A Versatile and Industrial-Strength SMT Solver

Haniel Barbosa,1 Clark Barrett,2 Martin Brain,3 Gereon Kremer,2 Hanna Lachnitt,2 Makai Mann,2 Abdalrhman Mohamed,4 Mudathir Mohamed,4 Aina Niemetz,2 Andres Nötzli,2 Alex Ozdemir,2 Mathias Preiner,2 Andrew Reynolds,4 Ying Sheng,2 Cesare Tinelli,4 Yoni Zohar5

1 Universidade Federal de Minas Gerais, 2 Stanford University, 3 City, University of London, 4 The University of Iowa, 5 Bar-Ilan University
Introduction

A Versatile and Industrial-Strength SMT Solver

- Support for *all* standard SMT-LIB and additional non-standard theories
- Beyond SMT solving
  - Proof generation
  - Syntax-Guided Synthesis (SyGuS)
  - Interpolation
  - Abduction

- Extensively used in industry
- Comprehensive, stable API and documentation
- Permissive license
Introduction

A Versatile and Industrial-Strength SMT Solver

- Support for *all* standard SMT-LIB and additional non-standard theories
- Beyond SMT solving
  - Proof generation
  - Syntax-Guided Synthesis (SyGuS)
  - Interpolation
  - Abduction

- Extensively used in industry
- Comprehensive, stable API and documentation
- Permissive license
A Versatile and Industrial-Strength SMT Solver

- Support for all standard SMT-LIB and additional non-standard theories
- Beyond SMT solving
  - Proof generation
  - Syntax-Guided Synthesis (SyGuS)
  - Interpolation
  - Abduction

- Extensively used in industry
- Comprehensive, stable API and documentation
- Permissive license
Agenda

- Architecture
- Feature Highlights
  - New API
  - Proofs
  - SyGuS
    - Interpolation and Abduction
- Evaluation
Architecture
### Workflow

<table>
<thead>
<tr>
<th>Java API</th>
<th>Python API</th>
<th>C++ API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abduction Solver</td>
<td>Interpolation Solver</td>
<td>cvc5 Core</td>
</tr>
<tr>
<td>SyGuS Solver</td>
<td>Quantifier-Elimination Solver</td>
<td></td>
</tr>
<tr>
<td>SMT Solver</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preprocessor</td>
<td>Propositional Engine</td>
<td>Theory Engine</td>
</tr>
<tr>
<td>Preprocessing Passes</td>
<td>MiniSat</td>
<td>Combination Engine</td>
</tr>
<tr>
<td>Rewriter</td>
<td>Decision Engine</td>
<td>Theory Solvers</td>
</tr>
<tr>
<td></td>
<td>Clausifier</td>
<td>Quantifiers Module</td>
</tr>
</tbody>
</table>

**CLI**

- Parser

**C++ API**

- LFSC Converter
- Alethe Converter
- Lean Converter

**Proof Module**

- Node Manager
- Skolem Manager

**Context-Dependent Data Structures**
<table>
<thead>
<tr>
<th>CLI</th>
<th>Java API</th>
<th>Python API</th>
<th>C++ API</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Python API</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++ API</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **cvc5 Core**
  - Abduction Solver
  - Interpolation Solver
  - SyGuS Solver
  - Quantifier-Elimination Solver

- **SMT Solver**
  - Preprocessor
    - Preprocessing
    - Passes
  - Propositional Engine
    - MiniSat
    - Decision Engine
  - Clausifier
  - Theory Engine
    - Combination Engine
    - Theory Solvers
    - Quantifiers Module

