_ BitVec 32)}\]

- Bit-vector operators, e.g.:
  
  - Bitwise: \textit{bvnot, bvand, bvor, bvxor, etc.}
  - Arithmetic: \textit{bvadd, bvsub, bvmul, bvdiv, etc.}
  - Relational: \textit{bvult, bvslt, bvugt, bvsgt, etc.}
  - Shift: \textit{bvshl, bvlshr, bvashr, etc.}

- Uninterpreted functions:
  
  \[\text{declare-fun f ((_ BitVec 8) (_ BitVec 8)) (_ BitVec 16)}\]

- Quantifiers: \textit{forall, exists}
Bit-vector Logics

Commonly used in hardware verification:
- **QF_BV**: Quantifier-Free Bit-Vector logic
- **QF_UFBV**: Quantifier-Free Bit-Vector logic with Uninterpreted Functions

Commonly used in software verification:
- **BV**: Bit-Vector logic (with quantifiers)
- **UFBV**: Bit-Vector logic with Uninterpreted Functions (and quantifiers)
How Hard is a Decision Problem? – Computational Complexity

- **P**
  - Problems with polynomially time-bounded algorithms.

- **NP**
  - Same as P, but with non-deterministic choice.
  - Can be solved by a SAT solver.

- **PSpace**
  - Same as P, but space-bounded.

- **NExpTime**
  - Same as NP, but with exponential time.

- **ExpSpace**
  - Same as PSpace, but with exponential space.

- **2-NExpTime**
  - Same as NExpTime, but with double exponential time.

Gergely Kovásznai

How Hard is Bit-Precise Reasoning?
The computational complexity of a decision problem is precisely characterized if it is proved to be complete for a complexity class.

E.g., SAT is NP-complete. Why?

- SAT ∈ NP:
  - SAT’s complexity ≤ NP.
- SAT is NP-hard:
  - Any NP problem can be reduced to SAT.
  - SAT’s complexity ≥ NP.
What is the Computational Complexity of Bit-Vector Logics?

What is the complexity of the satisfiability problem in
- QF_BV?
- QF_UFBV?
- BV?
- UFBV?

Precise complexity ( = completeness) suggests *solving approaches* for the logic.
Historical Background in Literature

QF_BV is massively used in hardware verification.
- For decades, QF_BV was implicitly assumed to be NP-complete.
- [Barrett et al., 1998] QF_BV is NP-hard
- [Bruttomesso, Sharygina, 2009] QF_BV has an NP-complete sublogic
- [Bryant et al., 2007] QF_BV is NP-complete

UFBV is massively used in software verification.
- [Wintersteiger et al., 2011] UFBV is $\text{NExpTime}$-complete
We successfully proved:

**QF_BV**: $\text{NExpTime}$-complete
It has $\text{NP}$-complete and $\text{PSPACE}$-complete sublogics.

**QF_UFBV**: $\text{NExpTime}$-complete
**BV**: ?
**UFBV**: $2\text{-NExpTime}$-complete
Complexity of QF\_BV

[Kovasznai et al., 2012] QF\_BV is $\text{NExpTime}$-complete.

Proof of $\text{NExpTime}$-hardness requires exponentially concise encoding of the underlying Boolean (SAT) problem.

- DQBF is a $\text{NExpTime}$-complete logic.
- We gave a polynomial reduction of DQBF to QF\_BV.
- For this, we used binary magic numbers [Knuth, 2011].

$$binmagic(2^m, 2^n) := \frac{2^{2^n} - 1}{2^{2^m} + 1}$$

$$b^{[2^n]} \ll (1 \ll m) = \sim b^{[2^n]}$$
Can we reduce the complexity of $\text{QF}_\text{BV}$ by restricting the set of permitted bit-vector operators? 

$\text{QF}_\text{BV} \ll_c$: $\text{NExpTime}$-complete. 
Only bitwise operations, equality, and left shift by constant, i.e., $t[n] \ll c$, are allowed.
Can we reduce the complexity of QF_BV by restricting the set of permitted bit-vector operators?

QF_BV \ll_1 \text{ PSpace}-complete.
Only bitwise operations, equality, and \textit{left shift by 1}, i.e., \( t^n \ll 1 \), are allowed.

Such bit-vector problems can be solved by QBF solvers, model checkers [Fröhlich et al, 2013].
[Kovasznai et al., 2016] Can we reduce the complexity of \( QF_{BV} \) by restricting the set of permitted bit-vector operators?

\( QF_{BV_{bw}} \): NP-complete.
Only bitwise operations and equality are allowed.

Such bit-vector problems can be solved by SAT solvers.
Sublogics of $QF_{BV}$ with Lower Complexity

2-NExpTime
ExpSpace
NExpTime
PSPACE
NP
QF_{BV}^c
QF_{BV}^w
QF_{BV}^l

Gergely Kovásznai  How Hard is Bit-Precise Reasoning?
What if we add uninterpreted functions to QF_BV? Does the complexity increases?
What if we add quantifiers to QF_UFBV?
Does the complexity increase?
Complexity of UFBV

[Kovasznai et al., 2012] UFBV is $2\text{-NExpTime}$-complete.

Proof of $2\text{-NExpTime}$-hardness:
- The $2^{(2^n)}$-square tiling problem is $2\text{-NExpTime}$-complete.

• We gave a polynomial reduction to UFBV.
Complexity of UFBV

[Kovasznai et al., 2012] UFBV is 2-NExpTime-complete.

Proof of 2-NExpTime-hardness:
- The $2^{(2^n)}$-square tiling problem is 2-NExpTime-complete.
- We gave a polynomial reduction to UFBV.

\[
\forall i^{[2^n]}, j^{[2^n]}. \\
\lambda(0, 0) = 0 \land \lambda(\sim 0^{[2^n]}, \sim 0^{[2^n]}) = k - 1 \\
\land \bigwedge_{(t_1, t_2) \in H} h(t_1, t_2) \land \bigwedge_{(t_1, t_2) \in V} v(t_1, t_2) \\
\land \left( j \neq \sim 0^{[2^n]} \implies h(\lambda(i, j), \lambda(i, j + 1)) \right) \\
\land \left( i \neq \sim 0^{[2^n]} \implies v(\lambda(i, j), \lambda(i + 1, j)) \right)
\]
What if we do not use uninterpreted functions in quantified bit-vector formulas? Is the complexity the same?

BV is
- $\text{NExpTime}$-hard
- in $\text{ExpSpace}$

BV is probably not complete for any of those two classes.
Complexity of BV

[Jonáš, Strejček, 2016] BV is $\text{AExp}(\text{poly})$-complete.

$\text{AExp}(\text{poly})$: Problems decidable by an alternating Turing machine using exponential space and polynomial number of alternations.
Conclusion

- Bit-precise formal verification is important to save lives and money in space industry, aviation, power plants, self-driving cars, etc.
- Precise computation complexity of bit-vector logics helps companies in practice to find suitable solving approaches.
- The complexity "map" of common bit-vector logics is now complete.

Thank you for your attention!