A Lazy SMT Bit-vector Solver for Binary Symbolic Execution

Clark Barrett (NYU), David Brumley (CMU),
Cesare Tinelli (U Iowa)

with Kshitij Bansal (NYU), Liana Hadarean (NYU), Dejan Jovanović (NYU)

High Confidence Software and Systems Conference
May 8, 2013

Outline

1. Introduction
   - Satisfiability Modulo Theories
   - CVC4
2. The CVC4 Bit-Vector Solver
   - Solver Design
   - Core Solver
   - Inequality Solver
   - Bit-Blasting Solver
3. Decision Heuristic
4. Results and Summary
   - QF_BV Benchmarks
   - QF_AUFBV benchmarks
   - Summary

Outline

SMT Solvers

- Powerful new automated engines for solving problems
- Speed and automation of SAT, expressive power of full first-order logic
- Can reason about arithmetic, bit-vectors, arrays, etc.

What people are saying

- Most promising contribution to fields of software and hardware verification and test in the last five years

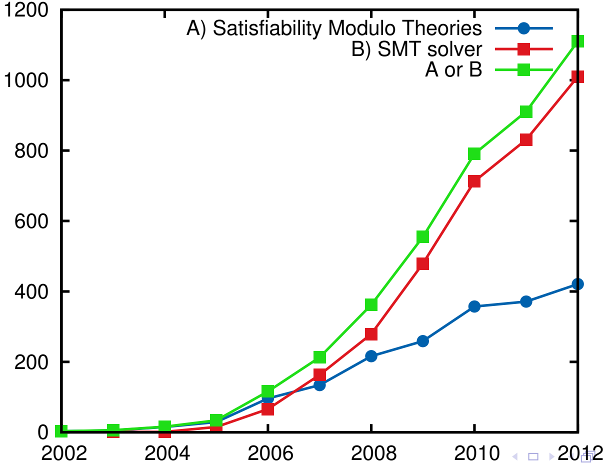
  (from the text of the HVC 2010 award)
- The biggest advance in formal methods in last 25 years

  (John Rushby, FMIS 2011)
- Most successful academic community related to logics and verification ... built in the last decade

  (editors of FMSD special issue on SMT, 2012)

Impact of SMT

Articles per year by search phrase (from Google Scholar)



Example Application: Binary Symbolic Execution

Inputs

- Binary Program, Safety Property

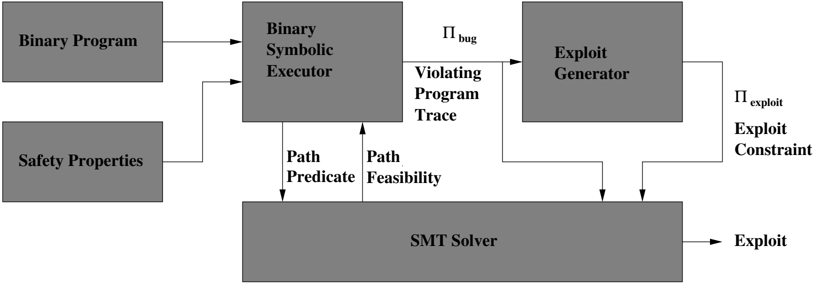
Model

- Memory, registers modeled as arrays of bit-vectors
- Instructions modeled as constraints over bit-vectors and arrays

Symbolic Execution

- Enumerate paths through binary program
- Symbolically simulate each path to generate SMT formula
- SMT solver reports bug if path is feasible but violates safety property

Automatic Exploit Generation



T. Avgerinos, S. Cha, B. Hao, and D. Brumley, “AEG: Automatic Exploit Generation,” (NDSS 2011).

C. Barrett, D. Brumley, and C. Tinelli, “Breaking the SMT Bottleneck in Symbolic Security Analysis,” Current NSF-funded Project.

CVC4

About CVC4

- Open-source (BSD) SMT solver
- Joint project of NYU and U Iowa
- Project Goals
  - Industrial-strength SMT engine
  - Flexible research platform

People

Clark Barrett (NYU) Cesare Tinelli (U Iowa)

Kshitij Bansal (NYU)	Morgan Deters (NYU)	Tim King (NYU)
François Bobot (Paris Sud)	Liana Hadarean (NYU)	Tianyi Liang (U Iowa)
Chris Conway (Google)	Dejan Jovanović (NYU)	Andrew Reynolds (U Iowa)

CVC4 Features

Performance

- Dramatic performance improvement over CVC3
- Support for parallel (portfolio) execution

Expressivity

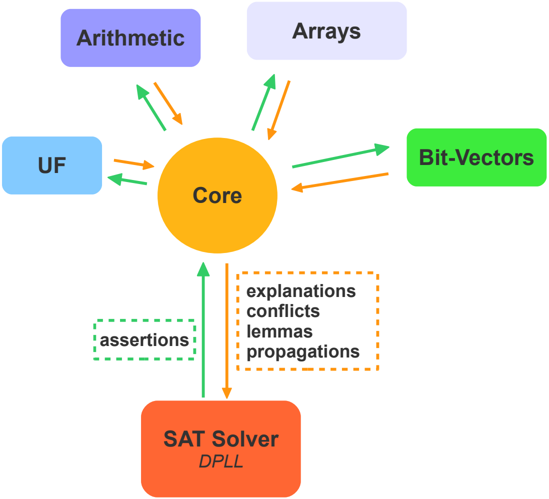
- Support for all standard SMT theories
- Work in progress: non-linear arithmetic, strings

