Reconstructing Fine-Grained Proofs of Rewrites Using a Domain-Specific Language

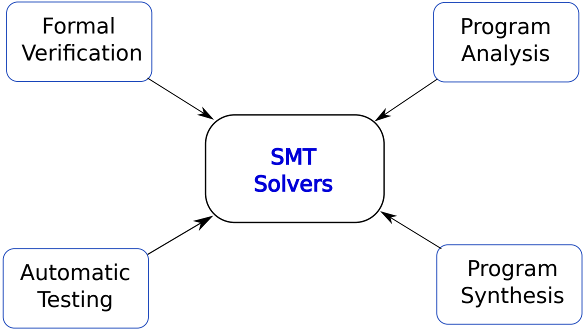
Andres Nötzli,[1] Haniel Barbosa,[2] Aina Niemetz,[1] Mathias Preiner,[1] Andrew Reynolds,[3] Clark Barrett,[1] Cesare Tinelli[3]

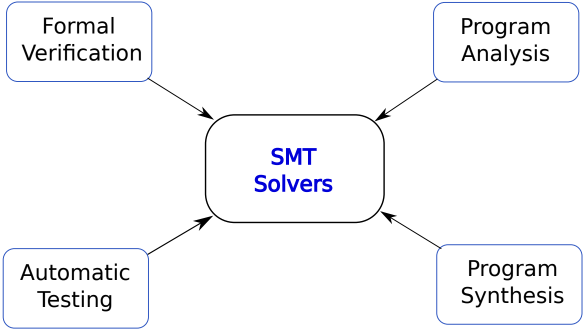
[1] Stanford University, [2] Universidade Federal de Minas Gerais, [3] The University of Iowa

Stanford | Center for Automated Reasoning

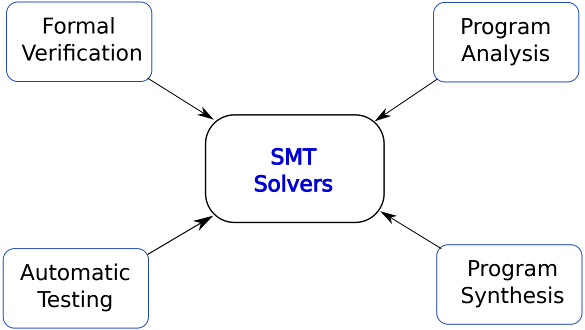
U F m G

The University of Iowa


Formal Verification
Program Analysis
SMT Solvers
Automatic Testing
Program Synthesis



Called billions of times a day in industry...



Called billions of times a day in industry...

- Even a tiny fraction of wrong answers is **bad**

Bugs in SMT Solvers

- State-of-the-art solvers are large projects:
  - Bitwuzla: 90k LoC (C/C++)
  - cvc5: 300k LoC (C++)
  - z3: 500k LoC (C++)

Bugs in SMT Solvers

- State-of-the-art solvers are large projects:
  - Bitwuzla: 90k LoC (C/C++)
  - cvc5: 300k LoC (C++)
  - z3: 500k LoC (C++)
- How do developers try to avoid bugs?
  - Code reviews
  - Testing on benchmark sets
  - Random input testing

Bugs in SMT Solvers

- State-of-the-art solvers are large projects:
  - Bitwuzla: 90k LoC (C/C++)
  - cvc5: 300k LoC (C++)
  - z3: 500k LoC (C++)
- How do developers try to avoid bugs?
  - Code reviews
  - Testing on benchmark sets
  - Random input testing
- But...
  - Disagreements between solvers at SMT-COMP
  - Fuzzing tools often find bugs in solvers

