An Efficient Decision Procedure for the Theory of Fixed-Sized Bit-Vectors

David Cyrluk*, Oliver Möller**, Harald Rueß**

* Computer Science Laboratory
SRI International
Menlo Park, CA 94025, USA

cyrluk@csl.sri.com

** Fakultät für Informatik
Universität Ulm
D-89069 Ulm, Germany

{moeller, ruess}@ki.informatik.uni-ulm.de

Introduction

Bit-vectors are a fundamental data-structure in hardware-verification

Operations

$\otimes$, $^{\wedge}$, *fill*, *shift*, *AND*, *XOR*, *bv2nat*, ...

e.g. solve $\quad x_{[3]} \otimes 0_{[1]} \stackrel{!}{=} y_{[4]}$ *XOR* $z_{[4]}$

Crucial for effective automation

- Efficiency
- Integratability

The Idea

Leave the largest possible chunks together

Overview

1. Core Theory of Bit-Vectors
2. An Efficient Solver
3. Extensions
4. Conclusion

Core Theory of Bit-Vectors

Terms

- Bit-vectors $x_{[n]}$ have fixed length n
- The bits are numbered from right to left, beginning with 0
- Constants are denoted with $0_{[n]}$ and $1_{[n]}$
- Composition: $x_{[8]} \otimes y_{[16]} \otimes 0_{[2]}$
- Extraction: $(x_{[8]} \otimes z_{[8]})^{\wedge}(9,4)$

Canonical form

- “flat” :

$$x_{[8]}{}^{\wedge}(1,0) \otimes z_{[8]}{}^{\wedge}(7,4)$$

- … and “maximally connected” :

$$x_{[8]}{}^{\wedge}(5,3) \otimes x_{[8]}{}^{\wedge}(2,0) \quad \to$$
$$x_{[8]}{}^{\wedge}(5,0)$$

What we want

Canonizer

- $t = u \;\Leftrightarrow\; \textbf{canon}(t) \equiv \textbf{canon}(u)$
- $\textbf{canon}(\textbf{canon}(\mathrm{t})) \equiv \textbf{canon}(t)$
- every subterm is canonical, too

Solver

- Given an equation over $x_{[n]}$, $y_{[m]}$, .. we expect an output like
  - **true** or
  - **false** or
  - $$\begin{array}{lll} x_{[n]} & = & \ldots \\ y_{[m]} & = & \ldots \\ .. & & \end{array}$$

How the Solver works

1. Canonize both sides of the equation
2. Slice the corresponding parts to get pairs of chunks having the same length
3. Process each of these pairs to get $0 - 2$ equations of the form
   <var> = <term over constants & fresh vars>
4. Collect these equations for all original variables in blocks
5. Perform a slicing within these blocks
6. Propagate the equality using an union-find structure

Paradigm

Treat only the largest chunks possible as distinct objects

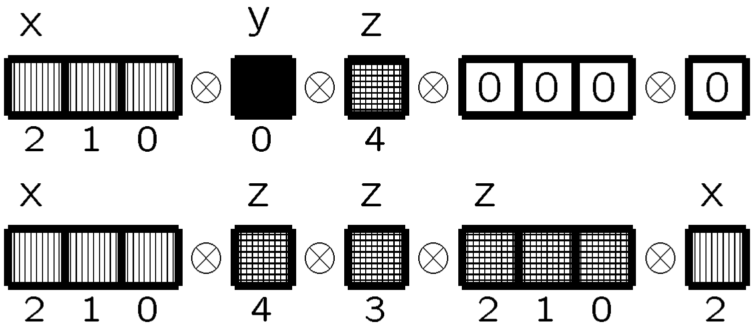
Example

$$x : bvec_{[3]} \qquad y : bvec_{[4]} \qquad z : bvec_{[5]}$$

$$x \otimes (y \otimes z)^\wedge(5,4) \otimes 0_{[4]} \stackrel{!}{=}$$
$$((y^\wedge(2,1) \otimes x)^\wedge(4,0) \otimes z \otimes x)^\wedge(10,2)$$

$$\leftrightarrow \quad x \otimes y^\wedge(0,0) \otimes z^\wedge(4,4) \otimes 0_{[4]} \stackrel{!}{=}$$
$$x \otimes z \otimes x^\wedge(2,2)$$