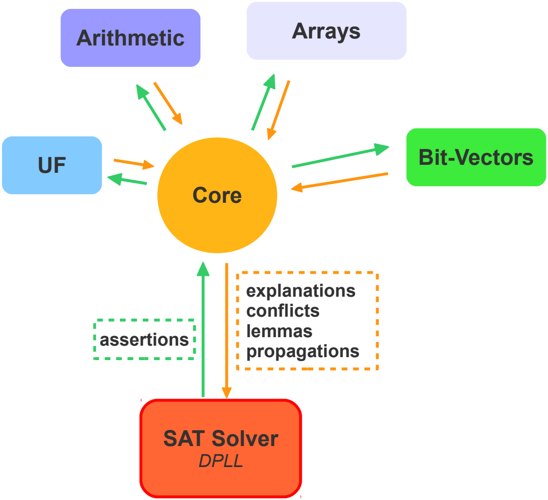
Usability

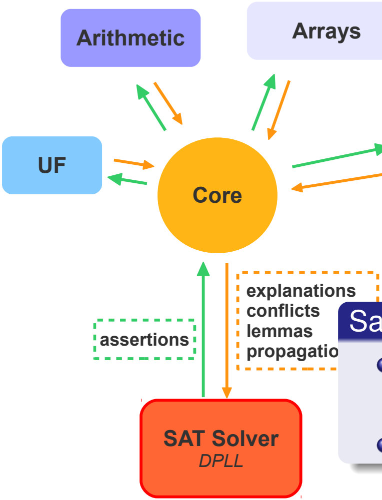
- Supports SMT-LIB v1-2, CVC, C++/Java APIs

Robustness and Reliability

- Independently checkable proofs in LFSC format

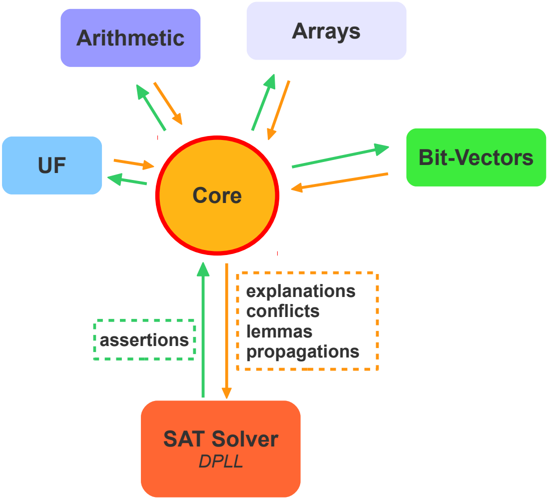

Arithmetic
Arrays
UF
Core
Bit-Vectors
assertions
explanations
conflicts
lemmas
propagations
SAT Solver
DPLL

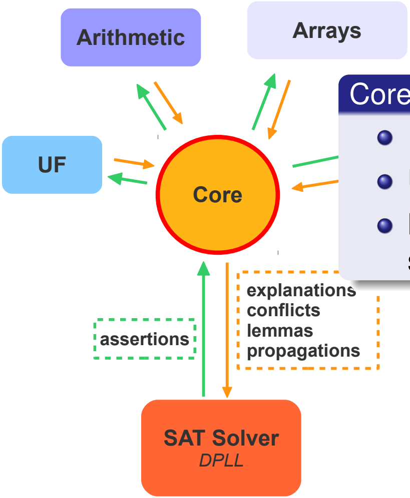

Arithmetic
Arrays
UF
Core
Bit-Vectors
assertions
explanations
conflicts
lemmas
propagations
SAT Solver
DPLL



Sat Solver

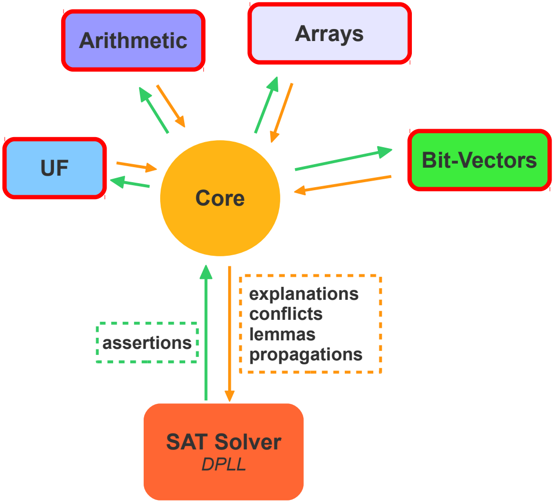
- Builds partial model of Boolean skeleton of problem
- Informs theories of decisions

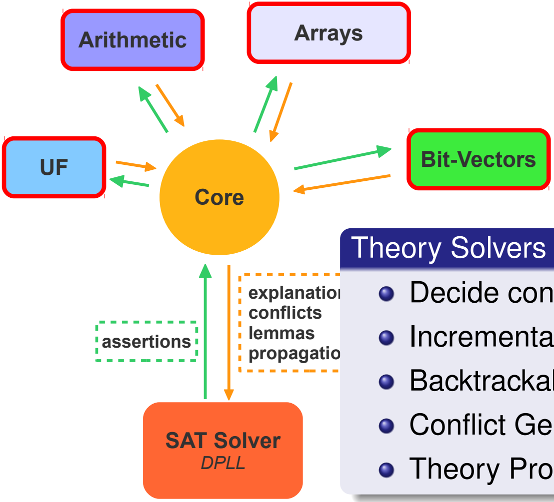

Arithmetic
Arrays
UF
Core
Bit-Vectors
assertions
explanations
conflicts
lemmas
propagations
SAT Solver
DPLL



Core

- Does theory combination
- Uses Nelson-Oppen or variant
- Must agree on arrangement of shared terms


Arithmetic
Arrays
UF
Core
Bit-Vectors
assertions
explanations
conflicts
lemmas
propagations
SAT Solver
DPLL



Theory Solvers

- Decide conjunctions of literals
- Incremental
- Backtrackable
- Conflict Generation
- Theory Propagation

Outline

Eager vs Lazy

Eager SMT Solvers

- Read input formula
- Apply rewriting and simplification
- Translate to SAT and run SAT solver

Lazy SMT Solvers

- SAT solver cooperates with multiple theory solvers
- Each theory solver only sees a conjunction of literals in its theory
- Generic theory combination mechanism