- **Proof Module**
  - LFSC Converter
  - Alethe Converter
  - Lean Converter

- **Node Manager**
  - Skolem Manager

- **Context-Dependent Data Structures**

- **Workflow**
  - CLI
  - Java API
  - Python API
  - C++ API

- **Context-Dependent Data Structures**
## Workflow

<table>
<thead>
<tr>
<th>Java API</th>
<th>Python API</th>
<th>CLI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abduction Solver</td>
<td>Interpolation Solver</td>
<td>Parser</td>
</tr>
<tr>
<td>SyGuS Solver</td>
<td>Quantifier-Elimination Solver</td>
<td></td>
</tr>
<tr>
<td>Preprocessor</td>
<td>Propositional Engine</td>
<td></td>
</tr>
<tr>
<td>Preprocessing Passes</td>
<td>MiniSat</td>
<td>LFSC Converter</td>
</tr>
<tr>
<td>Rewriter</td>
<td>Decision Engine</td>
<td>Alethe Converter</td>
</tr>
<tr>
<td>Propositional Engine</td>
<td>Clausifier</td>
<td>Lean Converter</td>
</tr>
<tr>
<td>Theory Engine</td>
<td>Clausifier</td>
<td></td>
</tr>
<tr>
<td>Theory Solvers</td>
<td>Theory Solvers</td>
<td></td>
</tr>
<tr>
<td>Quantifiers Module</td>
<td>Quantifiers Module</td>
<td></td>
</tr>
<tr>
<td>Combination Engine</td>
<td>Combination Engine</td>
<td></td>
</tr>
<tr>
<td>Context-Dependent Data Structures</td>
<td>Context-Dependent Data Structures</td>
<td></td>
</tr>
<tr>
<td>Node Manager</td>
<td>Node Manager</td>
<td></td>
</tr>
<tr>
<td>Skolem Manager</td>
<td>Skolem Manager</td>
<td></td>
</tr>
<tr>
<td>Proof Module</td>
<td>Proof Module</td>
<td></td>
</tr>
</tbody>
</table>
Workflow

<table>
<thead>
<tr>
<th>CLI</th>
<th>Java API</th>
<th>Python API</th>
<th>C++ API</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Abduction Solver</td>
<td>Interpolation Solver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SyGuS Solver</td>
<td>Quantifier-Elimination Solver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMT Solver</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preprocessor</td>
<td>Propositional Engine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preprocessing Passes</td>
<td>MiniSat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rewriter</td>
<td>Decision Engine</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Clausifier</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

cvc5 Core

<table>
<thead>
<tr>
<th>LFSC Converter</th>
<th>Alethe Converter</th>
<th>Lean Converter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proof Module</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node Manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skolem Manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context-Dependent Data Structures</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Theory Solvers

- Linear arithmetic [Kin14, KBD13, KBT14]
- Non-linear arithmetic [RTJB17]
- Arrays [JB13]
- Bit-vectors
- Datatypes [BST07, RB15, RVB\(^+\)18]
- Floating-point arithmetic [BSS19]
- Sets and relations [BBRT17, MRTB17]
- Separation logic [RISK16]
- Strings and sequences [LRT\(^+\)14, RWB\(^+\)17, LTR\(^+\)15, RNBT19, RNBT20]
- Uninterpreted functions (with support for finite cardinality constraints) [RTGK13]
- Quantifiers [RTdM14, BFR17, RTG\(^+\)13, RBF18, RKK17, NPR\(^+\)21a, NPR\(^+\)21b, RK15, RBCT16, RDK\(^+\)15]
Feature Highlights
Feature Highlights: API

- New C++ API
  - Lean, comprehensive, feature-complete
  - Parser module uses the same API
  - Comprehensive documentation

- Python bindings: 2 variants
  - Base bindings: Complete Cython-based bindings for the API
  - Pythonic bindings: High-level bindings, drop-in replacement for Z3py

- Java bindings
  - Complete JNI-based bindings for the API

Demo

Solving a simple problem using the Pythonic API
Feature Highlights: API

- New C++ API
  - Lean, comprehensive, feature-complete
  - Parser module uses the same API
  - Comprehensive documentation

- Python bindings: 2 variants
  - Base bindings: Complete Cython-based bindings for the API
  - Pythonic bindings: High-level bindings, drop-in replacement for Z3py

- Java bindings
  - Complete JNI-based bindings for the API

Demo

Solving a simple problem using the Pythonic API
Feature Highlights: API

- New C++ API
  - Lean, comprehensive, feature-complete
  - Parser module uses the same API
  - Comprehensive documentation

- Python bindings: 2 variants
  - Base bindings: Complete Cython-based bindings for the API
  - Pythonic bindings: High-level bindings, drop-in replacement for Z3py

