Fast and Trustworthy
SMT Solving for String Constraints

Andrew Reynolds
Aug 24, 2022
Overview

• Satisfiability Modulo Theories (SMT) solvers widely used tools
  ⇒ the SMT solver cvc5

• cvc5 for strings and regular expressions
  • *Fast*: new advances in SMT string solving
  • *Trustworthy*: producing externally checkable proofs

• Future Directions
Satisfiability Modulo Theories (SMT) Solvers

- Software Verification Tools
- Interactive Proof Assistants
- Symbolic Execution Engines
- Synthesis Tools, Planners
- Security Analyzers

Verification Conditions → Conjectures → Path Constraints → Specifications → Queries

[CVC5] SAT → UNSAT

⇒ SMT solvers are fully automated reasoners, widely used in applications
cvc5 is latest SMT solver in CVC line of tools
- Open source, builds on code base of CVC4
- 1.0 launched in April 2022

⇒ Best tool paper, ETAPS 2022
cvc5: A Versatile and Industrial-Strength SMT Solver*

Haniel Barbosa\textsuperscript{3,\textcircled{1}}, Clark Barrett\textsuperscript{1}, Martin Brain\textsuperscript{4,\textcircled{2}}, Gereon Kremer\textsuperscript{1,\textcircled{3}}, Hanna Lachnitt\textsuperscript{1}, Makai Mann\textsuperscript{1,\textcircled{2}}, Abdalrhman Mohamed\textsuperscript{2,\textcircled{3}}, Mudathir Mohamed\textsuperscript{2,\textcircled{3}}, Aina Niemetz\textsuperscript{1,\textcircled{2}}, Andres Nötzli\textsuperscript{1,\textcircled{2}}, Alex Ozdemir\textsuperscript{1,\textcircled{2}}, Mathias Preiner\textsuperscript{1,\textcircled{2}}, Andrew Reynolds\textsuperscript{2,\textcircled{3}}, Ying Sheng\textsuperscript{1,\textcircled{2}}, Cesare Tinelli\textsuperscript{2,\textcircled{3}}, and Yoni Zohar\textsuperscript{1,\textcircled{5}}

1 Stanford University, Stanford, USA  
2 The University of Iowa, Iowa City, USA  
3 Universidade Federal de Minas Gerais, Belo Horizonte, Brazil  
4 City, University of London, London, UK  
5 Bar-Ilan University, Tel Aviv, Israel
cvc5: A **Versatile** SMT Solver

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

- Many features:
  - `get-model`, `get-unsat-core`, `get-proof`
  - **Extended:** syntax-guided synthesis, `get-interpolant`, `get-abduct`, `get-quant-elim`

⇒ If you have a new problem domain, we’d like to support it!
cvc5: SMT and beyond

Problem

Higher-Order ∀ Satisfiability
Function Synthesis
First-Order ∀ Satisfiability
∀-Free Satisfiability

Theories

Boolean, Bit-vectors, FP
Equality with UF, Arrays
Datatypes, Sets, Bags
Linear Arithmetic
Non-linear Arithmetic
Strings+

SAT QF_BV EPR QBF BV

Decidable My Research This Talk

Reals Ints
Reals Ints
Reals Ints
cvc5: A Versatile and Industrial-Strength SMT Solver

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

1 Stanford University, Stanford, USA
2 The University of Iowa, Iowa City, USA
3 Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
4 City, University of London, London, UK
5 Bar-Ilan University, Tel Aviv, Israel
cvc5: An Industrial-Strength SMT Solver

- Efficient solving algorithms
    * shared with predecessor CVC4

- High coding standards
  - Streamlined API, high code coverage, code reviews

- Extensively tested
  - 3000+ hand-crafted regressions and counting
  - Fuzzed internally (Murxla [Niemetz et al CAV22]) and by external groups

- New: Produces externally checkable proofs
cvc5: state-of-the-art SMT solver *for Strings*

- **Fast** solving techniques
  - CDCL(T)
  - Core calculus for strings and length constraints
  - Extensions to extended functions and regular expressions

- **Trustworthy** results
  - Proof production for the full theory of strings
Designing a **Fast** String Solver
Architecture of cvc5

- Preprocessor
- SAT Solver
- Theory Solver(s)
- Satisfying Assignments
- CDCL(T)
- Conflicts, Lemmas

*.smt2, ...

