The CVC4 SMT Solver

Martin Brain on behalf of Morgan Deters

February 2, 2015
The CVC4 Team

Clark Barrett (NYU)
Cesare Tinelli (U Iowa)
Morgan Deters (NYU)

Kshitij Bansal (NYU)
François Bobot (CEA)
Chris Conway (Google)

Liana Hadarean (NYU)
Dejan Jovanović (SRI)
Tim King (Verimag)

Tianyi Liang (U Iowa)
Andrew Reynolds (EPFL)
Nestan Tsiskaridze (U Iowa)
1 Overview of SMT

2 CVC4
   - Architecture
   - Quantifiers
   - Bit-Vectors
   - Proofs

3 Conclusion
Does this hold? Does this not hold?

\[ 0 < x \]
\[ 0 < y \]
\[ x + y < x \]
Does this hold? Does this not hold?

0 ♣ x

0 ♣ y

x♠ y ♣ x
First Order Logic

**Syntax**

Fix a *signature* $\Sigma$  
(i.e. $\Sigma = \{\spadesuit, \diamondsuit\}$)

**Semantics**

An *interpretation* is 
$M = (D, [\cdot] : \Sigma \rightarrow (2^{D^n}))$

**Satisfiability**

An interpretation $M$ *satisfies* a formula $\phi$:

$M \models \phi$

If $\phi$ evaluated over $D$ (using $[\cdot]$) is true.
Does this hold? Does this not hold?

0 ♠ x
0 ♠ y
x♠y ♠ x

It depends on the interpretation (of ♠ and ♠)!

\[ D = \mathbb{Z} \]
\[ \llbracket ♠ \rrbracket = \prec \mathbb{Z} \]
\[ \llbracket ♠ \rrbracket = +\mathbb{Z} \]
Does this hold? Does this not hold?

0 ♠ x
0 ♣ y
x♠y ♠ x

It depends on the interpretation (of ♠ and ♣)!

\[ D = \{00, 01, 10, 11\} \]

\[ \left[\♠\right] = \text{bvult} \]

\[ \left[\♣\right] = \text{bvplus} \]
How Do We Fix The *Meaning* of Symbols?

**Option 1 – Axiomatic**

\[ M \models \text{Axioms} \Rightarrow \phi \]

\[
\text{axioms} = \forall a, b, c . a \clubsuit b \land b \clubsuit c \Rightarrow a \clubsuit c
\]

\[
= \forall a . \neg a \clubsuit a
\]

\[ \ldots \]

**Pros**

+ Easy to implement
+ Flexible
+ Can add theorems

**Cons**

- All formulae quantified
- Axioms not always simple
- Hard to solve
How Do We Fix The *Meaning* of Symbols?

Option 2 – Algebraic

Fix signature $\Sigma'$ and its interpretation $M' = (D, \llbracket . \rrbracket : \Sigma' \to (2^{D^n}))$.

$$D = \mathbb{Z}, \llbracket \clubsuit \rrbracket = <\mathbb{Z}, \llbracket \spadesuit \rrbracket = +\mathbb{Z}$$

Is there $M$ extension of $M'$ such that:

$$M \models \phi$$

Pros

- Fast decision procedures
- Counter-examples
- Few quantifiers

Cons

- Theory has to be built into solver
- Implementation harder
How Do We Fix The *Meaning* of Symbols?

Option 2 – Algebraic

Fix signature $\Sigma'$ and its interpretation $M' = (D, []. : \Sigma' \rightarrow (2^{D^n}))$.

$$D = \mathbb{Z} \quad [\spadesuit] = < \mathbb{Z} \quad [\heartsuit] = + \mathbb{Z}$$

Is there $M$ extension of $M'$ such that:

$$M \models \phi$$

Satisfiability Modulo Theories (SMT)

**Pros**

+ Fast decision procedures
+ Counter-examples
+ Few quantifiers

**Cons**

- Theory has to be built into solver
- Implementation harder
The SMT-LIB Initiative

http://smt-lib.org

- International initiative
- Rigorously standardise descriptions of theories for SMT
  Arithmetic ($\mathbb{Z}$ and $\mathbb{R}$), arrays, bit-vectors, floating-point
  (in preparation strings, data-types, sets ...)
- Promote common syntax for SMT interactions
- Benchmarks
- Annual competition
## SMT Solvers

