How to use cvc5 Effectively

Andrew Reynolds
September 28, 2023
Overview

• cvc5: a state-of-the-art SMT solver for verification
  • Supports many techniques for quantified formulas
  • Combined with a wide array of theory solvers

• Interfaces for when things go *right*

• Interfaces for when things go *wrong*
Architecture of cvc5

*.smt2, ...

Preprocessor

SAT Solver

cvc5

Theory Solver(s)

UNSAT

SAT

UNKNOWN
Architecture of cvc5

Preprocessor

SAT Solver

Satisfying Assignments

CDCL(T)

Conflicts, Lemmas

Arithmetic

Datatypes

Strings

Arrays

Bit-vectors

Quantifier Instantiation

T-Combination
Many verification applications rely on *quantifier instantiation*
- cvc5 supports many variants
Landscape of Quantifier Strategies in cvc5

Lightweight

General purpose

[Detlefs et al 2003, deMoura et al 2007, Ge et al 2007]

Heavyweight

Domain Specific

E-Matching
Landscape of Quantifier Strategies in cvc5

Lightweight

General purpose

Domain Specific

Heavyweight

E-Matching

[Detlefs et al 2003, deMoura et al 2007, Ge et al 2007]

Model-Based

[Ge et al 2009]
Landscape of Quantifier Strategies in cvc5

- **General purpose**
  - Lightweight
  - Heavyweight
  - Model-Based: [Ge et al 2009]
  - Finite Model Finding: [Reynolds et al 2013]

- **Domain Specific**
Landscape of Quantifier Strategies in cvc5

- General purpose
  - Lightweight
    - Conflict-Based
      - [Reynolds et al 2014, Barbosa et al 2017]
    - E-Matching
      - [Detlefs et al 2003, deMoura et al 2007, Ge et al 2007]
  - Heavyweight
    - Model-Based
      - [Ge et al 2009]
    - Finite Model Finding
      - [Reynolds et al 2013]
- Domain Specific
Landscape of Quantifier Strategies in cvc5

- **Lightweight**
  - Conflict-Based
    - [Reynolds et al 2014, Barbosa et al 2017]
  - E-Matching
    - [Detlefs et al 2003, deMoura et al 2007, Ge et al 2007]

- **Heavyweight**
  - Model-Based
    - [Ge et al 2009]
  - Finite Model Finding
    - [Reynolds et al 2013]

- **Domain Specific**
  - CEX-Guided
    - [Reynolds et al 2015] (LIA)
    - [Niemetz et al 2018] (BV)
Landscape of Quantifier Strategies in cvc5

General purpose

Lightweight

Conflict-Based
[Reynolds et al 2014, Barbosa et al 2017]

E-Matching
[Detlefs et al 2003, deMoura et al 2007, Ge et al 2007]

Enumerative
[Reynolds et al 2017, Janota et al 2021]

Heavyweight

Model-Based
[Ge et al 2009]

Finite Model Finding
[Reynolds et al 2013]

Domain Specific

CEX-Guided
[Reynolds et al 2015] (LIA)
[Niemetz et al 2018] (BV)
Landscape of Quantifier Strategies in cvc5

- **General purpose**
  - **Conflict-Based**
    - [Reynolds et al 2014, Barbosa et al 2017]
  - **E-Matching**
    - [Detlefs et al 2003, deMoura et al 2007, Ge et al 2007]
  - **Enumerative**
    - [Reynolds et al 2017, Janota et al 2021]

- **Lightweight**
  - **Model-Based**
    - [Ge et al 2009]
  - **Finite Model Finding**
    - [Reynolds et al 2013]

- **Heavyweight**
  - **Syntax-Guided**
    - [Reynolds et al 2015] (LIA)
    - [Niemetz et al 2018] (BV)
  - **CEX-Guided**
    - [Reynolds et al 2015] (LIA)
    - [Niemetz et al 2018] (BV)
Landscape of Quantifier Strategies in cvc5