$\leftrightarrow$



$$\leftrightarrow \quad
\begin{aligned}
y &= \boxed{?}\boxed{?}\boxed{?} \otimes \boxed{c} \\
z &= \boxed{c} \otimes \boxed{?}\boxed{?}\boxed{?}\boxed{?} \\
z &= \boxed{b} \otimes \boxed{b} \otimes \boxed{?}\boxed{?}\boxed{?} \\
z &= \boxed{?}\boxed{?} \otimes \boxed{0}\boxed{0}\boxed{0} \\
x &= \boxed{0} \otimes \boxed{?}\boxed{?}
\end{aligned}
\quad \leftrightarrow \quad
\begin{aligned}
x &= 0_{[1]} \otimes a_{[2]} \\
y &= a'_{[3]} \otimes c_{[1]} \\
z &= c_{[1]} \otimes c_{[1]} \otimes 0_{[3]}
\end{aligned}$$

Requirements on the Solver

In the following let $E = solve(e)$, t_i denote terms and x_i represent original variables.

1. $\vdash e \Leftrightarrow E$ is in the theory
2. $E \in \{\textbf{true}, \textbf{false}\}$ or $E = \bigwedge_i x_i = t_i$
3. If e contains no variables, then $E \in \{\textbf{true}, \textbf{false}\}$
4. If $E = \bigwedge_i x_i = t_i$ then the following holds:
   (a) $x_i \in vars(e)$
   (b) for all i, j: $x_i \notin vars(t_j)$
   (c) for all $i \neq j$: $x_i \neq x_j$
   (d) for all i: $\sigma(t_i) = t_i$

[D.Cyrluk, P.Lincoln, N.Shankar; CADE '96]

Observations

- It is not necessary to reduce the data-structure *bitvector* to its bits
- Given several terms (of the same length) it is easy to compute their slicing
- It is easy to keep track of the "largest chunks" occuring
- If original variables are brought to the left hand side, equality has to be propagated only between the large chunks of fresh variables

Paradigm

Keep original variables on the left hand side and fresh ones on the right hand side

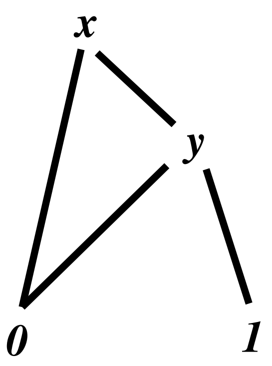
Complexity Results

- The solver for the core-theory has a moderate run-time:

$$\mathcal{O}(\ |{<}\,\textit{term-length}\,{>}|\ \cdot \textit{log}\, n + n^2\,)$$

- Trivial operations like *fill* and *shift* can be added without noticable loss of performance
- If we allow boolean operations over bit-vectors, the theory becomes *NP*-complete
- The core theory with (integer) variables in its extractions, but still fixed size, is also *NP*-complete
- If we allow variable size, the theory becomes *NP*-hard, even if there are no other operators than " $\otimes$ "

Adding Binary Decision Diagrams

$x \wedge y \hat{=}$ 

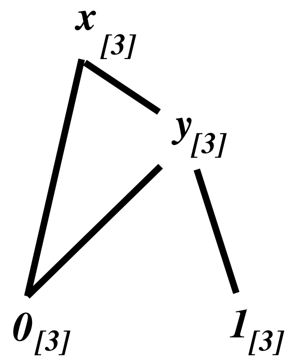
- BDDs enable us to represent boolean Formulas in a canonical form