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>License</th>
<th>SMT-LIB</th>
<th>CVC</th>
<th>DIMACS</th>
<th>Features</th>
<th>API</th>
<th>SMT-COMP</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABsolver</td>
<td>Linux beagleos, Windows</td>
<td>GPLv3</td>
<td>v1.2</td>
<td>No</td>
<td>No</td>
<td>built-in theories</td>
<td>C++</td>
<td>no</td>
<td>DPLL-based</td>
</tr>
<tr>
<td>Barcelogic</td>
<td>Linux</td>
<td>Proprietary</td>
<td>v1.2</td>
<td>No</td>
<td>No</td>
<td>built-in theories</td>
<td>OCaml</td>
<td>2009</td>
<td>DPLL-based, congruence closure</td>
</tr>
<tr>
<td>Beaver</td>
<td>Linux, Windows</td>
<td>BSD</td>
<td>v1.2</td>
<td>No</td>
<td>No</td>
<td>built-in theories</td>
<td>OCaml</td>
<td>2009</td>
<td>SAT-solver based</td>
</tr>
<tr>
<td>Boolector</td>
<td>Linux</td>
<td>GPLv3</td>
<td>v1.2</td>
<td>No</td>
<td>No</td>
<td>built-in theories</td>
<td>C</td>
<td>2009</td>
<td>SAT-solver based</td>
</tr>
<tr>
<td>CVC3</td>
<td>Linux</td>
<td>BSD</td>
<td>v1.2</td>
<td>Yes</td>
<td>Yes</td>
<td>built-in theories</td>
<td>C/C++</td>
<td>2010</td>
<td>proof output to HOL</td>
</tr>
<tr>
<td>CVC4</td>
<td>Linux, beagleos, Windows</td>
<td>BSD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>built-in theories</td>
<td>C++</td>
<td>2010</td>
<td>version 1.4 released July 2014</td>
</tr>
<tr>
<td>Decision</td>
<td>Linux</td>
<td>Apache</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td>OCaml</td>
<td>no</td>
<td>DPLL-based</td>
</tr>
<tr>
<td>Procedure Toolkit (DPT)</td>
<td>Linux</td>
<td>Proprietary</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td>OCaml</td>
<td>no</td>
<td>DPLL-based</td>
</tr>
<tr>
<td>ISAT</td>
<td>Linux</td>
<td>Proprietary</td>
<td>No</td>
<td></td>
<td></td>
<td>empty theory, linear arithmetic, bitvectors, arrays</td>
<td>C/C++, Python, Java</td>
<td>2010</td>
<td>DPLL-based</td>
</tr>
<tr>
<td>MathSAT</td>
<td>Linux</td>
<td>Proprietary</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>empty theory, linear arithmetic, bitvectors, arrays</td>
<td>C/C++</td>
<td>2010</td>
<td>DPLL-based</td>
</tr>
<tr>
<td>MiniSmt</td>
<td>Linux</td>
<td>LGPL</td>
<td>partial v2.0</td>
<td>Yes</td>
<td></td>
<td>non-linear arithmetic</td>
<td>C++</td>
<td>2010</td>
<td>SAT-solver based, Yices-based</td>
</tr>
<tr>
<td>OpenSMT</td>
<td>Linux, beagleos, Windows</td>
<td>GPLv3</td>
<td>partial v2.0</td>
<td>Yes</td>
<td></td>
<td>empty theory, differences, linear arithmetic, bitvectors</td>
<td>C++</td>
<td>2011</td>
<td>Lazy SMT Solver</td>
</tr>
<tr>
<td>Sage</td>
<td>Linux</td>
<td>Proprietary</td>
<td>v1.2</td>
<td></td>
<td></td>
<td>built-in theories</td>
<td>none</td>
<td>2009</td>
<td>Focuses on generating high quality interpolants.</td>
</tr>
<tr>
<td>SMTInterpol</td>
<td>Linux, beagleos, Windows</td>
<td>LGPLv3</td>
<td>v2.0</td>
<td></td>
<td></td>
<td>built-in theories</td>
<td>Java</td>
<td>2012</td>
<td>Toolbox offering theory solver more development of SMT solvers for nonlinear arithmetic (NRA). Example embedded OpenSMT available.</td>
</tr>
<tr>
<td>SMT-RAT</td>
<td>Linux, beagleos, Windows</td>
<td>GPLv3</td>
<td>v2.0</td>
<td>No</td>
<td>No</td>
<td>built-in theories</td>
<td>C++</td>
<td>no</td>
<td>Can implement new theories using Handling Rules</td>
</tr>
<tr>
<td>SOONCLAR</td>
<td>Linux, Windows</td>
<td>Proprietary</td>
<td>partial v2.0</td>
<td></td>
<td></td>
<td>built-in theories</td>
<td>C</td>
<td>2010</td>
<td>SAT-solver based</td>
</tr>
<tr>
<td>SpeedSMT</td>
<td>Linux, beagleos, Windows</td>
<td>Proprietary</td>
<td>v1.2</td>
<td></td>
<td></td>
<td>built-in theories</td>
<td>C</td>
<td>2009</td>
<td>SAT-solver based</td>
</tr>
</tbody>
</table>
1 Overview of SMT

