A Tour of CVC4

Morgan Deters
mdeters@cs.nyu.edu

Cesare Tinelli
cesare-tinelli@uiowa.edu

Andrew Reynolds
andrew.reynolds@epfl.ch

Clark Barrett
barrett@cs.nyu.edu

Tim King
tim.king@imag.fr

CVC4 is supported in part by the Air Force Office of Scientific Research, Google, Intel Corporation, the National Science Foundation, and Semiconductor Research Corporation
The CVC4 Team

Clark Barrett (NYU)
Cesare Tinelli (U Iowa)

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

Morgan Deters (NYU)
Liana Hadarean (NYU)
Dejan Jovanović (SRI)

Tim King (Verimag)
Tianyi Liang (U Iowa)
Andrew Reynolds (EPFL)
Agenda

- Introduction and status report for CVC4
- Arithmetic
- Quantifiers (finite model finding)
- Examples/demos
Automated Reasoning

• Historically automated reasoning meant uniform proof procedures for FOL

• More recent trend is decidable fragments
  – Domain-specific reasoning
  – Equality
  – Arithmetic
  – Data structures (arrays, lists, records)
Automated Reasoning

• Examples

  – **SAT** – propositional, Boolean reasoning
    • efficient
    • expressive (NP) but involved encodings
  – **SMT** – first order, Boolean + DS reasoning
    • loss of efficiency
    • improves expressivity and scalability
Articles mentioning SMT over time
Applications of SMT

- extended static checking
- predicate abstraction
- model checking
- scheduling
- test generation
- synthesis
- (in)feasible paths
- verification
More on Expressivity

• Many theories of interest have efficient decision procedures for conjunctions of facts

• ...but in practice we need arbitrary Boolean combinations
  – also combined theory constraints
  – quantifiers
Architecture of SMT

Properties to (dis)prove
Context Strategies Formulas Queries
Proofs Translated counterexamples
Models Counterexamples Proof hints

Input Interfaces
SMT Engine
Theory Engine
Prop Engine

SAT core
Theory Implementations Decision Procedures
Arithmetic (and specialized fragments)
Arrays
Inductive data types
Bit-vectors
Uninterpreted Functions
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
CVC3 to CVC4

- CVC3 was very featureful…
  - support for many theories, proofs, quantifiers…

- But also suffered from serious problems
  - performance was problematic
### QF_UF (100%)

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MathSAT 5</td>
<td>200 / 205</td>
<td>9056.2 s</td>
</tr>
<tr>
<td>OpenSMT-1.0-alpha</td>
<td>186 / 205</td>
<td>7290.7 s</td>
</tr>
<tr>
<td>veriT 201007</td>
<td>175 / 205</td>
<td>36059.4 s</td>
</tr>
<tr>
<td>CVC3 2.3</td>
<td>127 / 205</td>
<td>121044.3 s</td>
</tr>
</tbody>
</table>

### QF_RDL (100%)

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSMT-1.0-alpha</td>
<td>157 / 170</td>
<td>9475.5 s</td>
</tr>
<tr>
<td>veriT 201007</td>
<td>137 / 170</td>
<td>9932.2 s</td>
</tr>
<tr>
<td>CVC3 2.3</td>
<td>68 / 170</td>
<td>10386.8 s</td>
</tr>
<tr>
<td>Z3.2, 2006</td>
<td>160 / 170</td>
<td>20449.6 s</td>
</tr>
<tr>
<td>Yices 2 prd</td>
<td>175 / 170</td>
<td>20579.7 s</td>
</tr>
</tbody>
</table>

### QF_IDL (100%)

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSMT-1.0-alpha</td>
<td>180 / 207</td>
<td>5880.7 s</td>
</tr>
<tr>
<td>veriT 201007</td>
<td>163 / 207</td>
<td>11624.2 s</td>
</tr>
<tr>
<td>CVC3 2.3</td>
<td>140 / 207</td>
<td>6930.4 s</td>
</tr>
<tr>
<td>Barcelogic 1.2</td>
<td>160 / 207</td>
<td>20434.2 s</td>
</tr>
<tr>
<td>Sateen-3.1</td>
<td>175 / 207</td>
<td>20578.7 s</td>
</tr>
</tbody>
</table>