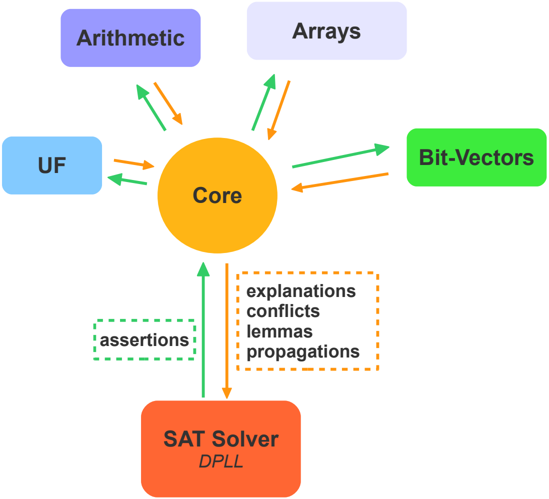
Eager vs Lazy for Bit-vectors

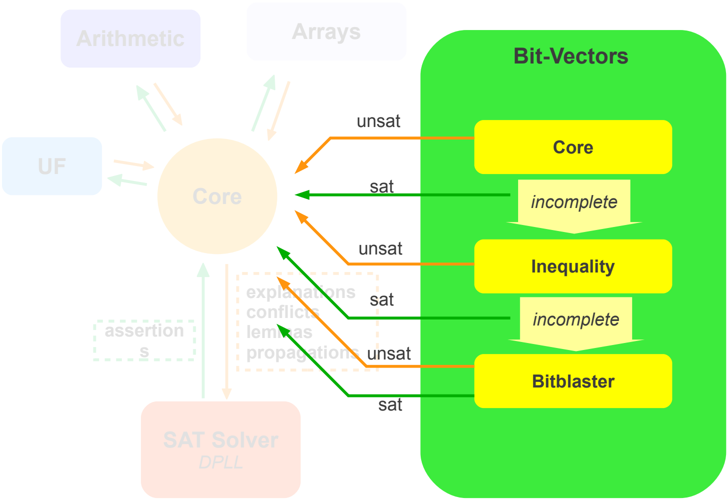
Eager bit-blasting solvers

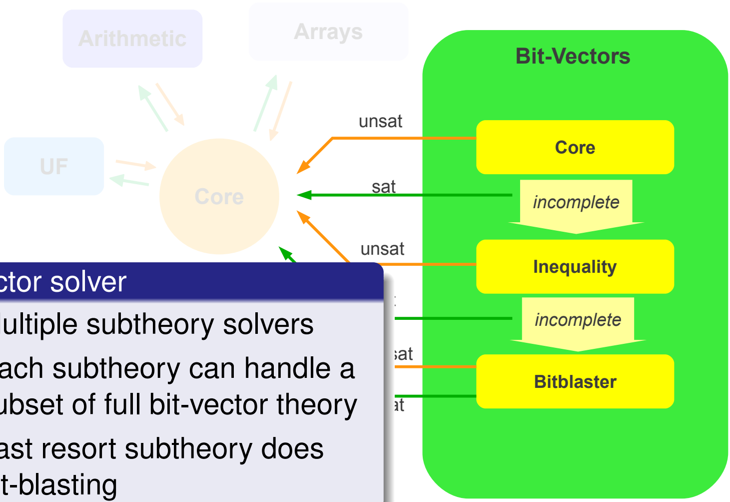
- Current state-of-the-art, but
- Benefit from high-level reasoning only via pre-solve rewriting
- Complexity grows with word size
- Requires monolithic approach
- Not clear how to combine with other theories in general

Lazy solver

- Can integrate high-level reasoning during solving
- Can focus only on the literals in the current search
- Clear mechanism for combining theories


Arithmetic
Arrays
UF
Core
Bit-Vectors
assertions
explanations
conflicts
lemmas
propagations
SAT Solver
DPLL


Arithmetic
Arrays
UF
Core
assertions
explanations
conflicts
lemmas
propagations
SAT Solver
DPLL
Bit-Vectors
unsat
Core
sat
incomplete
unsat
Inequality
sat
incomplete
unsat
Bitblaster
sat



Bit-vector solver

- Multiple subtheory solvers
- Each subtheory can handle a subset of full bit-vector theory
- Last resort subtheory does bit-blasting

Core Solver

Σ_c

constants	$\mathbf{0} :: [1]$, $\mathbf{1} :: [1]$
equal	$\_ \approx \_ :: [n], [n]$ for all $n \geq 0$
concat	$\_ \circ \_ :: [m], [n] \to [m+n]$ for all $m, n \geq 0$
extract	$\_[i:j] :: [m] \to [i-j+1]$ for all $m > i, j \geq 0$ with $i - j \geq -1$

Example

$x[7:4] \circ y \approx x[4:1] \circ z \wedge x[7:7] \not\approx x[1:1]$

Core Solver

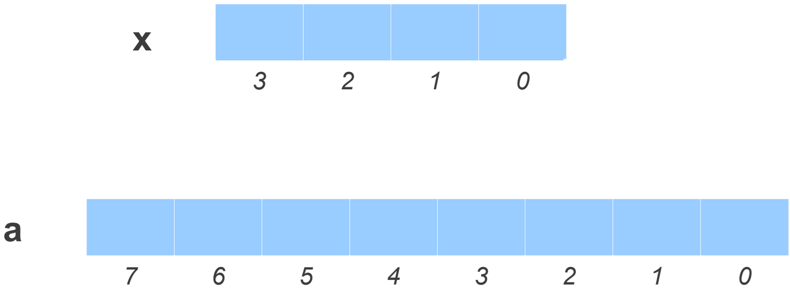
Core solver

- Reasons about equalities and disequalities
- Can also reason about concat and extract