When using BDDs over bit-vectors

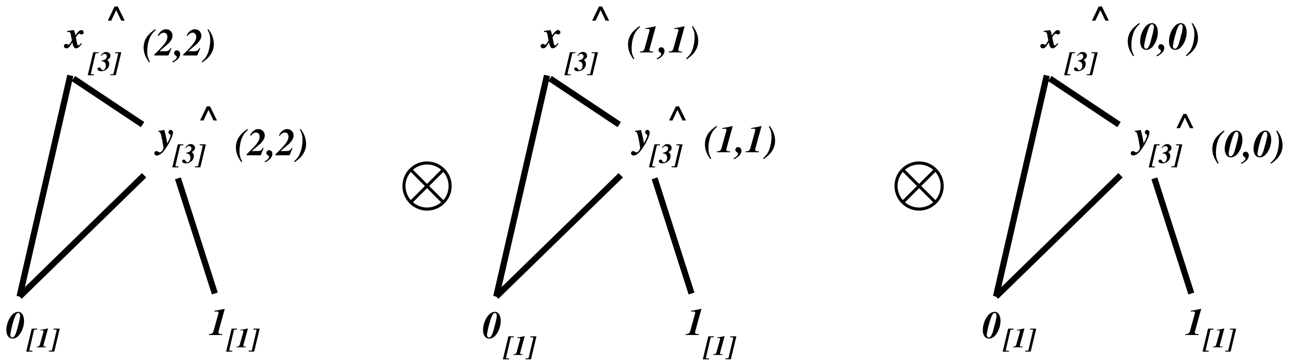
- The nodes contain bit-vector chunks like $x_{[3]}$ or $y_{[4]}$ $\wedge$ (2,0)
- Within one BDD the chunks at the nodes have the same number of bits
- Together with the paradigm of maximally connected BDDs, we get a canonical form

What is a Bit-vector BDD

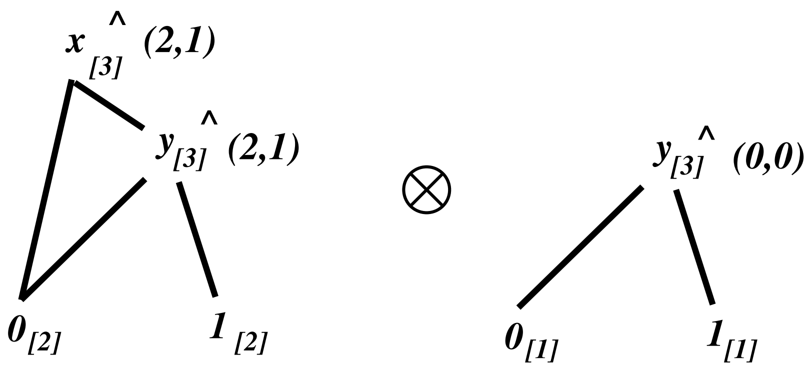
$x_{[3]} \wedge y_{[3]} \hat{=}$



is equivalent to the bit-wise form



Given the additional constraint $x_{[3]} \wedge (0{,}0) = 1_{[1]}$, this BDD specializes to



Solving BDD-equations

Instead of $BDD^1 \stackrel{!}{=} BDD^2$, we solve

$$(BDD^1 \Leftrightarrow BDD^2) \stackrel{!}{=} \mathbf{true} :$$

Recursive Procedure BDD-solve

Input S: BDD

LET $S = \mathbf{ite}(P, B_P, B_{\bar{P}})$ IN

LET δ be a fresh variable

$EQ_P := P = \mathbf{ite}(B_P, \mathbf{ite}(B_{\bar{P}}, \delta, \mathbf{true}), \mathbf{false})$