UNSAT → SAT

Nieuwenhius et al
JACM 2006
Architecture of cvc5

- Centralized methods (Nelson-Oppen, polite) for combining theories
Focus of this talk: solver for the *theory of strings and regular expressions*
Many applications require extended string functions and RegEx memberships:

- $\text{ctn}(x, "a"), \text{to_lower}(x)="abc", x\in\text{range}("A","Z")$
SMT Solvers for Strings: Timeline

2009
- CVC4 project begins
- First release CVC4 1.0
- CVC papers on strings
  - [Liang et al CAV 14]
- Other SMT String Solvers
  - Z3Str
  - S3

2012
- CVC4 support for strings + RE

2014
- Support for Extended String Functions

2017
- Support For Proofs
- [Reynolds et al CAV 17]
- [Reynolds et al CAV 19]
- [Reynolds et al FMCAD 20]
- [Reynolds et al IJCAR 20]
- [Noetzli et al CAV 22]
- [Noetzli et al FMCAD 22]
- Z3Str2
- Z3Seq
- Trau
- Z3Str3
- Z3Str4
- Norn
- OSTRICHH

2020
- cvc5 1.0 release
A Theory Solver for Strings [Liang et al, CAV 14]

- Designed a string solver for concat+length that is:
  - **Refutation and model sound** (“unsat” and “sat” can be trusted)
  - **Not terminating** in general
  - **Efficient** in practice

\[
\begin{align*}
    x &= \text{"abc"} \cdot y \\
    |y| &= 4 \\
    x &= \text{"b"} \cdot z
\end{align*}
\]

\[x \neq \text{"abc"} \cdot y \lor x \neq \text{"b"} \cdot z\]  

Conflict Clause
Extended Theory of Strings [Reynolds et al CAV17]

- Support extended string functions commonly used in applications
  - `substr(x,n,m)` substring of `x` at position `n` of length at most `m`
  - `ctn(x,y)` true if `x` contains substring `y`
  - `indexof(x,y,n)` position of `y` in `x` starting from position `n`
  - `replace(x,y,z)` result of replacing first occurrence of `y` in `x` by `z`

- For example: `¬ctn(x, "c")` denotes `x` does not contain substring "c"
Extended Theory of Strings [Reynolds et al CAV17]

- Use reduction lemmas
- Expensive:
  Introduces $3 \times |x|$ string vars

\[ x = \text{“ab”} \cdot y \quad y = \text{“c”} \quad \neg \text{ctn}(x, \text{“c”}) \]

String solver

\[ \forall 0 \leq n < |x|. \, \text{substr}(x, n, 1) \neq \text{“c”} \]

Bound length of $x$

\[ \text{substr}(x, 0, 1) \neq \text{“c”} \land \ldots \land \text{substr}(x, n, 1) \neq \text{“c”} \]

Reduce $\text{substr}$

\[ x = z_{11} \cdot k_1 \cdot z_{21} \land \ldots \land x = z_{1n} \cdot k_n \cdot z_{2n} \land \]

\[ |z_{11}| = 0 \land k_1 \neq \text{“c”} \land |z_{1n}| = n \land k_n \neq \text{“c”} \]
Context-Dependent Simplification [Reynolds et al CAV17]

- Alternatively: use context-dependent simplification:
  \[
  \begin{align*}
  x &= "ab" \cdot y \\
  y &= "c" \\
  \neg \text{ctn}(x, "c")
  \end{align*}
  \]

  \[
  x = "ab" \cdot y \land y = "c" \models x = "abc"
  \]
Context-Dependent Simplification [Reynolds et al CAV17]

- Alternatively: use context-dependent simplification:
  \[ x = "ab" \cdot y \land y = "c" \Rightarrow x = "abc" \]
- Thus:
  \[ \neg \text{ctn}(x, "c") \Rightarrow \neg \text{ctn}("abc", "c") \Leftrightarrow \bot \]

\[ x = "ab" \cdot y \land y = "c" \Rightarrow x = "abc" \]

By substitution

By rewriting
Recent Developments for Theory of Strings

• Context-dependent simplifications
  • Use aggressive rewriting [Reynolds et al CAV 2019]
  • Applied eagerly [Noetzli et al CAV 2022]

• Reduction lemmas
  • Leverage String-to-code point (code) conversion [Reynolds et al IJCAR 2020]
  • Improved encodings [Reynolds et al FMCAD 2020]
  • Applied lazily based on model [Noetzli et al CAV 2022]
Rewriting based on High-Level Abstractions

• Unlike arithmetic:

\[ x + x + 7y = y - 4 \quad \rightarrow \quad 2x + 6y + 4 = 0 \]

...rewrite rules for strings are *highly non-trivial*:

\[
\text{ctn("abcde", "b" \cdot x \cdot "a")} \quad \rightarrow \quad \bot
\]
\[
\text{substr(x \cdot "abcd", 1 + \text{len}(x), 2)} \quad \rightarrow \quad "bc"
\]
\[
\text{indexof("abc" \cdot x, "d" \cdot x, 0)} \quad \rightarrow \quad -1
\]

• Used syntax-guided synthesis to search for rewrite rules
  • Wrote 3000+ new LOC (C++) in cvc5’s string rewriter module
Rewriting based on High-Level Abstractions