Solution: Checking Outputs

For satisfiable inputs: Evaluate formula on values of model generated by solver

Solution: Checking Outputs

For satisfiable inputs: Evaluate formula on values of model generated by solver

$$\text{contains}(x, \texttt{"FMCAD"}) \quad \wedge \quad |x| \geq 5$$

Solution: Checking Outputs

For satisfiable inputs: Evaluate formula on values of model generated by solver

$$\text{contains}(x, \texttt{"FMCAD"}) \quad \wedge \quad |x| \geq 5$$

Model: $\mathcal{M} = \{x \mapsto \texttt{"FMCAD-2022"}\}$

Solution: Checking Outputs

For satisfiable inputs: Evaluate formula on values of model generated by solver

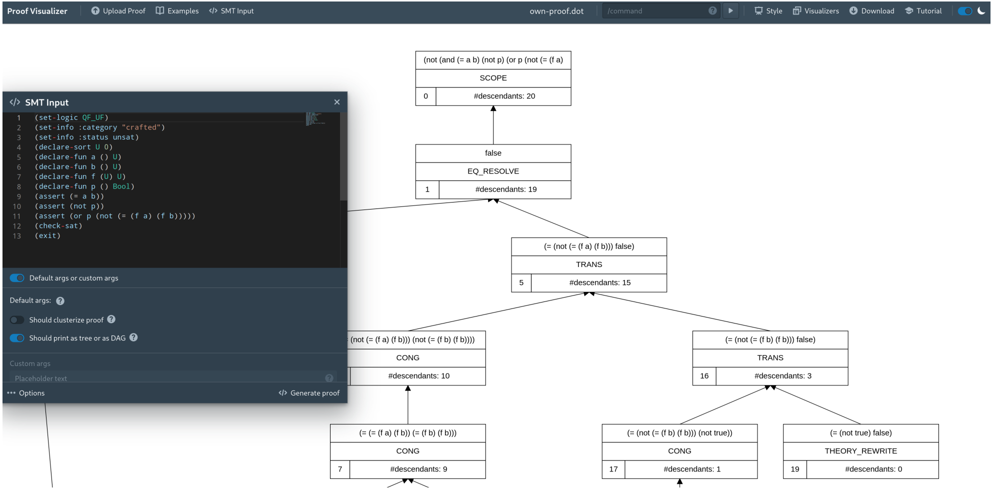
$$\text{contains}(x, \texttt{"FMCAD"}) \quad \wedge \quad |x| \geq 5$$

Model: $\mathcal{M} = \{x \mapsto \texttt{"FMCAD-2022"}\}$

▶ What about unsatisfiable inputs?

Proofs: A New Hope

- Proofs are a justification of the logical reasoning the solver has performed to find a solution
- A proof can be checked *independently*
  - Smaller trusted base: LFSC 5.5k (C++) + 2k (signatures) LoC vs. cvc5 300k LoC
  - Proof checking is generally more efficiently than solving the problem
- Other advantages
  - Confidence in results is decoupled from solver's implementation
  - Automation in interactive theorem proving
  - Formalization of proof rules improves code base, debugging



https://ufmg-smite.github.io/proof-visualizer/

The Challenge with Rewrites

- Modern SMT solvers implement *hundreds* of rewriting rules for state-of-the-art performance
  - String solver in cvc5: Over 200 rules in $3{,}000$ lines of C++ code

$$\text{substr}(\texttt{""}, m, n) \rightsquigarrow \texttt{""}$$

The Challenge with Rewrites

- Modern SMT solvers implement *hundreds* of rewriting rules for state-of-the-art performance
  - String solver in cvc5: Over 200 rules in $3{,}000$ lines of C++ code

$$\text{substr}(\texttt{""}, m, n) \rightsquigarrow \texttt{""}$$

- Many proof applications require *detailed* proofs
  - Easier proof checking, better integration with interactive theorem provers

The Challenge with Rewrites

- Modern SMT solvers implement *hundreds* of rewriting rules for state-of-the-art performance
  - String solver in cvc5: Over 200 rules in $3{,}000$ lines of C++ code

$$\text{substr}(\texttt{""}, m, n) \rightsquigarrow \texttt{""}$$

- Many proof applications require *detailed* proofs
  - Easier proof checking, better integration with interactive theorem provers

$$\text{rew}\ \frac{}{\text{substr}(\texttt{""}, m, n) \approx \texttt{""}} \qquad \text{rew}\ \frac{}{x + 0 \approx x} \qquad \text{rew}\ \frac{}{\neg(\neg p) \approx p}$$