### QF_BV (100%)

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>simplifyingSTP</td>
<td>155 / 205</td>
<td>8415.3 s</td>
</tr>
<tr>
<td>SONOLAR r252</td>
<td>140 / 205</td>
<td>14033.6 s</td>
</tr>
<tr>
<td>CVC3 2.3</td>
<td>68 / 205</td>
<td>7807.8 s</td>
</tr>
<tr>
<td>MathSAT 4.3, 2009 winner</td>
<td>171 / 205</td>
<td>3638.1 s</td>
</tr>
</tbody>
</table>
CVC3 to CVC4

- CVC3 was very featureful…
  - support for many theories, proofs, quantifiers…

- But also suffered from serious problems
  - performance was problematic
  - very difficult to extend for research
  - could not rapidly prototype new ideas
CVC4

• Complete redesign of internal architecture
• Five years in the making
• Performance a big improvement
  – placed 1\textsuperscript{st} in 14 of 32 divisions of SMT-COMP
  – performs well also in CASC
  – competitive for many common SMT uses

• …without sacrificing features
CVC4 is Expressive

• Boolean combinations of theory constraints
• Combination of theories
  – arrays of integers, functions on arrays, …
• Quantifiers
• Verification, test generation, synthesis, feasibility
• Models, proofs, unsatisfiable cores
CVC4 is Expressive

• (Linear) arithmetic over integer, rational
• Bitvectors
• Strings
• Functions
• Arrays
• Inductive datatypes
• Finite sets
CVC4 is Expressive

• Quantifiers

• If CVC4 doesn’t have support for a theory,
  – axiomatize it
Standardization

• Fully supports SMT-LIB standard
  – v1.2, v2.0, v2.5 (draft)
  – supports much of Z3’s extended command set
• Supports native CVC format
• Supports TPTP format
SMT-LIB – http://smt-lib.org

- International initiative
- Rigorously standardize descriptions of background theories for SMT
- Promote common syntax for SMT interactions
- Benchmarks
- Annual competition
SMT-LIB Command Language

- Declaring a logic
  
  (set-logic QF_UF)

- Setting an option
  
  (set-option :produce-models true)

- Declaring constants
  
  (declare-fun p () Bool)

- Making assertions
  
  (assert (or p q))
SMT-LIB Command Language

• Checking satisfiability
  \texttt{(check-sat)}

• Extracting a model
  \texttt{(get-model)}
SMT-LIB example
New and Upcoming Features

• Theory of **strings**
• Theory of **finite sets**
• Theory of **floating point**

• **Unsatisfiable cores** (for all theories)
• **Proofs** (under development, for some theories)

• **Better control** of preprocessing
Longer term

• More theories
• Increased proof support
• Automatic configuration of heuristics
• Quantifier elimination
• Optimization problems
Certificates

• **Satisfiable** comes with a satisfying model
• **Unsatisfiable** comes with a proof (or core)

• Both are fully machine-checkable
  – CVC4 need not be certified free of bugs to rely upon a result
http://cvc4.cs.nyu.edu/tryit/

Try CVC4 Online!

Your input to CVC4:

; Your SMT commands go here.
; You can select a version and input language below.
Circuit example

test is always supposed to be true

When does it hold?

How do we prove it?