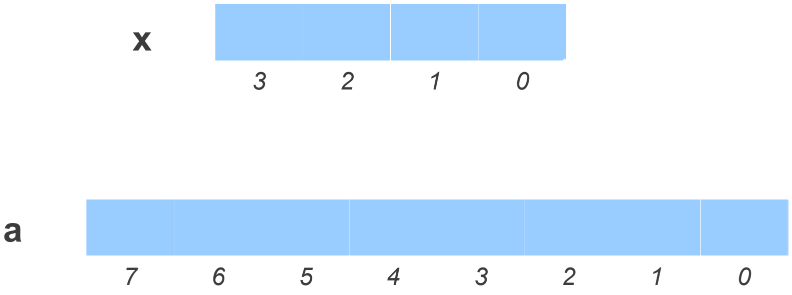
Core solver algorithm

- 1. Until fixed point is reached: propagate all slicings across equations and disequations
- 2. Split equations along slice points
  - e.g. $x_{[3]} \circ y_{[4]} \approx z_{[7]} \longrightarrow x_{[3]} \approx z[6:4] \wedge y_{[4]} \approx z[3:0]$
- 3. Check if normal forms of two disequalities are in the same equivalence class

Core Solver Example

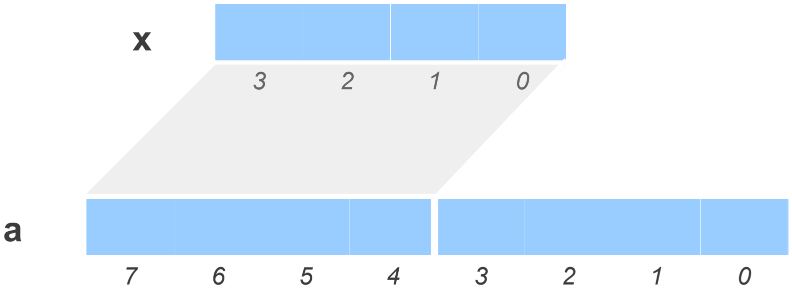


Core Solver Example



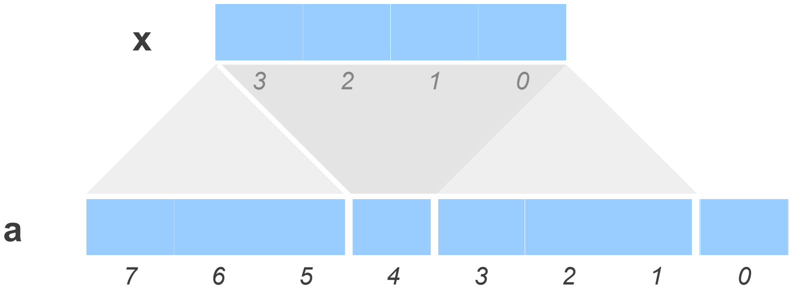
$$a\,[7:4] = x$$

Core Solver Example



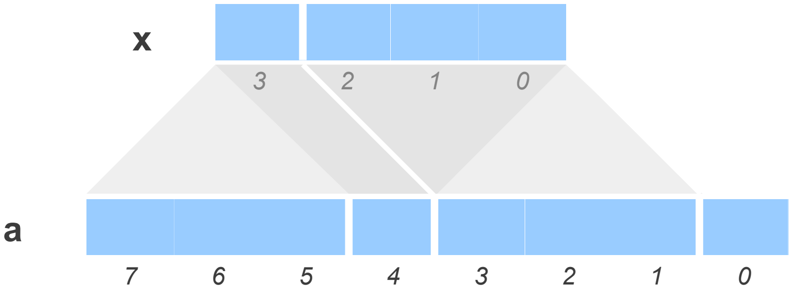
a [7 : 4] = x

Core Solver Example



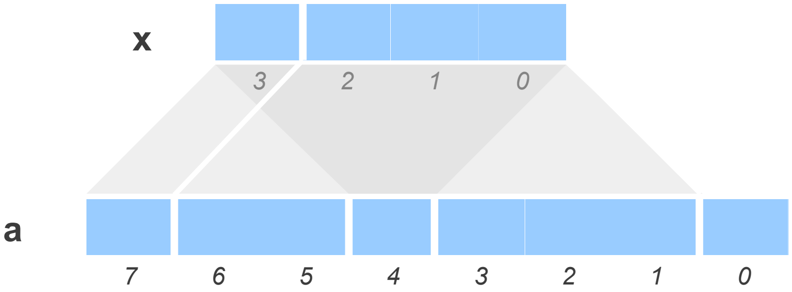
a [7 : 4] = x
a [4 : 1] = x

Core Solver Example



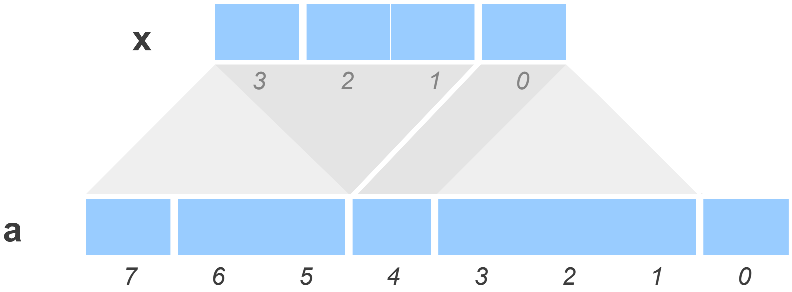
a [7 : 4] = x
a [4 : 1] = x

Core Solver Example



