CVC4Sy: Smart and Fast Term Enumeration for Syntax-Guided Synthesis

https://github.com/CVC4/CVC4

Andrew Reynolds
Haniel Barbosa
Cesare Tinelli

The University of Iowa

Andres Nötzli
Clark Barrett

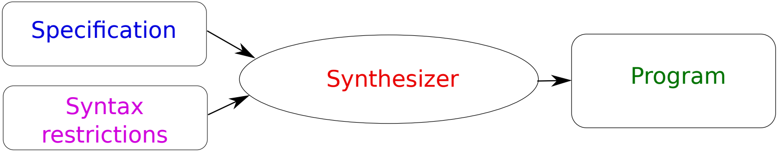
STANFORD

Artifact • Consistent • Complete • Well Documented • Easy to Reuse • Evaluated • CAV • AEC

CAV 2019

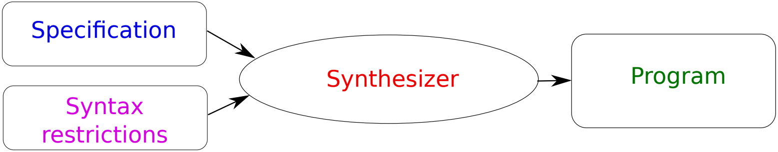
2019–07–17, New York, USA

Syntax-Guided Synthesis (SyGuS) [Alur et al. FMCAD'13]



- Specification is given by T-formula: $\exists f. \forall \bar{x}. \varphi[f, \bar{x}]$
- Syntactic restrictions given by context-free grammar R

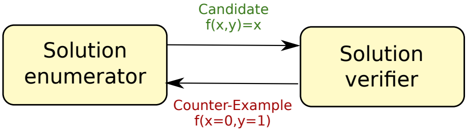
Syntax-Guided Synthesis (SyGuS) [Alur et al. FMCAD'13]



- ▷ Specification is given by T-formula: $\exists f.\, \forall \bar{x}.\, \varphi[f, \bar{x}]$
- ▷ Syntactic restrictions given by context-free grammar R

- ▷ Commonly solved via enumerative CEGIS [Solar-Lezama et al. ASPLOS'06]

$I ::= 0 \mid 1 \mid x \mid y \mid I + I \mid I - I$



$\exists f.\, \forall \bar{x}.\, f(x, y) > x + 1$

CVC4Sy: SyGuS extension of the CVC4 SMT solver

- CVC4 is an efficient SMT solver supporting a wide range of theories
  - Strings, bit-vector, (non-)linear arithmetic, algebraic datatypes, ...

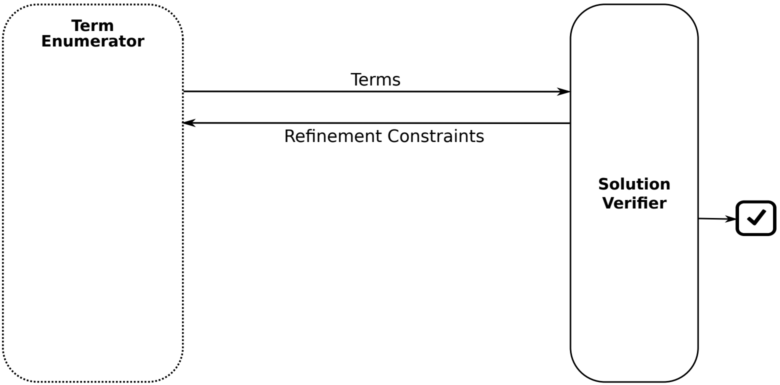
- SyGuS solver is based on a combination of methods
  - Enumerative CEGIS
  - Advanced techniques
    - Counterexample-guided quantifier instantiation [Reynolds et al. CAV'15]
    - Divide-and-conquer enumeration via decision tree learning [Alur et al. TACAS'17, Barbosa et al. FMCAD'19]

CVC4Sy: SyGuS extension of the CVC4 SMT solver

- CVC4 is an efficient SMT solver supporting a wide range of theories
  - Strings, bit-vector, (non-)linear arithmetic, algebraic datatypes, ...

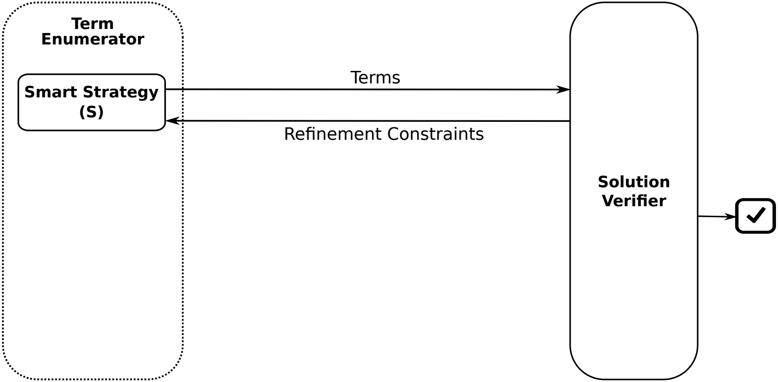
- SyGuS solver is based on a combination of methods
  - Enumerative CEGIS
  - Advanced techniques
    - Counterexample-guided quantifier instantiation [Reynolds et al. CAV'15]
    - Divide-and-conquer **enumeration** via decision tree learning [Alur et al. TACAS'17, Barbosa et al. FMCAD'19]