One way: by induction on number of clock cycles

Inductive step:
If test is true, it remains so
Circuit example

\[(y = x + 1 \text{ AND } z = x + 2 \text{ AND } x' = \text{IF } a \text{ THEN } x \text{ ELSE } y \text{ AND } y' = \text{IF } a \text{ THEN } y \text{ ELSE } z \text{ AND } z' = \text{IF } a \text{ THEN } z \text{ ELSE } y + 2) \text{ IMPLIES } y' = x' + 1 \text{ AND } z' = x' + 2\]

\[
\begin{align*}
& (z1 \leftrightarrow \neg x1) \land (z0 \leftrightarrow x0) \land \\
& (y1 \leftrightarrow (x1 \oplus x0)) \land (y0 \leftrightarrow \neg x0) \land \\
& (a \rightarrow ((xp1 \leftrightarrow x1) \land (xp0 \leftrightarrow x0))) \land \\
& (\neg a \rightarrow ((xp1 \leftrightarrow y1) \land (xp0 \leftrightarrow y0))) \land \\
& (a \rightarrow ((yp1 \leftrightarrow y1) \land (yp0 \leftrightarrow y0))) \land \\
& (\neg a \rightarrow ((yp1 \leftrightarrow z1) \land (yp0 \leftrightarrow z0))) \land \\
& (a \rightarrow ((zp1 \leftrightarrow z1) \land (zp0 \leftrightarrow z0))) \land \\
& (\neg a \rightarrow ((zp1 \leftrightarrow \neg y1) \land (zp0 \leftrightarrow y0))) \land \\
& (\neg (zp1 \leftrightarrow \neg xp1) \lor \neg (zp0 \leftrightarrow xp0) \lor \\
& \neg (yp1 \leftrightarrow (xp1 \oplus xp0)) \land (yp0 \leftrightarrow \neg xp0)
\end{align*}
\]
Circuit example

\[(y = x + 1 \text{ AND } z = x + 2 \text{ AND } \\
x' = \text{IF } a \text{ THEN } x \text{ ELSE } y \text{ AND } \\
y' = \text{IF } a \text{ THEN } y \text{ ELSE } z \text{ AND } \\
z' = \text{IF } a \text{ THEN } z \text{ ELSE } y + 2) \text{ IMPLIES } \\
y' = x' + 1 \text{ AND } z' = x' + 2\]
Circuit example

\[(y = x + 1 \text{ AND } z = x + 2 \text{ AND} \]
\[x' = \text{IF } a \text{ THEN } x \text{ ELSE } y \text{ AND} \]
\[y' = \text{IF } a \text{ THEN } y \text{ ELSE } z \text{ AND} \]
\[z' = \text{IF } a \text{ THEN } z \text{ ELSE } y + 2 \text{) IMPLIES} \]
\[y' = x' + 1 \text{ AND } z' = x' + 2 \]

(model
(define-fun x () Int (- 2))
(define-fun y () Int (- 1))
(define-fun z () Int 0)
(define-fun |x'| () Int (- 2))
(define-fun |y'| () Int (- 1))
(define-fun |z'| () Int (- 2))
(define-fun a () Bool true))
Arithmetic
Arithmetic in CVC4

- Quantifier-free linear real and integer arithmetic
  QF_LRA, QF_LIA, QF_LIRA
- Constraints of the form:
  \[ x - y \geq -1, \quad y \leq 4, \quad x \neq 5, \quad x + y \geq 6, \quad x < 5 \ldots \]
- Supports efficient theory combination:
  UF, Arrays, Sets, Datatypes
Linear Real Arithmetic

- Given the linear inequalities
  \[ \{x - y \geq -1, \ y \leq 4, \ x + y \geq 6\} \]
  is there an assignment to \(x\) and \(y\) that makes all of the inequalities true?

- Solve using simplex based approaches
Visually

Is an intersection of half planes empty?
Example Simplex Search
Simplex Solvers in CVC4

- 3 exact precision DPs
  - Simplex for DPLL(T)
  - Sum-Of-Infeasibilities (SOI) Simplex [FMCAD'13]
  - FCSimplex (variant of SOI simplex)
- External floating point solver GLPK
Simplex for DPLL(T)

- [CAV'06 Dutertre & de Moura]
- Highly incremental
  - Resumes from previous assignment
  - No backtracking the tableau
- Run after each new constraint
- Good on verification problems
Sum-of-infeasibilities Simplex