- Java bindings
  - Complete JNI-based bindings for the API

Demo

Solving a simple problem using the Pythonic API
Feature Highlights: API

- New C++ API
  - Lean, comprehensive, feature-complete
  - Parser module uses the same API
  - Comprehensive documentation
- Python bindings: 2 variants
  - Base bindings: Complete Cython-based bindings for the API
  - Pythonic bindings: High-level bindings, drop-in replacement for Z3py
- Java bindings
  - Complete JNI-based bindings for the API

Demo

Solving a simple problem using the Pythonic API
Feature Highlights: Proofs

- New module for producing proofs for unsatisfiable inputs
  - Enables independent checking of answers
  - Automating proofs in interactive theorem provers
- Goals
  - Low overhead
  - Detailed, efficiently checkable proofs
  - Support all performance-critical components
  - Output in different proof formats

Demo
Generating a proof for a simple problem.
Feature Highlights: Proofs

- New module for producing proofs for unsatisfiable inputs
  - Enables independent checking of answers
  - Automating proofs in interactive theorem provers
- Goals
  - Low overhead
  - Detailed, efficiently checkable proofs
  - Support all performance-critical components
  - Output in different proof formats

<table>
<thead>
<tr>
<th>LFSC Converter</th>
<th>Alethe Converter</th>
<th>Lean Converter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proof Module</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Demo

Generating a proof for a simple problem.
There exists a function \( f \) for which property \( P \) holds for all \( x \) in some theory \( T \).

**Specification**

\[ \exists f. \forall x. P(f, x) \]

**Syntax**

\[
A := A + A \mid -A \mid x \mid y \mid 0 \mid 1 \mid \text{ite}(B, A, A)
\]

\[
B := B \land B \mid \neg B \mid A = A \mid A \geq A \mid \perp
\]

**Demo**

Flash Fill-style synthesis.
Feature Highlights: Syntax-Guided Synthesis (SyGuS)

**Specification**

$$\exists f. \forall x. P(f, x)$$

There exists a function $f$ for which property $P$ holds for all $x$ in some theory $T$.

**Syntax**

$$A := A + A \mid \neg A \mid x \mid y \mid 0 \mid 1 \mid \text{ite}(B, A, A)$$

$$B := B \land B \mid \neg B \mid A = A \mid A \geq A \mid \perp$$

**Demo**

Flash Fill-style synthesis.
Feature Highlights: Syntax-Guided Synthesis (SyGuS)

**Specification**

$$\exists f. \forall x. P(f, x)$$

There exists a function \( f \) for which property \( P \) holds for all \( x \) in some theory \( T \).

**Syntax**

\[
A := A + A \mid -A \mid x \mid y \mid 0 \mid 1 \mid \text{ite}(B, A, A)
\]

\[
B := B \land B \mid \neg B \mid A = A \mid A \geq A \mid \perp
\]

**Demo**

Flash Fill-style synthesis.
Feature Highlights: Interpolation/Abduction

Interpolation

Find a term $C$ such that $A \models C$ and $C \models B$. Free symbols in $C$ are from set of shared symbols between $A$ and $B$.

Abduction

Find a term $C$ such that $A \land C$ is satisfiable and $A \land C \models B$.

Demo

Fixing a floating-point rewrite using abduction.
**Interpolation**

Find a term $C$ such that $A \models C$ and $C \models B$. Free symbols in $C$ are from set of shared symbols between $A$ and $B$.

**Abduction**

Find a term $C$ such that $A \land C$ is satisfiable and $A \land C \models B$.

**Demo**

Fixing a floating-point rewrite using abduction.
Interpolation

Find a term \( C \) such that \( A \models C \) and \( C \models B \). Free symbols in \( C \) are from set of shared symbols between \( A \) and \( B \).

Abduction

Find a term \( C \) such that \( A \wedge C \) is satisfiable and \( A \wedge C \models B \).