The Challenge with Rewrites

- Modern SMT solvers implement *hundreds* of rewriting rules for state-of-the-art performance
  - String solver in cvc5: Over 200 rules in $3{,}000$ lines of C++ code

$$\mathsf{substr}(\texttt{""}, m, n) \rightsquigarrow \texttt{""}$$

- Many proof applications require *detailed* proofs
  - Easier proof checking, better integration with interactive theorem provers

$$\mathsf{substr\text{-}empty}\ \cancel{\mathsf{rew}}\ \frac{}{\mathsf{substr}(\texttt{""}, m, n) \approx \texttt{""}} \qquad \mathsf{add\text{-}zero}\ \cancel{\mathsf{rew}}\ \frac{}{x + 0 \approx x} \qquad \mathsf{dbl\text{-}neg}\ \cancel{\mathsf{rew}}\ \frac{}{\neg(\neg p) \approx p}$$

The Challenge with Rewrites

- Modern SMT solvers implement *hundreds* of rewriting rules for state-of-the-art performance
  - String solver in cvc5: Over 200 rules in $3{,}000$ lines of C++ code

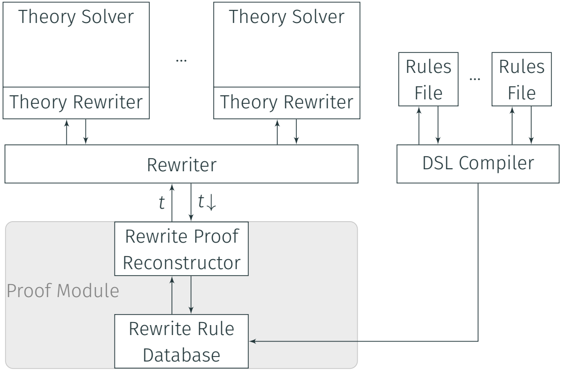
$$\text{substr}(\texttt{""}, m, n) \rightsquigarrow \texttt{""}$$

- Many proof applications require *detailed* proofs
  - Easier proof checking, better integration with interactive theorem provers

$$\text{substr-empty}\ \cancel{\text{rew}}\ \frac{}{\text{substr}(\texttt{""}, m, n) \approx \texttt{""}} \qquad \text{add-zero}\ \cancel{\text{rew}}\ \frac{}{x + 0 \approx x} \qquad \text{dbl-neg}\ \cancel{\text{rew}}\ \frac{}{\neg(\neg p) \approx p}$$

- Traditional approach: More instrumentation!
  - Difficult and tedious: Define proof rule and instrument code for every rewrite

Proofs for Rewrites: Our Approach



- Treat rewriter as black box and reconstruct proofs for rewrites externally
- A domain-specific language (DSL), RARE, to specify a database of rewrite rules
- A compiler for RARE that generates the C++ code that populates the rewrite rule database
- A general reconstruction algorithm, applied as a *post-processor*

Agenda

- A tour of RARE
- Proof reconstruction
- Implementation/Evaluation

RARE: Design Goals

- *Succinct*: Writing rewrite rules should be simple and concise
- *Expressive*: Support for the majority of the rewrite rules in a state-of-the-art solver
- *Accessible*: Easy to parse and familiar for developers

Rare: Basic Rules

```
(define-rule substr-empty ((m Int) (n Int))
  (str.substr "" m n) "")
```

Rare: Basic Rules

```
(define-rule substr-empty ((m Int) (n Int))
  (str.substr "" m n) "")
```

- Parts: Name, arguments, match expression, target expression
- Syntax is an extension of SMT-LIB