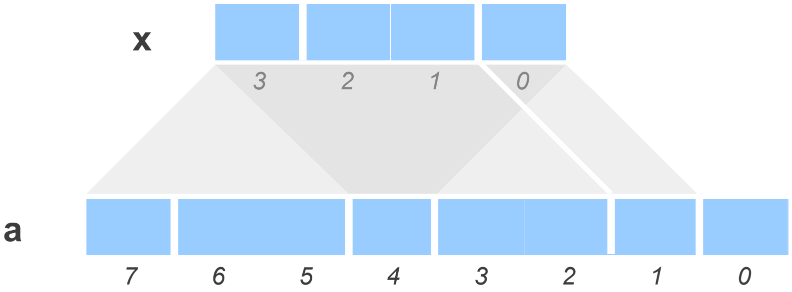
a [7 : 4] = x
a [4 : 1] = x

Core Solver Example



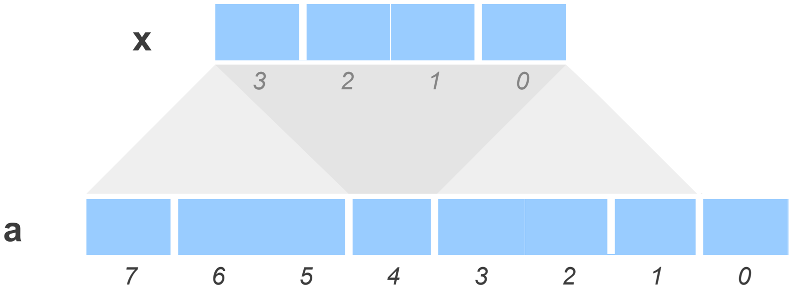
a [7 : 4] = x
a [4 : 1] = x

Core Solver Example



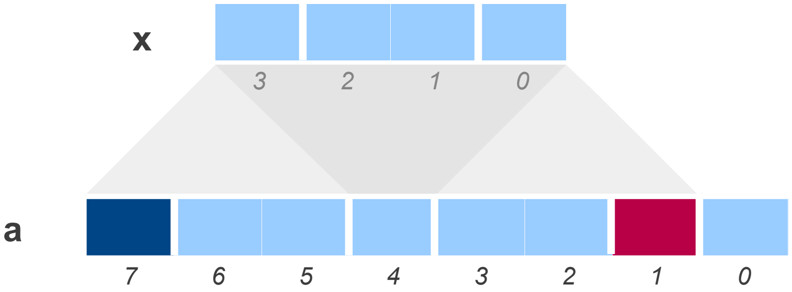
a [7 : 4] = x
a [4 : 1] = x

Core Solver Example



$$a[7:4] = x$$
$$a[4:1] = x$$
$$a[7:7] \neq a[1:1]$$

Core Solver Example

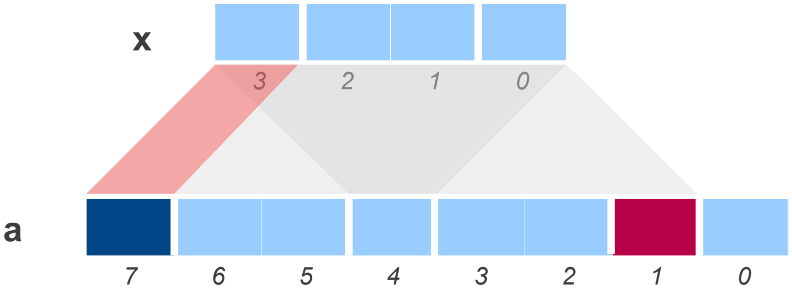


$a\,[7:4] = x$

$a\,[4:1] = x$

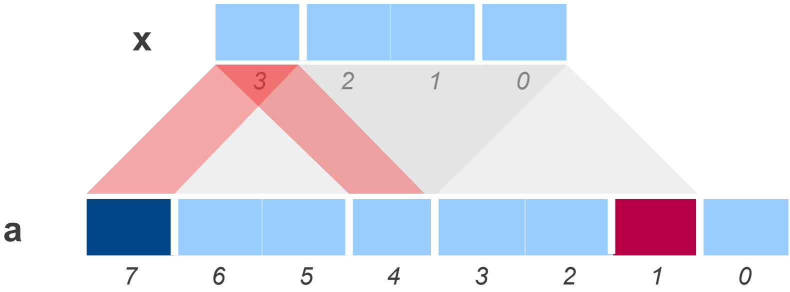
$a\,[7:7] \neq a\,[1:1]$

Core Solver Example



$a[7:4] = x$
$a[4:1] = x$
$a[7:7] \neq a[1:1]$

Core Solver Example

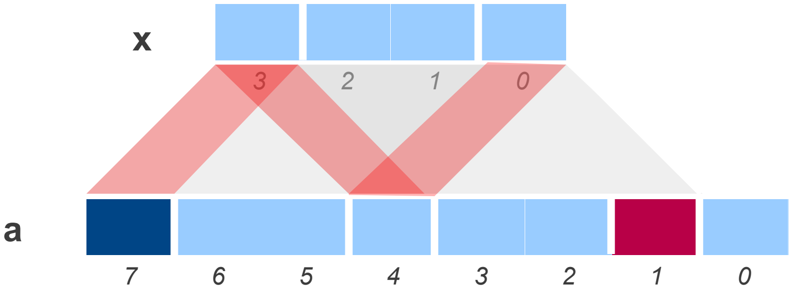


$a\,[7:4] = x$

$a\,[4:1] = x$

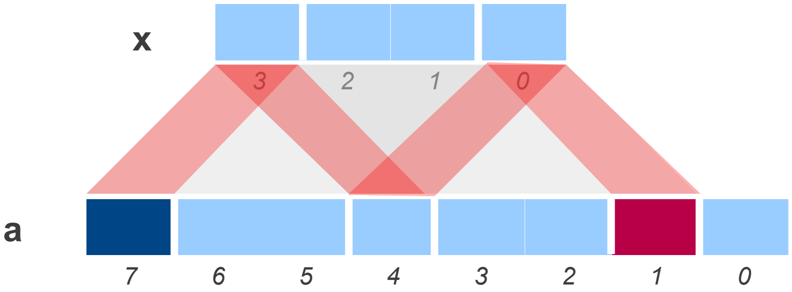
$a\,[7:7] \neq a\,[1:1]$

Core Solver Example



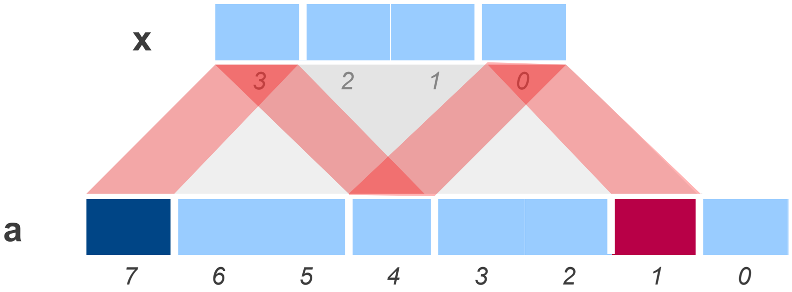
$a[7:4] = x$
$a[4:1] = x$
$a[7:7] \neq a[1:1]$