• Rules based on high-level abstractions
  • Strings as #characters (e.g. reasoning about their length):
    \[ \text{ctn}(\text{substr}(x, i, j), x \cdot \text{“a”}) \]
    ...since the second argument is longer than the first

  • Strings as elements in containment lattice:
    \[ \text{ctn}(x \cdot y, \text{substr}(x, i, j)) \]
    ...since \( x \cdot y \) contains \( x \), which contains the second argument

  • Strings as multisets of characters:
    \[ x \cdot x \cdot y \cdot \text{“ab”} = x \cdot \text{“bbbbb”} \cdot y \]
    ...since LHS contains at least 1 more occurrences of “a”

⇒ With more rewrites, context-dependent simplification applies *more often*
A Decision Procedure for Code Points

• Even with aggressive simplification, still require reductions
  • Many extended function reductions require reasoning about characters

• **Idea**: extend core solver for strings to reason about *code points*
  • Assume ordering on characters of alphabet \( \mathcal{A} \):
    - \( c_1 < \ldots < c_{|\mathcal{A}|-1} \) where for each \( c_i \), we call \( i \) its code point
  • \( \text{code} : \text{Str} \rightarrow \text{Int} \) is interpreted as:
    1. For \( w \) in \( \mathcal{A}^1 \), \( \text{code} (w) \) is the code point of the single character in \( w \)
    2. For all other \( w \), \( \text{code} (w) \) is \(-1\)

• Fragment with string length + string code point (w/o concatenation):
  • Procedure is **sound, complete, terminating**
  • Can be combined modularly with the existing string solver

Reynolds et al IJCAR20
A Decision Procedure for Code Points

• **More efficient reductions** that leverage code, including:

• Conversion between strings and integers `to_int(x)`:
  - $\text{ite}(x[i]="9", 9, \text{ite}(x[i]="8", 8, \ldots \text{ite}(x[i]="0", 0, -1)\ldots)$
  - $\Rightarrow \text{ite}(48 \leq \text{code}(x[i]) \leq 57, \text{code}(x[i])-48, -1)$

...note 48 is Unicode for “0”

• Regular expression ranges $x \in \text{range}(c_1, c_2)$:
  - $\text{len}(x) = 1 \land (x = c_1 \lor \ldots \lor x = c_2)$
  - $\Rightarrow \text{code}(c_1) \leq \text{code}(x) \leq \text{code}(c_2)$

• Similar for conversions to lower/upper case, lexicographic ordering

$\Rightarrow$ Reasoning about code points is deferred to cvc5’s linear arithmetic solver
Revisiting Reductions for Strings

• **Observation:** there exist equivalent ways of expressing the same constraint
  • For strings $x, y$:
    \[
    \exists z. x = z \cdot y \land \text{len}(z) = 1 \\
    \text{substr}(x, 1, \text{len}(x) - 1) = y \\
    x \in \Sigma \cdot \text{to_re}(y)
    \]

    ... $y$ is the result of removing the first character from $x$

• **Idea:** *reuse variables* in extended functions and regular expression reductions

Reynolds et al FMCAD20
Revisiting Reductions for Strings

• Reduction for $\text{substr}(x, 1, n)$:
  \[
  \Rightarrow (\text{len}(x)>0 \land n>0) \Rightarrow (x=z_1 \cdot z_2 \cdot z_3 \land \text{len}(z_1)=1 \land \text{len}(z_2) \leq n \land \ldots)
  \]

• Map $\tilde{W}$ from variables to “witness form”
  • E.g. $\tilde{W}(z_1) = \text{substr}(x, 0, 1)$, $\tilde{W}(z_2) = \text{substr}(x, 1, n)$, $\tilde{W}(z_3) = \ldots$

• Reduction for $x \in \Sigma \cdot \text{to}_\text{re}(y)$:
  \[
  \Rightarrow x=z_4 \cdot z_5 \land z_4 \in \Sigma \land z_5 \in \text{to}_\text{re}(y)
  \Rightarrow x=z_1 \cdot z_5 \land z_1 \in \Sigma \land z_5 \in \text{to}_\text{re}(y)
  \]
  ... since first component $z_4$ also corresponds to $\text{substr}(x, 0, 1)$

• Witness forms can leverage rewriting $\downarrow$, share variables $z_i$ and $z_j$ when $\tilde{W}(z_i) \downarrow = \tilde{W}(z_j) \downarrow$
Even Faster Conflicts and Lazier Reductions

**Idea**: apply simplifications **eagerly** during CDCL(T) search

- Instrument congruence closure to detect conflicts via:
  - Rewriting
  - Inferred properties of equivalence classes
    - Upper/lower bounds for integer equivalence classes
    - Prefix and suffix approximations for string equivalence classes
- Report conflicts as soon as they arise
  - Avoids redundant search space

\[
\neg \text{ctn}(x, "c") \iff \bot
\]