RARE: Matching

```
(define-rule eq-refl ((t ?)) (= t t) true)
```

RARE: Matching

```
(define-rule eq-refl ((t ?)) (= t t) true)
```

- Generic sorts
- All occurrences of argument must match same term

Rare: Lists

```
(define-rule str-concat-flatten (
    (xs String :list) (s String)
    (ys String :list) (zs String :list))
  (str.++ xs (str.++ s ys) zs) ; match
  (str.++ xs s ys zs))         ; target
```

RARE: Lists

```
(define-rule str-concat-flatten (
    (xs String :list) (s String)
    (ys String :list) (zs String :list))
  (str.++ xs (str.++ s ys) zs) ; match
  (str.++ xs s ys zs))         ; target
```

- Support for matching n-ary functions using list arguments
- List arguments can match zero terms

Rare: Conditional Rules

```
(define-cond-rule concat-clash (
    (s1 String) (s2 String :list)
    (t1 String) (t2 String :list))
  (and (= (str.len s1) (str.len t1)) ; precondition
    (not (= s1 t1)))
  (= (str.++ s1 s2) (str.++ t1 t2))  ; match
  false)                              ; target
```

Rare: Conditional Rules

```
(define-cond-rule concat-clash (
    (s1 String) (s2 String :list)
    (t1 String) (t2 String :list))
  (and (= (str.len s1) (str.len t1)) ; precondition
    (not (= s1 t1)))
  (= (str.++ s1 s2) (str.++ t1 t2))  ; match
  false)                             ; target
```

- Must show that the precondition holds for rewrite to apply

RARE: Fixed-Point Rules

```
(define-rule* str-len-concat-rec (
    (s1 String) (s2 String)
    (rest String :list))
  (str.len (str.++ s1 s2 rest)) ; match
  (str.len (str.++ s2 rest))    ; target
  (+ (str.len s1) _))           ; context
```

RARE: Fixed-Point Rules

```
(define-rule* str-len-concat-rec (
    (s1 String) (s2 String)
    (rest String :list))
  (str.len (str.++ s1 s2 rest)) ; match
  (str.len (str.++ s2 rest))    ; target
  (+ (str.len s1) _))           ; context
```

- Optimization for rules that should be applied repeatedly

Reconstructing Proofs

Base Rules

$$\text{eval}\,\frac{}{t \approx t{\downarrow_{\mathrm{e}}}} \qquad \text{trans}\,\frac{r \approx s \quad s \approx t}{r \approx t} \qquad \text{cong}\,\frac{\vec{s} \approx \vec{t}}{f(\vec{s}) \approx f(\vec{t})} \qquad \text{ceval}\,\frac{\vec{s}{\downarrow} \approx \vec{t}{\downarrow}}{f(\vec{s}) \approx (f(\vec{t})){\downarrow_{\mathrm{e}}}}$$

A bounded recursive search to prove $t \approx s$:

1. If t and s evaluate to the same value then return eval

Reconstructing Proofs

Base Rules

$$\text{eval}\,\frac{}{t \approx t{\downarrow_{\mathrm{e}}}} \qquad \text{trans}\,\frac{r \approx s \quad s \approx t}{r \approx t} \qquad \text{cong}\,\frac{\vec{s} \approx \vec{t}}{f(\vec{s}) \approx f(\vec{t})} \qquad \text{ceval}\,\frac{\vec{s}{\downarrow} \approx \vec{t}{\downarrow}}{f(\vec{s}) \approx (f(\vec{t})){\downarrow_{\mathrm{e}}}}$$

A bounded recursive search to prove $t \approx s$:

1. If t and s evaluate to the same value then return eval
2. If $t \approx s$ rewrites to $\bot$: fail

Reconstructing Proofs

Base Rules

$$\text{eval}\frac{}{t \approx t{\downarrow_{\mathrm{e}}}} \qquad \text{trans}\frac{r \approx s \quad s \approx t}{r \approx t} \qquad \text{cong}\frac{\vec{s} \approx \vec{t}}{f(\vec{s}) \approx f(\vec{t})} \qquad \text{ceval}\frac{\vec{s}{\downarrow} \approx \vec{t}{\downarrow}}{f(\vec{s}) \approx (f(\vec{t})){\downarrow_{\mathrm{e}}}}$$

A bounded recursive search to prove $t \approx s$:

1. If t and s evaluate to the same value then return eval
2. If $t \approx s$ rewrites to $\bot$: fail
3. If $t \approx s$ has form $f(\vec{u}) \approx f(\vec{v})$ then try to prove $\vec{u} \approx \vec{v}$, return cong

Reconstructing Proofs

Base Rules

$$\text{eval}\,\frac{}{t \approx t{\downarrow_e}} \qquad \text{trans}\,\frac{r \approx s \quad s \approx t}{r \approx t} \qquad \text{cong}\,\frac{\vec{s} \approx \vec{t}}{f(\vec{s}) \approx f(\vec{t})} \qquad \text{ceval}\,\frac{\vec{s}{\downarrow} \approx \vec{t}{\downarrow}}{f(\vec{s}) \approx (f(\vec{t})){\downarrow_e}}$$

A bounded recursive search to prove $t \approx s$:

1. If t and s evaluate to the same value then return eval
2. If $t \approx s$ rewrites to $\bot$: fail
3. If $t \approx s$ has form $f(\vec{u}) \approx f(\vec{v})$ then try to prove $\vec{u} \approx \vec{v}$, return cong
4. If:
    - t has form $f(\vec{u})$
    - $\vec{u}$ rewrites to $\vec{c}$
    - $f(\vec{c})$ evaluates to the same as s

   then try to prove $\vec{u} \approx \vec{c}$, return ceval

Reconstructing Proofs

Base Rules

$$\text{eval}\,\frac{}{t \approx t{\downarrow_{\mathrm{e}}}} \qquad \text{trans}\,\frac{r \approx s \quad s \approx t}{r \approx t} \qquad \text{cong}\,\frac{\vec{s} \approx \vec{t}}{f(\vec{s}) \approx f(\vec{t})} \qquad \text{ceval}\,\frac{\vec{s}{\downarrow} \approx \vec{t}{\downarrow}}{f(\vec{s}) \approx (f(\vec{t})){\downarrow_{\mathrm{e}}}}$$

A bounded recursive search to prove $t \approx s$:

1. If t and s evaluate to the same value then return eval
2. If $t \approx s$ rewrites to $\bot$: fail
3. If $t \approx s$ has form $f(\vec{u}) \approx f(\vec{v})$ then try to prove $\vec{u} \approx \vec{v}$, return cong
4. If:
    - t has form $f(\vec{u})$
    - $\vec{u}$ rewrites to $\vec{c}$
    - $f(\vec{c})$ evaluates to the same as s

   then try to prove $\vec{u} \approx \vec{c}$, return ceval
5. Recursive call: Find matching rules for t, try to prove rewritten $t' \approx s$ and preconditions

Reconstructing Proofs: Example

Rule in database:

```
(define-cond-rule substr-empty-s (
    (s String) (m Int) (n Int))
  (= s "") (str.substr s m n) "")
```

Rewrite:

$$\text{substr}(\text{substr}(\texttt{"abc"}, 4, 1), m, n) \rightsquigarrow \texttt{""}$$

Reconstructing Proofs: Example

Rule in database:

```
(define-cond-rule substr-empty-s (
    (s String) (m Int) (n Int))
  (= s "") (str.substr s m n) "")
```

Rewrite:

$$\text{substr}(\text{substr}(\texttt{"abc"}, 4, 1), m, n) \rightsquigarrow \texttt{""}$$