Core Solver Example



$$a\,[7:4] = x$$
$$a\,[4:1] = x$$
$$a\,[7:7] \neq a\,[1:1]$$

Core Solver Example



$a\,[7:4] = x$

$a\,[4:1] = x$

$a\,[7:7] \neq a\,[1:1]$

Inequality Solver

Σ_o

constants	$\mathbf{0} :: [1], \mathbf{1} :: [1]$
equal	$\_ \approx \_ :: [n], [n]$ for all $n \geq 0$
less	$\_ < \_ :: [n], [n]$ for all $n \geq 0$
leq	$\_ \lesssim \_ :: [n], [n]$ for all $n \geq 0$

Example

$$b < c \land c < 3 \land a < c \land a < b \land 2 \lesssim a$$

The CVC4 inequality solver is complete for constraints including only equalities, disequalities and inequalities

Inequality Solver

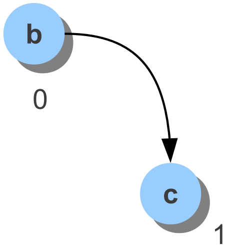
Graph construction

- Build incremental graph based on constraints
- Edge with weight 1 from x to y if $x < y$
- Edge with weight 0 form x to y if $x \lesssim y$

Model construction

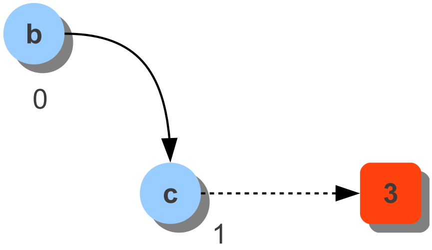
- Label each root with 0 and each constant with itself
- If some unlabeled node, all of whose parents are labeled
  - Label with the max of parents plus weight from that parent, and repeat
- If constant node c has parent such that label of parent plus weight from parent is larger than c, conflict

Inequality Solver Example



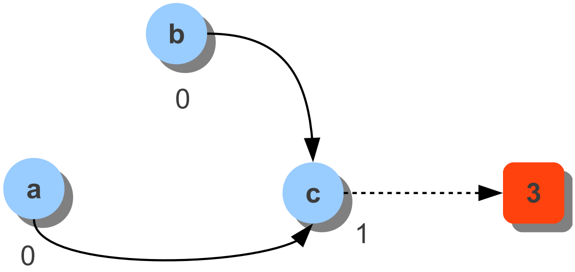
$$b < c, c < 3, a < c, a < b, 2 \leq a$$

Inequality Solver Example



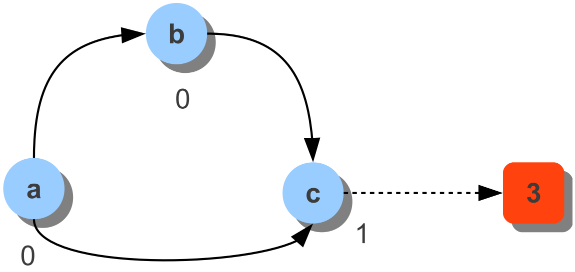
$$b < c, c < 3, a < c, a < b, 2 \leq a$$

Inequality Solver Example



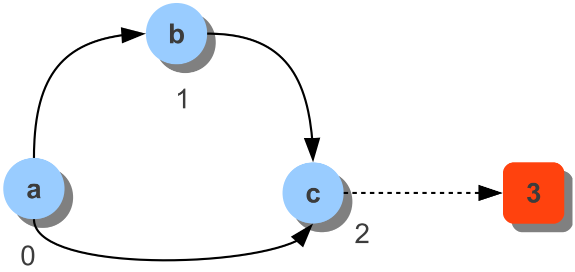
$$b < c, c < 3, a < c, a < b, 2 \leq a$$

Inequality Solver Example



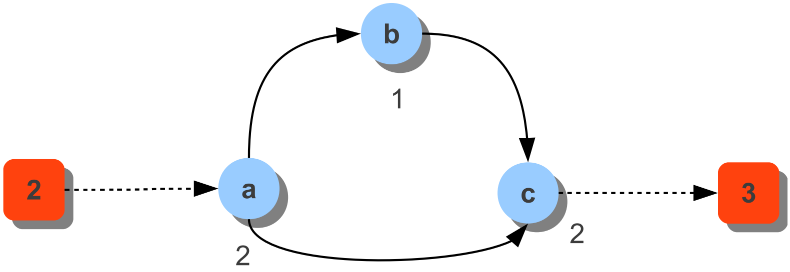
$$b < c, c < 3, a < c, a < b, 2 \leq a$$

Inequality Solver Example



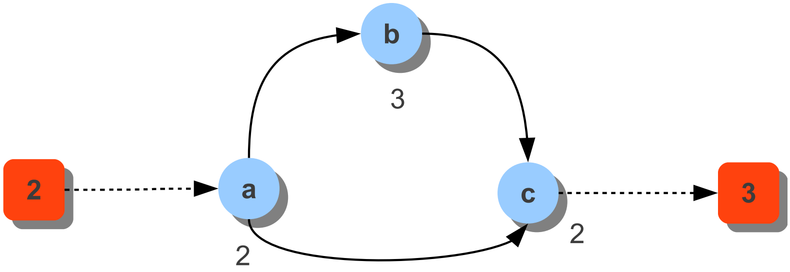
$$b < c, c < 3, a < c, a < b, 2 \leq a$$