---

Noetzli et al CAV22
Even Faster Conflicts and **Lazier Reductions**

- Avoid reasoning about *unnecessary* reduction lemmas

- Regular expression inclusion tests
  - E.g. do not reduce $x \in \Sigma^* a \Sigma^*$ if already reduced $x \in \Sigma^* a \Sigma^* b \Sigma^*$
    - Since $L(\Sigma^* a \Sigma^* b \Sigma^*) \subseteq L(\Sigma^* a \Sigma^*)$
  - Fast incomplete procedure for language inclusion
  - Can also be used for finding conflicts

- Model-based reductions
  - Construct candidate model $M$
  - Do not reduce e.g. string predicates $p$ that are already satisfied by $M$
  - Often, *negative* reg exp memberships are satisfied by current model
Even Faster Conflicts and Lazier Reductions

• Results on 10857 SMT-LIB string benchmarks, 1200 second timeout
  • cvc5 solves 10347, z3 solves 8863

Noetzli et al CAV22
Designing a Trustworthy String Solver
The Need for SMT Proofs

• Correctness of cvc5 is highly critical to applications
  • In particular, refutational soundness
    \[ \Rightarrow \text{An incorrect UNSAT response may tell a user a system is safe when it is not!} \]

• cvc5 is a highly complex code base
  • 150k+ LOC, constantly changing with new algorithmic advancements
    \[ \Rightarrow \text{Infeasible to verify statically} \]

• Solution: Instrument cvc5 to generate externally checkable proofs

---

CVC5

SAT

UNSAT + Proof

Proof Checker
Proofs in cvc5

• Covers many parts of the system
• Evaluated on many SMT-LIB theories
  Barbosa et al IJCAR22

• Highly detailed and complete
• Fine-grained proofs for rewrites, for strings
  Noetzli et al FMCAD22
Proofs in cvc5: Design Principles

• Flexible
  • Target several backend formats: LFSC, Lean, Alethe, visualization formats

• Also *internally* checkable
  • Use of native proof checker in cvc5 for the purposes of catching errors early

• Provide proofs for all components required for fast solving
  • User should not have to disable features when asking for proofs

• Acceptable performance overhead (~50% performance overhead)
  • Make all optimizations capable of tracking proofs
  • Lazy proof generation
String solving involves *many parts of the system*:
- Preprocessing
- SAT solver (resolution)
- CNF conversion
- Theory Combination
- UF / Congruence closure
- Linear Arithmetic Solver
- Rewriting
- Quantifier instantiation (for reductions)
- Strings Theory Solver
  - Core calculus (Liang et al CAV 2014)
  - Extended function reductions
  - Regular expression unfolding
Proof Architecture

Input F

Preprocessor
SAT Solver

Arithmetic
T-Combination
Strings
UF

T-Combination

Postprocessor
Proof Sketch
Proof (internal format)

Proof Converter X

SAT Solver
Arithmetic
Strings
T-Combination
UF

Proof Converter Y

Proof Checker X
Proof (format X)

cvc5

UNSAT +

UNSAT +
Future Directions
String Solving: Better, Faster

• **Better proofs:**
  • Fine-grained proofs for string rewrites
    • User control over granularity
  • Better integration with external proof checkers
  • Modular extraction of parts of the proof (e.g. SAT skeleton, theory lemmas)

• **Faster solver:**
  • Techniques specialized to constraints of interest to applications
  • More advanced solving architectures
Advanced Architectures in cvc5

- What if we used the CDCL(T) engine as a black box?
Advanced Architectures in cvc5

• What if we used the CDCL(T) engine as a black box?
Advanced Architecture: Portfolio

CDCL(T) → F → Options #1 → cvc5 → Options #n → CDCL(T)

Options #1

UNSAT SAT

UNSAT SAT
Advanced Architecture: Deep Restarts

- Idea: Restart after learning a set of literals that are implied by $F$
Deep Restarts

• Given input formula $F$, a learnable literal $l$ is:
  • Meets some syntactic criteria, e.g. $l$ is a literal from $F$
  • Is entailed by input, $F \models_T l$

• Strategy to apply deep restarts based on e.g. time threshold
  • Restart, preprocess, solve again

$\Rightarrow$ Preprocessing after learning may make problem significantly easier
Deep Restarts: Possible Variants

• Restart while saving other learned formulas?
  • E.g. theory lemmas based on usefulness criteria

• Maintain SAT solver state on restart?
  • Dynamic mapping between SAT and theory literals

• Save state to disk and restart later?

• Only solve for part of the input formula at a time?
Summary

• SMT solver cvc5 is efficient tool widely used in applications
  • Handles many problem domains
  • State-of-the-art for string solving

• Always looking for new features, faster techniques, increased trust

• Thanks for listening!