- Adds an optimization function $V(X)$
- $V(X)$ is the total amount bounds are violated
- Minimizes $V(X)$ using primal simplex
- “--use-soi”
- Heavy hammer
Direction of Vio(X)
Leveraging LP & MIP for SMT

- GPLK is an LP/MIP Solver
- ‘Call GLPK if the problem seems hard’
- “--use-approx”
- Very heavy hammer
- See my talk Thursday
- Compile against: https://github.com/timothy-king/glpk-cut-log
From Reals to Mixed Integers

- Add IsInt(x) constraints
- First solve real relaxation
  - Ignore IsInt(x) constraints
- If real relaxation is sat:
  - check if assignment from Simplex a(x) satisfies IsInt(x) constraints
  - Refine by branching:
    \[ x \geq 2 \text{ or } x \leq 1 \]
    Cuts from Proofs [Dillig’06]
ITE Preprocessing

- ITE cofactoring [Kim et al. '09]
- Lifting sums out of ites
  \[(\text{ite } c (\text{+ x s}) (\text{+ x t})) \rightarrow x + (\text{ite } c \text{ s t})\]
- GCD factorization
  \[(\text{ite } c \text{ 0 (ite } d \text{ 1024 2048})) \rightarrow 1024 * (\text{ite } c \text{ 0 (ite } d \text{ 1 2}))\]
Non-linear support in CVC4

- Extremely rudimentary support
  - Parsing, rewriting, solving
  - No models
- Rewrites terms into sum of monomials form
  \((x+y)(x-y) \rightarrow x^2 - y^2\)
- Abstracts each monomial as a fresh variable
  - \(x^2, xy, x^2y\) are all new variables
- Usable if you instantiate axioms manually
- (may improve in the future)
Optimization

- (Coming soon)
- Primal Simplex is implemented
- Not yet user accessible
- Will implement
  - Linear Search [Sebastiani ’12]
  - Symba [Li et al. ’14]
CVC4’s Arithmetic Module

- Optimized for challenging QF_LRA and QF_LIA non-incremental benchmarks
- On the lookout for collaborations to motivate improvements
- For hard linear problems, try: “--use-soi” or “--use-approx”
Quantifiers
Quantified Formulas in CVC4

• CVC4 supports multiple techniques:
  – E-matching
  – Conflict-based instantiation [FMCAD 2014]
  – Rewrite rules
  – Induction
  – Finite Model Finding

\Rightarrow Focus in this tutorial
Overview : Finite Model Finding

• Finite Model Finding in SMT
  – Reduction from quantified $\rightarrow$ ground constraints

• Two techniques for scalability:
  – Minimizing model sizes
  – Model-based quantifier instantiation

• Variants of approach/Examples
Finite Model Finding in SMT

• To determine the satisfiability of:

\[ G \cup \forall_{x,y:S} Q(x, y) \]

For all \( x, y \) of sort \( S \)
Finite Model Finding in SMT

• To determine the satisfiability of:

\[ G \cup \forall xy:S.Q(x, y) \]

For all \( x, y \) of sort \( S \)

\[ \Rightarrow \text{If } S \text{ has finite interpretation,} \]

• use finite model finding
Finite Model Finding in SMT

∀xy:S.Q(x,y)

G

Ground Theory Solvers

CVC4

FMF Module
Finite Model Finding in SMT

\[ \forall_{x, y} : S \cdot Q(x, y) \]

Ground Theory Solvers

FMF Module

\[ S = \{a, b, c, d, e\} \]
Finite Model Finding in SMT

- Reduce quantified formula to ground formula(s)
  \[
  \forall xy : S . Q(x, y)
  \]
  \[
  G \land \begin{align*}
  Q(a, a) & \land \ldots \land Q(e, a) \\
  Q(a, b) & \land \ldots \\
  Q(a, c) & \land \ldots \\
  Q(a, d) & \land \ldots \\
  Q(a, e) & \land \ldots \land Q(e, e)
  \end{align*}
  \]