Inequality Solver Example



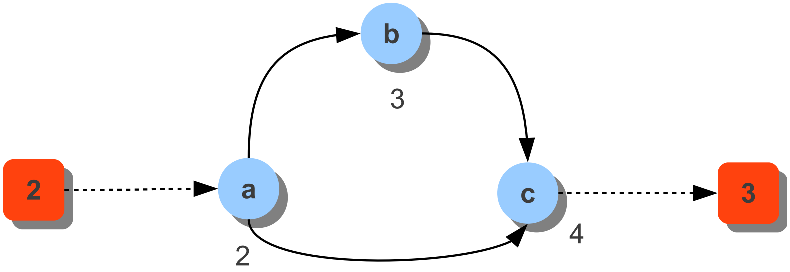
$$b < c, c < 3, a < c, a < b, 2 \leq a$$

Inequality Solver Example



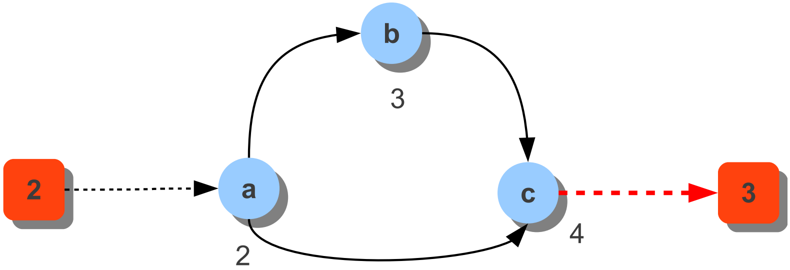
$b < c, c < 3, a < c, a < b, 2 \leq a$

Inequality Solver Example



$$b < c, c < 3, a < c, a < b, 2 \leq a$$

Inequality Solver Example



$$b < c, c < 3, a < c, a < b, 2 \leq a$$

DPLL(*T*) Bit-Blasting Solver

Bit-blasting solver

- Uses dedicated SAT solver (SAT_{bv}) for bit-vector reasoning
- Uses the *solve with assumptions* SAT solver feature, supported by many SAT solvers

Incremental SAT

Given propositional formula ϕ and literals $l_1, l_2, \ldots, l_n$ as unit clause assumptions, a call to the SAT_{bv} solver *SolveAssumps*$(\phi, l_1 \ldots l_n)$ will decide whether $\phi \wedge l_1 \wedge \ldots \wedge l_n$ holds.

Solver Requirements

Features of all solvers

- Incremental
- Backtrackable
- Able to produce conflicts
- Able to produce theory propagations
- Able to produce explanations for propagations

Outline

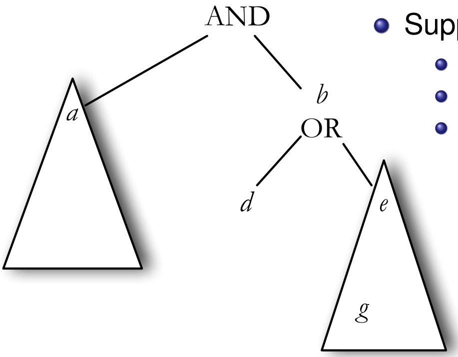
Decision Heuristic

Idea

Retain original structure of formula in order to

- Restrict SAT splits to relevant literals
- Stop when top formula is justified (even if not all literals are assigned)

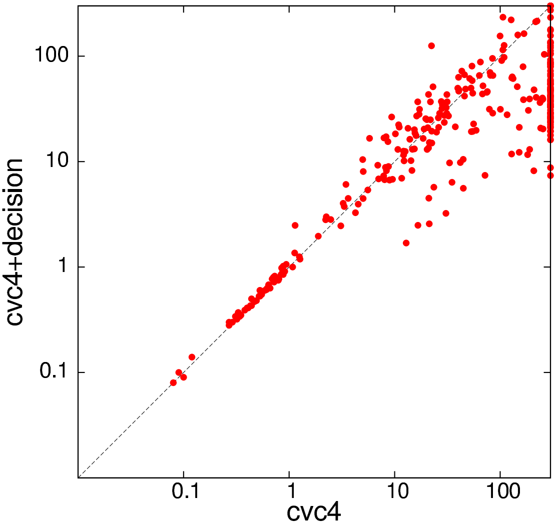
Decision Heuristic Example



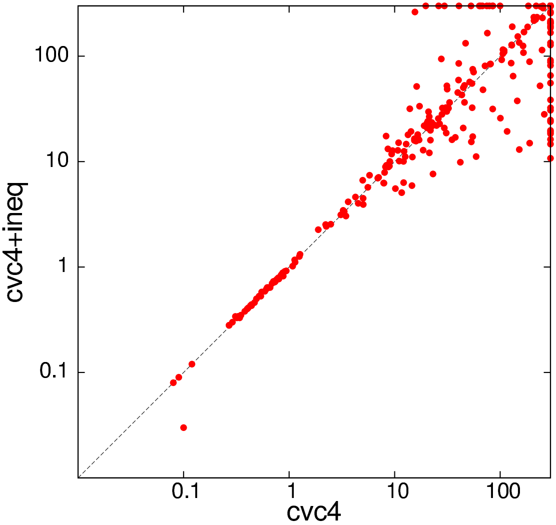
- We wish root to be true, so *a* and *b* must be true
- Suppose we set *d* to true, then:
  - *b* and *d* are justified
  - subtree at *a* is relevant
  - subtree at *e* (including node *g*) is not relevant