2 CVC4
   - Architecture
   - Quantifiers
   - Bit-Vectors
   - Proofs

3 Conclusion
CVC4 : The Cooperating Validity Checker, Version 4

- Full-fledged open-source SMT solver
- Jointly developed at NYU and U. Iowa
- Competitive with other top solvers
- Theories include linear arithmetic, arrays, bit-vectors, algebraic data-types, uninterpreted functions, quantifiers
- Support for CVC, SMT-LIB 1.2/2.0/2.5, TPTP (FOF) input formats

http://cvc4.cs.nyu.edu/
History of CVC

**SVC**  1996, own SAT solver

**CVC**  Chaff, optimized internal design

**CVC Lite**  2003, rewrite to make more flexible supported quantifiers

**CVC3**  major overhaul
better DP implementations

**CVC4**  first stable release 2012
complete redesign of internal architecture
significant performance improvements

http://cvc4.cs.nyu.edu/
## Using CVC4

**Stand-alone tool**
- SMT-LIB 1.2, 2.0, 2.5 (plus Z3’s extensions)
- Native CVC language
- TPTP format

**Library**
- Bindings for:
  - C++
  - Java
  - OCaml

High-Level Architecture

PropEngine

SAT Solver

assertions

conflicts

propagations

explanations

lemmas

TheoryEngine

UF

Arith

BV

Array

Quantifiers
Quantifiers

Compactness

\[ \phi \text{ is unsatisfiable} \iff \text{There is a finite set } S \text{ of instantiation of the quantifiers that is also unsatisfiable} \]

How do we find \( S \)? How do we know when we have found it?
# E-Matching

Use unification to identify which terms could be relevant

## Optimisations

- Minimise the number of new terms created
- Equality aware (one unification per congruence class)
- Prefer instantiations that create unit clauses
- Theory propagation aware

\[
\psi(x, y, z) \land \forall a, b \cdot \phi(a, b) \sim \psi(x, y, z) \land \phi(x, y) \land \phi(x, 4)
\]
Model Based Quantifier Instantiation

**MBQI**

1. Generate model of the ground part of $\phi$
2. Search the space of quantification
3. Use counter-examples to pick instantiations

$$\psi(x, y, z) \land \forall a, b \cdot \phi(a, b) \leadsto \psi(x, y, z)$$

$$\leadsto x = 1, y = 4, z = 23 \land \neg \phi(a, b)$$

$$\leadsto \psi(x, y, z) \land \phi(17, 42)$$
## Finite Model Finder

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td>Generate a model of the ground part of $\phi$</td>
</tr>
<tr>
<td><strong>2</strong></td>
<td>Minimise the number of equality classes of values</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td>Complete instantiation for these values only</td>
</tr>
</tbody>
</table>
Also...

- Conflict-based instantiation
- Induction
- Local theory extensions
- Rewrite rules
Eager Bit-Vector Solver

Preprocessor

Bit-blaster

SAT

sat unsat
Lazy Bit-Vector Solver

- SAT\textsubscript{main} CDCL
- Bit-vector Theory
  - Equality
  - Ineq
  - In-processing
  - BB Solver
    - SAT\textsubscript{bv}
Proof Generation

- Solver is beyond current functional verification.
- Instead, generate proof for unsatisfiable formulae
- Current proof uses LFSC meta-proof framework
Proof Module in CVC4

PropEngine
- toCNF
- CDCL SAT
- Propagation conflict lemma
- propagation

TheoryEngine
- core
- bit-vectors
- uf

ProofManager
- cnf map
- resolution steps
- CNF Proof
- SAT Proof

CNF Proof
- resolution steps
- T-lemmas
- cnf map

TheoryProofEngine
- bv-lemmas
- uf-lemmas

Bit-vector Proof
- Bit-blast Proof
- CNF Proof
- SAT Proof

Uf Proof
1 Overview of SMT

2 CVC4
   - Architecture
   - Quantifiers
   - Bit-Vectors
   - Proofs

3 Conclusion
Getting CVC4

- Binaries for x86/Linux, x86/Windows, x64/Linux
  http://cvc4.cs.nyu.edu/download/
- Packaged in Debian and Fedora
- MacPorts package
- Build from source
  https://github.com/CVC4/CVC4
- Try it online!
  http://cvc4.cs.nyu.edu/tryit/
The Future

- New Theories: strings, finite sets, floating-point
- Unsatisfiable cores
- Proofs
- Quantifier elimination
- Optimisation problems
Conclusions

**SMT**  Fix interpretation, not axiomatisation

**CVC4**  State-of-the-art, open source SMT solver

**You?**  Always looking for new users and new collaborators!
Conclusions

SMT  Fix interpretation, not axiomatisation

CVC4  State-of-the-art, open source SMT solver

You?  Always looking for new users and new collaborators!

Thank you for your time and attention.