Ground Theory Solvers

FMF Module

\[ S = \{ a, b, c, d, e \} \]
Finite Model Finding in SMT

- Check satisfiability of $G \land Q(a,a) \land \ldots \land Q(e,e)$
  
  \[ G \land (Q(a,a) \land \ldots \land Q(e,e)) \land (Q(a,b) \land \ldots \land Q(e,e)) \land (Q(a,c) \land \ldots \land Q(e,e)) \land (Q(a,d) \land \ldots \land Q(e,e)) \land (Q(a,e) \land \ldots \land Q(e,e)) \land (\forall xy:S.Q(x,y)) \]

- Ground Theory Solvers
- FMF Module

- SAT
- UNSAT

$S = \{a, b, c, d, e\}$
Finite Model Finding in SMT

$G \wedge \forall xy : S . Q(x, y)$

Q(a, a) \wedge ... Q(e, a) \wedge
Q(a, b) \wedge 
Q(a, c) \wedge 
Q(a, d) \wedge 
Q(a, e) \wedge ... Q(e, e)

Ground Theory Solvers

FMF Module

S = \{a, b, c, d, e\}

UNSAT

SAT

SAT
Finite Model Finding in SMT

\[
\forall xy: S. Q(x, y)
\]

\[
Q(a, a) \land \ldots Q(e, a) \land
Q(a, b) \land \ldots
Q(a, c) \land \ldots
Q(a, d) \land \ldots
Q(a, e) \land \ldots Q(e, e)
\]

- Can be very large

Ground Theory Solvers

FMF Module

\( S = \{a, b, c, d, e\} \)

UNSAT

SAT

SAT
Scalable Approach to FMF

• Address large # instantiations by:
  1. Minimizing model sizes
  2. Only consider instantiations that refine model
  • Model-based quantifier instantiation
    – [Ge/deMoura CAV 2009]
1. Minimize model size (Example)

• For $G = \{ a \neq b, b \neq c, c \neq d, d \neq e, e \neq a \}$
1. Minimize model size (Example)

- Has model of size 5:

\[
S = \{a, b, c, d, e\}
\]
1. Minimize model size (Example)

- Can identify $a=d$, $c=e$
1. Minimize model size (Example)

- Also has model of size 3:

\[ S = \{a, b, c\} \]
1. Minimize model size

- UF+cardinality constraints [CAV 2013]

\[
|S| \leq 1 \quad \neg |S| \leq 1
\]

Search for models where \(|S|=1\)

If none exist, search for models where \(|S|=2\)

\[
|S| \leq 2 \quad \neg |S| \leq 2
\]

\[
|S| \leq 3 \quad \neg |S| \leq 3
\]

e tc.
2. Model-Based Quantifier Instantiation

• Construct *candidate model* $M$

\[ \forall x y : S . Q(x,y) \]

**Diagram:**
- Ground Theory Solvers
- $G$
- $M$
- FMF Module
- $S = \{a, b, c, d, e\}$
- $\{Q \rightarrow Q^M, ...\}$
2. Model-Based Quantifier Instantiation

- Using model, evaluate $Q^M(x, y)$ [CADE 2013]
2. Model-Based Quantifier Instantiation

- If all $T$ instances, $M$ is a model, answer “SAT”
2. Model-Based Quantifier Instantiation

- If all T instances, M is a model, answer “SAT”
- Else, add (subset of) F instances to G

\[ \forall x y : S . Q(x, y) \]

\[ \begin{align*}
G & \quad \begin{cases}
Q(b, a) \\
Q(e, e)
\end{cases} \\
Q^M & \quad \begin{cases}
a \\
b \\
c \\
d \\
e
\end{cases} \quad \begin{cases}
T \\
F \\
F \\
F \end{cases}
\end{align*} \]

\[ S = \{ a, b, c, d, e \} \]
\[ \{ Q \rightarrow Q^M, \ldots \} \]
Application: DVF at Intel