Outline

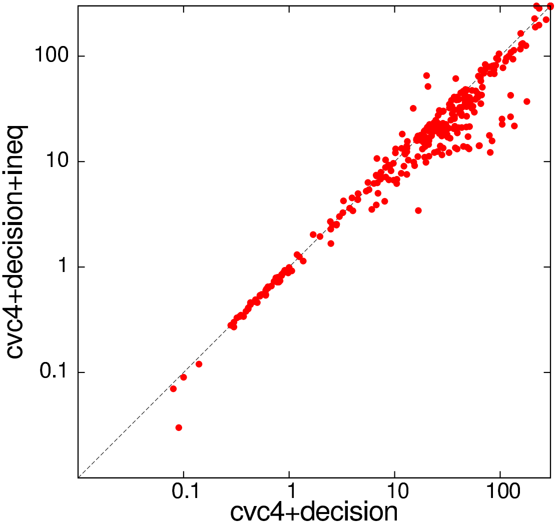
Effect of Decision Heuristic



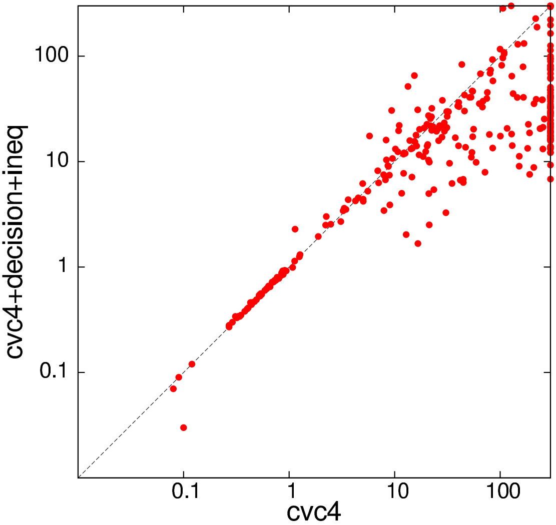
Effect of Inequality Solver



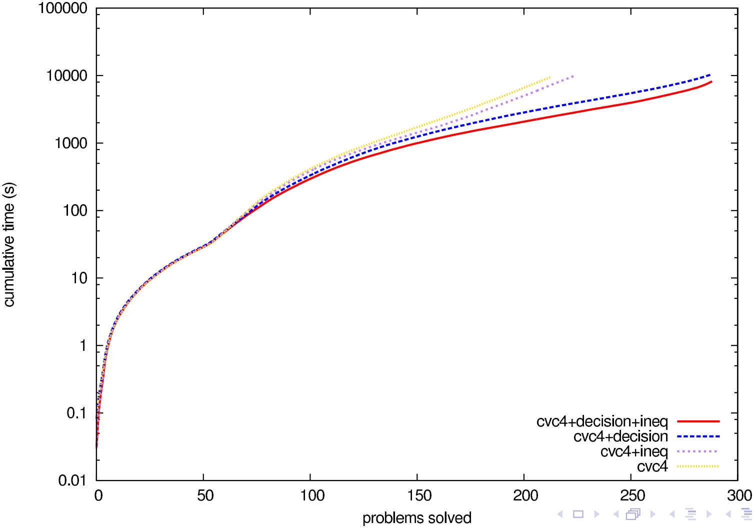
Effect of Inequality Solver on top of Decision Heuristic



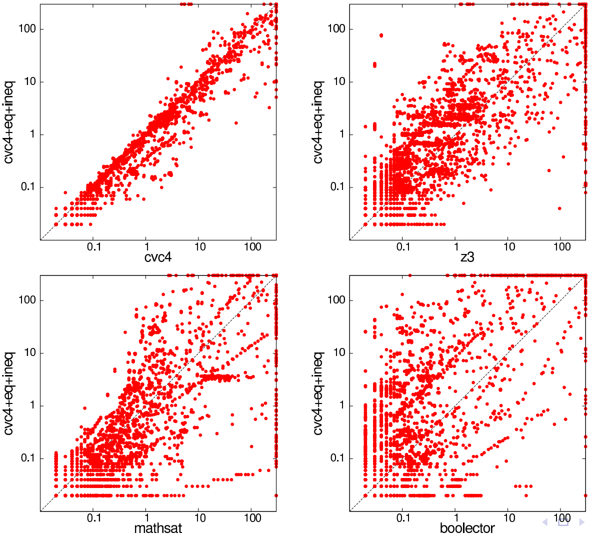
Effect of Both



Cactus comparison plot



Results on all QF_AUFBV SMT-LIB benchmarks



QF_AUFBV Results Summary

solver	Results on 15267 benchmarks solved	time (s)
boolector	15152	13578.66
cvc4+eq+ineq	15046	22697.36
cvc4	15016	25045.50
mathsat	14958	19605.99
z3	14877	16583.71

Elliptic Curve Cryptography

Verification of Elliptic Curve Cryptography, Joe Hendrix, Galois, Inc., HCSS 2012

Summary

- Goal: Create an efficient verified implementation of ECDSA in Java
- SMT verification conditions generated by comparing forward simulation of Java byte code to specification
- Resulting SMT formulas use bit-vectors, arrays, and uninterpreted functions (QF_AUFBV)

Results on ECC Benchmarks

Results on 138 benchmarks

Solver	Solved
boolector*	100
cvc4	137
mathsat	133
yices2	132
z3	133

*Note: boolector does not support UF but solves all benchmarks in its supported logic

Summary

New bit-vector subtheory solvers

- Core theory
- Inequality theory
- Bit-blaster theory now only called if previous two theories can't handle the constraints

Structural Decision Heuristic

- Chooses only relevant atoms to split on
- Solver can stop early if all assertions are justified

Summary

Results

- Reasonable improvement on pure QF_BV benchmarks, but still trails best eager solvers
- Very competitive results on QF_AUFBV benchmarks
  - Supports our hypothesis that a lazy Bit-Vector solver is good for theory combination
  - Can solve a number of benchmarks no other solver can solve (SOTA solver)

What's next?

- Difference logic/Arithmetic subtheory solver
- Better integration of subtheory solvers
- New model-based array solver

The End

Visit the CVC4 web page at:

`http://cvc4.cs.nyu.edu/`

AEG Online:

`http://forallsecure.com/`