Rule matches, recursive call with new goal:

$$\text{substr}(\texttt{"abc"}, 4, 1) \approx \texttt{""}$$

Reconstructing Proofs: Example

Rule in database:

```
(define-cond-rule substr-empty-s (
    (s String) (m Int) (n Int))
  (= s "") (str.substr s m n) "")
```

Rewrite:

$$\mathrm{substr}(\mathrm{substr}(\texttt{"abc"}, 4, 1), m, n) \rightsquigarrow \texttt{""}$$

Rule matches, recursive call with new goal:

$$\mathrm{substr}(\texttt{"abc"}, 4, 1) \approx \texttt{""}$$

- Show using evaluation

Reconstructing Proofs: Example

Rule in database:

```
(define-cond-rule substr-empty-s (
    (s String) (m Int) (n Int))
  (= s "") (str.substr s m n) "")
```

Rewrite:

$$\text{substr}(\text{substr}(\texttt{"abc"}, 4, 1), m, n) \rightsquigarrow \texttt{""}$$

Rule matches, recursive call with new goal:

$$\text{substr}(\texttt{"abc"}, 4, 1) \approx \texttt{""}$$

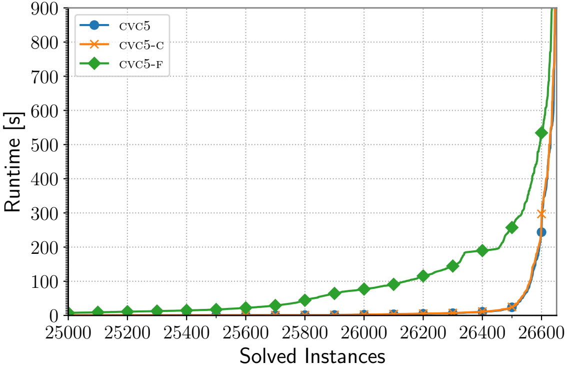
- Show using evaluation

$$\text{substr-empty-s}\ \dfrac{\text{eval}\ \dfrac{}{\text{substr}(\texttt{"abc"}, 4, 1) \approx \texttt{""}}}{\text{substr}(\text{substr}(\texttt{"abc"}, 4, 1), m, n) \approx \texttt{""}}$$

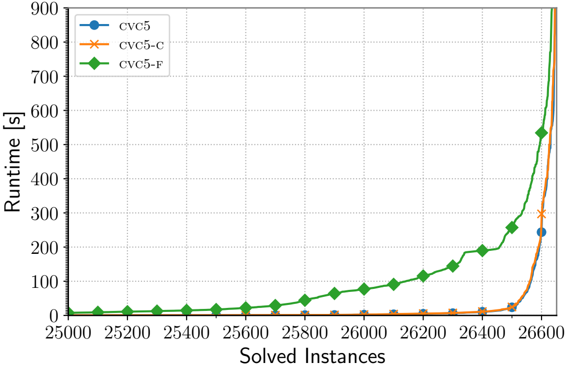
Implementation and Evaluation

- Implemented in cvc5 with focus on theory of strings
- Rewrite rules:
  - 40 rules for the theory of strings
  - 25 rules for integer arithmetic, complemented with manual rule for polynomial normalization
  - 22 rules for Boolean terms
- Benchmark sets:
  - 25 unsatisfiable industrial benchmarks
  - $26{,}626$ unsatisfiable SMT-LIB benchmarks

Evaluation: Results



Evaluation: Results



- Rewrites reconstructed: 95% for problems from the industrial set and of 87% for SMT-LIB
- Fully detailed: 20% of the proofs for industrial benchmarks, 23% of all proofs for SMT-LIB benchmarks with rewrite steps (6,120 out of 26,611)

Conclusion

- Proofs can be used to check answers of SMT solvers
- Approaches for proof generation
  - Traditional: Instrument code
  - Alternative: Reconstruction as a post-processing step
- Rare is a DSL for defining a rewrite rule database
- Implementation in cvc5, can reconstruct a proof for most rewrites in string benchmarks

`https://cvc5.github.io/`