Demo

Fixing a floating-point rewrite using abduction.
Evaluation
Evaluation: Setup

- Comparison with CVC4 1.8 and Z3
- Benchmark set: 379,750 non-incremental SMT-LIB benchmarks
  - All logics (quantified and quantifier-free)
  - Excluding 1,173 misclassified benchmarks
- Timeout: 1,200 seconds (like SMT-COMP)
Evaluation: Results

![Graph showing runtime vs. solved instances for cvc5, CVC4, and z3.](chart.png)
Future Work

- Optimization solver
  - Computing satisfying assignments that optimize objectives
- New theories/extensions of theories
  - Support for higher-order map/fold combinators
- Parallel SMT solving
  - Integrated support for running multiple configurations in parallel/sequence
- Performance tuning
  - Complete replacement of ANTLR parser
  - Lifting local search approach for bit-vectors to floating-point arithmetic
Future Work

- Optimization solver
  - Computing satisfying assignments that optimize objectives
- New theories/extensions of theories
  - Support for higher-order map/fold combinators
- Parallel SMT solving
  - Integrated support for running multiple configurations in parallel/sequence
- Performance tuning
  - Complete replacement of ANTLR parser
  - Lifting local search approach for bit-vectors to floating-point arithmetic
Future Work

- Optimization solver
  - Computing satisfying assignments that optimize objectives
- New theories/extensions of theories
  - Support for higher-order map/fold combinators
- Parallel SMT solving
  - Integrated support for running multiple configurations in parallel/sequence
- Performance tuning
  - Complete replacement of ANTLR parser
  - Lifting local search approach for bit-vectors to floating-point arithmetic
Future Work

- Optimization solver
  - Computing satisfying assignments that optimize objectives
- New theories/extensions of theories
  - Support for higher-order map/fold combinators
- Parallel SMT solving
  - Integrated support for running multiple configurations in parallel/sequence
- Performance tuning
  - Complete replacement of ANTLR parser
  - Lifting local search approach for bit-vectors to floating-point arithmetic
About cvc5

Cvc5 is an efficient open source automated theorem prover for Satisfiability Modulo Theories (SMT) problems. It can be used to prove the unsatisfiability (or, dually, the validity) of first-order formulas with respect to combinations of a variety of arithmetic theories. It further provides a theory-based Satisfiability Modulo Theories (SMT) engine for optimization functions with respect to background theories and their combinations.

Cvc5 is the successor of CVC4 and is intended to be an open and extensible SMT engine. It can be used as a core component for a theory, which can be a certificate checker (e.g., as a library with essentially no limit on its use for research or commercial purposes). To contribute to Cvc5, please refer to our contribution guidelines.

Cvc5 is a joint project led by Stanford University and the University of Iowa.

Technical Support

For bug reports, please use the GitHub issue tracker.

If you have a question, a feature request, or if you would like to contribute in some way, please use the discussion feature on the Cvc5 GitHub repository.

Guidelines for Fuzzing cvc5

The development team of cvc5 is committed to ensuring that its core upstream model (without experimental options) is extremely robust. At the same time, our team is small and we know that our priority, including providing user bugs over feature bugs.

When applying fuzzing techniques to cvc5, we ask not to follow those guidelines.

https://cvc5.github.io/

https://github.com/cvc5/cvc5/
<table>
<thead>
<tr>
<th>Division</th>
<th>cvc5</th>
<th>CVC4</th>
<th>Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arith (7104)</td>
<td>6593</td>
<td>6498</td>
<td>6844</td>
</tr>
<tr>
<td>Bitvec (6045)</td>
<td>5741</td>
<td>5690</td>
<td>5664</td>
</tr>
<tr>
<td>Equality (12159)</td>
<td>6677</td>
<td>6681</td>
<td>4688</td>
</tr>
<tr>
<td>Equality+LinearArith (55948)</td>
<td>49395</td>
<td>48487</td>
<td>49503</td>
</tr>
<tr>
<td>Equality+MachineArith (4712)</td>
<td>2065</td>
<td>1832</td>
<td>1804</td>
</tr>
<tr>
<td>Equality+NonLinearArith (17260)</td>
<td>11088</td>
<td>10906</td>
<td>9341</td>
</tr>
<tr>
<td>FPArith (3170)</td>
<td>2625</td>
<td>2113</td>
<td>2593</td>
</tr>
<tr>
<td>QF Bitvec (42450)</td>
<td>41569</td>
<td>41448</td>
<td>40582</td>
</tr>
<tr>
<td>QF Equality (16254)</td>
<td>16124</td>
<td>16121</td>
<td>16115</td>
</tr>
<tr>
<td>QF Equality+Bitvec (16518)</td>
<td>16274</td>
<td>16333</td>
<td>16318</td>
</tr>
<tr>
<td>QF Equality+LinearArith (3924)</td>
<td>3778</td>
<td>3782</td>
<td>3822</td>
</tr>
<tr>
<td>QF Equality+NonLinearArith (673)</td>
<td>598</td>
<td>610</td>
<td>616</td>
</tr>
<tr>
<td>QF FPArith (76084)</td>
<td>75998</td>
<td>75965</td>
<td>75816</td>
</tr>
<tr>
<td>QF LinearIntArith (9765)</td>
<td>8619</td>
<td>8778</td>
<td>8464</td>
</tr>
<tr>
<td>QF LinearRealArith (2008)</td>
<td>1849</td>
<td>1881</td>
<td>1864</td>
</tr>
<tr>
<td>QF NonLinearIntArith (24261)</td>
<td>17525</td>
<td>16860</td>
<td>18357</td>
</tr>
<tr>
<td>QF NonLinearRealArith (11552)</td>
<td>10889</td>
<td>9207</td>
<td>10354</td>
</tr>
<tr>
<td>QF Strings (69863)</td>
<td>69231</td>
<td>69367</td>
<td>68074</td>
</tr>
<tr>
<td>Total (379750)</td>
<td>346638</td>
<td>342559</td>
<td>340819</td>
</tr>
</tbody>
</table>
Kshitij Bansal, Clark W. Barrett, Andrew Reynolds, and Cesare Tinelli.  
**A new decision procedure for finite sets and cardinality constraints in SMT.**  

Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds.  
**Congruence closure with free variables.**  

Martin Brain, Florian Schanda, and Youcheng Sun.  
**Building better bit-blasting for floating-point problems.**  

Clark Barrett, Igor Shikanian, and Cesare Tinelli.
An abstract decision procedure for a theory of inductive data types.  

Dejan Jovanovic and Clark W. Barrett.  
**Being careful about theory combination.**  

Tim King, Clark W. Barrett, and Bruno Dutertre.  
**Simplex with sum of infeasibilities for SMT.**  

Tim King, Clark W. Barrett, and Cesare Tinelli.  
**Leveraging linear and mixed integer programming for SMT.**  

Tim King.
Effective Algorithms for the Satisfiability of Quantifier-Free Formulas Over Linear Real and Integer Arithmetic.


Baoluo Meng, Andrew Reynolds, Cesare Tinelli, and Clark W. Barrett. Relational constraint solving in SMT.
Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli.

**On solving quantified bit-vector constraints using invertibility conditions.**

Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli.

**Syntax-guided quantifier instantiation.**
Andrew Reynolds and Jasmin Christian Blanchette.  
**A decision procedure for (co)datatypes in SMT solvers.**  

Andrew Reynolds, Jasmin Christian Blanchette, Simon Cruanes, and Cesare Tinelli.  
**Model finding for recursive functions in SMT.**  

Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine.
Revisiting enumerative instantiation.

Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark W. Barrett.

**Counterexample-guided quantifier instantiation for synthesis in SMT.**

Andrew Reynolds, Radu Iosif, Cristina Serban, and Tim King.

**A decision procedure for separation logic in SMT.**
Andrew Reynolds and Viktor Kuncak.
**Induction for SMT solvers.**

Andrew Reynolds, Tim King, and Viktor Kuncak.
**Solving quantified linear arithmetic by counterexample-guided instantiation.**

Andrew Reynolds, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli.
**High-level abstractions for simplifying extended string constraints in SMT.**
Andrew Reynolds, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. 
**A decision procedure for string to code point conversion.**  

Andrew Reynolds, Cesare Tinelli, and Leonardo Mendonça de Moura. 
**Finding conflicting instances of quantified formulas in SMT.**  

Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstic, Morgan Deters, and Clark W. Barrett. 
**Quantifier instantiation techniques for finite model finding in SMT.**  
Andrew Reynolds, Cesare Tinelli, Amit Goel, and Sava Krstic. **Finite model finding in SMT.**

Andrew Reynolds, Cesare Tinelli, Dejan Jovanovic, and Clark W. Barrett. **Designing theory solvers with extensions.**

**Datatypes with shared selectors.**

Andrew Reynolds, Maverick Woo, Clark W. Barrett, David Brumley, Tianyi Liang, and Cesare Tinelli.

**Scaling up DPLL(T) string solvers using context-dependent simplification.**