- CVC4+FMF used as a backend for:
  - DVF tool at Intel Research
    - Verifying software/hardware architectures

⇒ Required:
- Scalable methods for finding counterexamples (FMF)
- Support for many theories (arith, BV, arrays, …)
Variants of Approach

- CVC4 has techniques for handling:
  - Bounded Integer Quantification
  - Strings of bounded length
  - Syntax-Guided Synthesis (in progress)

⇒ Each follow similar pattern
Variant #1: Bounded Integers

\[ \forall x: \text{Int}. \ 0 \leq x < t \Rightarrow P(x) \]

- Search for models where \( t = 0 \)
- If none exist, search for models where \( t = 1 \)
- etc.
Variant #2 : Bounded Length Strings

• Given input $F[s_1,\ldots,s_n]$ for strings $s_1\ldots s_n$:

\[ \sum_{i=1}^{n} |s_i| \leq 0 \quad \neg \sum_{i=1}^{n} |s_i| \leq 0 \]

- Search for models where sum of lengths=0
- Search for models where sum of lengths=1
- Search for models where sum of lengths=2
- etc.

\[ \sum_{i=1}^{n} |s_i| \leq 1 \quad \neg \sum_{i=1}^{n} |s_i| \leq 1 \]

\[ \sum_{i=1}^{n} |s_i| \leq 2 \quad \neg \sum_{i=1}^{n} |s_i| \leq 2 \]
Variant #3: Syntax-Guided Synthesis

\[ (\neg) \exists f: \text{Program.} \forall i. P(f, i) \]

\[
\begin{align*}
\text{Search for programs of size 0} & : \text{size}(f) \leq 0 \\
\text{Search for programs of size 1} & : \text{size}(f) \leq 1 \\
\text{Search for programs of size 2} & : \text{size}(f) \leq 2 \\
\vdots \\
\Rightarrow \text{In development} 
\end{align*}
\]
Wrap-up
Use in Research


Jovanović, Barrett, and de Moura. The design and implementation of the model constructing satisfiability calculus. *FMCAD*, 2013.

King and Barrett. Exploring and categorizing error spaces using BMC and SMT. *Satisfiability Modulo Theories (SMT)*, 2011.


King, Barrett, and Tinelli. Leveraging linear and mixed integer programming for SMT. *FMCAD*, 2014.


Reynolds, Tinelli, Goel, Krstić, Deters, and Barrett. Quantifier instantiation techniques for finite model finding in SMT. *CADE*, 2013.

This week at FMCAD

King, Barrett, and Tinelli. Leveraging Linear and Mixed Integer Programming for SMT.
Thursday 1:30pm (just after lunch).

Reynolds, Tinelli, and de Moura. Finding Conflicting Instances of Quantified Formulas in SMT.
Thursday 3:45pm (final session of the day).
We use CVC4, too

cascade  http://cascade.cims.nyu.edu/

Verification platform for C programs. Supports a range of data representation and memory models for precise or abstract analysis.

\{ KIND \}  http://clc.cs.uiowa.edu/Kind/

Practicalities of using CVC4

• Open-source (BSD-licensed)

• Accepted into MacPorts

• Already in Fedora

• Debian package built (forthcoming in Debian)
Easy to use

• Straightforward API
  – follows SMT-LIB command format
  – C++, Java, ...

• Compatibility support for CVC3’s APIs
  – C, C++, and Java
Well-supported

- GitHub, Travis-CI
- Users’ mailing list
- StackOverflow
- Bugzilla
- Nightly development builds

...under active development
The CVC4 Team

Clark Barrett (NYU)
Cesare Tinelli (U Iowa)
Kshitij Bansal (NYU)
François Bobot (CEA)
Chris Conway (Google)
Morgan Deters (NYU)
Liana Hadarean (NYU)
Dejan Jovanović (SRI)
Tim King (Verimag)
Tianyi Liang (U Iowa)
Andrew Reynolds (EPFL)
Calling all users

- Always looking for users
- Always looking for collaborations

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