- **Lightweight**
  - Conflict-Based
  - E-Matching
  - Enumerative

- **Heavyweight**
  - Model-Based
  - Finite Model Finding

- **General purpose**
  - Enabled by default

- **Domain Specific**
  - CEX-Guided
    - Enabled in specific logics (LIA, BV)
  - Syntax-Guided
    - Optionally enabled if the above do not suffice
Theory Solvers supported in cvc5

• Support for many theories
  • Arithmetic, Bit-vectors, Arrays, Datatypes, Floating-Points, Strings
  • **Extended:** Sets, Sequences, Multisets, Finite Fields

• The use of theories can avoid (some) use of quantified formulas, see:
  • (Co)datatypes [Reynolds et al CADE 2015]
  • Relations [Meng et al CADE 2017]
  • Sequences [Shing et al IJCAR 2022]

⇒ If you have a new problem domain, we can add custom support for it
**cvc5**: Interfaces for When Things go *Right*

i.e. when the solver says “sat” or “unsat”

- **get-model**
  - *What is the counterexample to the theorem?*
    - Can be refined to only include relevant assignments `get-model-core`

- **get-unsat-core**
  - *What are the necessary assertions for proving this theorem?*
    - Can be minimized via option `--minimal-unsat-core`
    - Finer-grained versions `get-instantiations`

- **get-proof**
  - *What is the precise reasoning for proving the theorem?*


**cvc5: Interfaces for When Things go Wrong**

i.e. when the solver says “unknown” or times out

- **get-model**
  - *What is a candidate counterexample to this theorem?*
    - Available even when the solver times out or gives up

- **get-difficulty**
  - *Which assertions were the reason why this problem was hard?*

- **get-timeout-core**
  - *Which assertions suffice to make the solver time out again?*

- **get-learned-lits**
  - *What immediate formulas were learned during solving?*

- External tools for delta-debugging e.g. ddSmt [Kremer et al 2020]
Difficulty Estimation

• When cvc5 can’t solve an input, can we estimate why it was difficult?

When cvc5 can’t solve an input, the difficulty estimation process involves:

1. Input: $F_1 \land \ldots \land F_n$
2. cvc5
3. get-difficulty

The difficulty map is:

- $F_1 \rightarrow d_1$
- $\ldots$
- $F_n \rightarrow d_n$

The larger $d_i$, the harder $F_i$ was to solve for.

Difficulty map:

- UNSAT
- SAT
- UNKNOWN
Difficulty Estimation

• Given input $F_1 \land ... \land F_n$
  • Model-based:
    • When a candidate model $M$ is constructed
      • Increment difficulty measure for each $F_j$ that $M$ does not satisfy

• Conflict-based:
  • When a conflict clause $(l_1 \lor ... \lor l_n)$ is raised
    • For each literal $l_i$, increment difficulty measure for the $F_j$ s.t. $F_j \models \neg l_i$
Timeout Cores

- Given a timeout, can we construct a smaller problem cvc5 also cannot solve?

\[ F_1 \land ... \land F_n \]

\[ \text{cvc5} \]

\[ \text{get-timeout-core} \]

\[ F_{i1} \land F_{i2} \land F_{i3} \]

\[ \text{cvc5} \]

\[ \text{TIMEOUT} \]
Timeout Cores

• To compute a timeout core for $F = \{ F_1, \ldots, F_n \}$:
  • Maintain an (initially empty) set of models $M$
  • Maintain an (initially empty) set of formulas $C \subseteq F$ such that
    • Each model in $M$ does not satisfy at least one formula in $C$
  • Repeat:
    • If $C$ is unsat
      • Report that $F$ is unsat, $C$ is an unsat core of $F$
    • If $C$ makes the solver timeout
      • Report that $C$ is a timeout core of $F$
    • If $C$ is sat with model $m$
      • If $m$ satisfies $F$
        • Report that $F$ is sat
      • Else, add $m$ to $M$, add some $F_i$ to $C$ s.t. $m$ does not satisfy $F_i$, refine $C$
• SMT solver cvc5 is
  • Efficient tool widely used in applications
  • Handles many problem domains
  • Many interfaces for when things go right (or wrong)

• Questions?