$Cstrnt := \mathbf{ite}(B_P, \mathbf{true}, B_{\bar{P}})$

/* $B_P \vee B_{\bar{P}}$ */

RETURN $\{EQ_P\} \cup \mathbf{BDD\text{-}solve}(Cstrnt)$

An Example

$\mathbf{ite}(p, \mathbf{ite}(q, \mathbf{false}, \mathbf{true}), \mathbf{false})$ is processed to

$$\begin{array}{lll} EQ_P: & P = \mathbf{ite}(\mathbf{ite}(q, \mathbf{false}, \mathbf{true}), \mathbf{ite}(\mathbf{false}, \delta, \mathbf{true}), \mathbf{false}) \\ \quad \leftrightarrow & P = \mathbf{ite}(q, \mathbf{false}, \mathbf{true}) \\ \wedge & \\ Cstrnt: & \mathbf{ite}(\mathbf{ite}(q, \mathbf{false}, \mathbf{true}), \mathbf{true}, \mathbf{false}) \\ \quad \leftrightarrow & \mathbf{ite}(q, \mathbf{false}, \mathbf{true}) \end{array}$$

This *Cstrnt* is solved recursively to

$$q = \mathbf{false}$$

And this, inserted in EQ_P produces

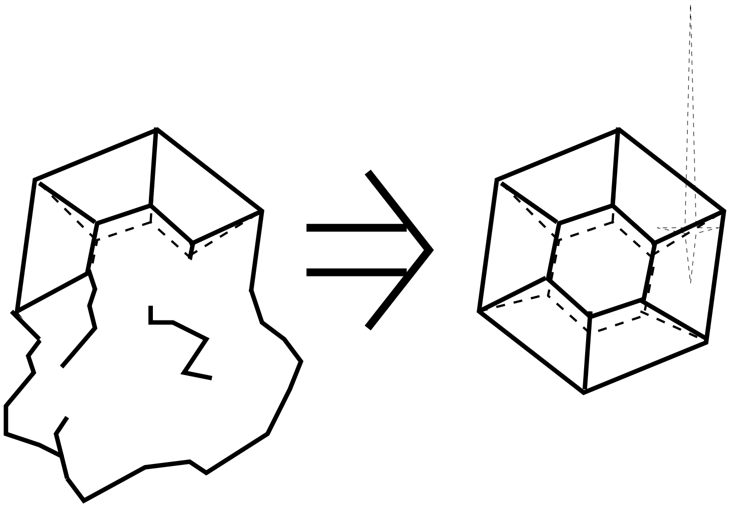
$$p = \mathbf{true}$$

Conclusion

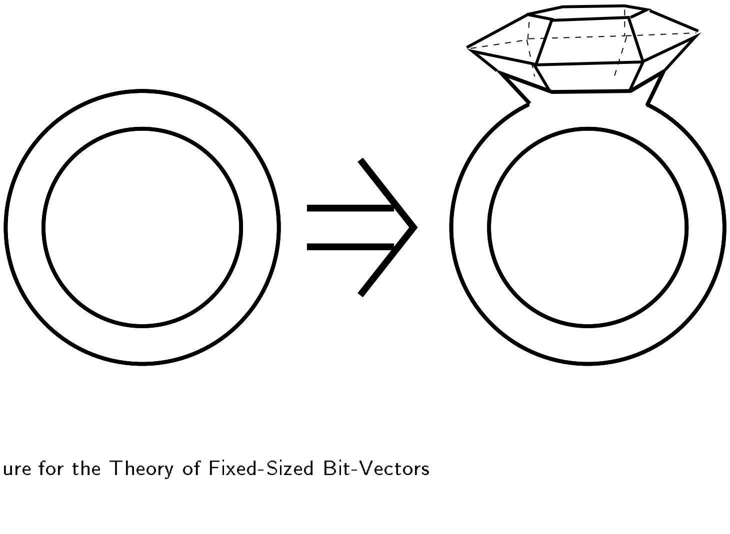
- The concept of the largest chunks yields an efficient decision procedure for the core theory
- Main advantage: the bit-length of the input does not necessarily affect run-time
- The concept can be easily enhanced to boolean operations by means of BDDs
- The procedure can readily be integrated in Shostak's framework of combining decision procedures
- Experiments with lemmas from the $AAMP5$-Verification showed that about 50% of them could be proven automatically in reasonable time

Things to do...

improve the implemention...



- Adding variable extractions to the canonizer and solver
- Allow fixed but arbitrary size
- Support arithmetic operations
- ... use core theory as a sub-procedure



Questions & Discussion