Enumerative synthesis in CVC4Sy



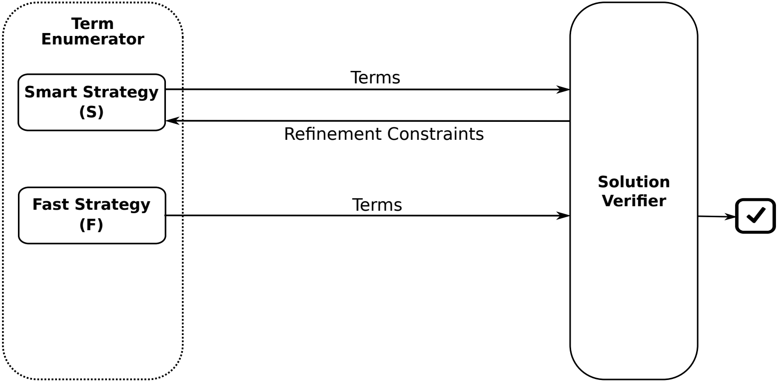
- *Bounded* enumeration, according to term ordering (e.g. term size)
- If there is a solution up to enumeration threshold, guaranteed to find it

Enumerative synthesis in CVC4Sy



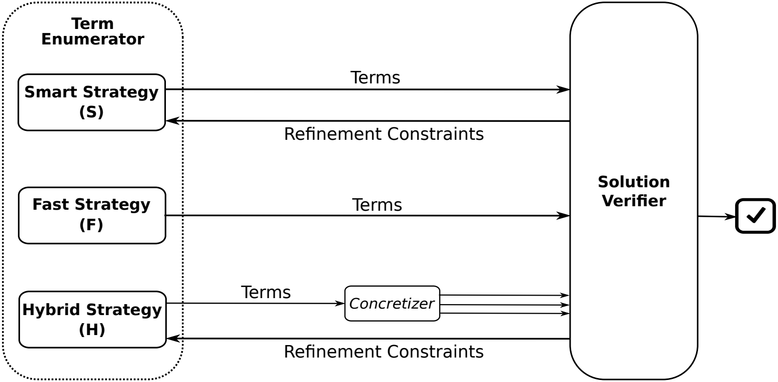
- *Bounded* enumeration, according to term ordering (e.g. term size)
- If there is a solution up to enumeration threshold, guaranteed to find it

Enumerative synthesis in CVC4SY



- *Bounded* enumeration, according to term ordering (e.g. term size)
- If there is a solution up to enumeration threshold, guaranteed to find it

Enumerative synthesis in CVC4SY



- ▷ *Bounded* enumeration, according to term ordering (e.g. term size)
- ▷ If there is a solution up to enumeration threshold, guaranteed to find it

From Grammars to Datatypes to Theory Terms

- Syntax restrictions encoded as algebraic datatypes

Grammar

$$\begin{array}{rcl} I & ::= & 0 \mid 1 \mid x \mid y \mid I + I \mid I - I \mid \mathsf{ite}(B, I, I) \\ B & ::= & B \geq B \mid I \simeq I \mid \neg B \mid B \wedge B \end{array}$$

Datatypes

$$\begin{array}{rcl} \mathcal{I} & = & \mathsf{0} \mid \mathsf{1} \mid \mathsf{x} \mid \mathsf{y} \mid \mathsf{plus}(\mathcal{I},\mathcal{I}) \mid \mathsf{minus}(\mathcal{I},\mathcal{I}) \mid \mathsf{ite}(\mathcal{B},\mathcal{I},\mathcal{I}) \\ \mathcal{B} & = & \mathsf{geq}(\mathcal{I},\mathcal{I}) \mid \mathsf{eq}(\mathcal{I},\mathcal{I}) \mid \mathsf{not}(\mathcal{B}) \mid \mathsf{and}(\mathcal{B},\mathcal{B}) \end{array}$$

- Datatype values are translated to corresponding theory terms

$$\mathsf{plus}(\mathsf{x}, 1) \rightarrow x + 1$$

Smart Strategy

- Uses datatype solver to generate new terms
- Redundant candidates are blocked via learned constraints
- Admits several optimizations via different classes of constraints

Example

Blocking the candidate $x + 1$:

$$\neg \mathsf{is}_{\mathsf{plus}}(d) \vee \neg \mathsf{is}_{\mathsf{x}}(\mathsf{sel}_1^{\mathcal{I}}(d)) \vee \neg \mathsf{is}_{1}(\mathsf{sel}_2^{\mathcal{I}}(d))$$

where d is the datatype constant representing the solution.

Blocking via Theory Rewriting with Generalization

- Pervasive goal: enumerate fewer terms!
- Terms equivalent up to rewriting are redundant
  - Blocking constraints added to discard redundancies

Blocking via Theory Rewriting with Generalization

- Pervasive goal: enumerate fewer terms!
- Terms equivalent up to rewriting are redundant
  - Blocking constraints added to discard redundancies
- Sometimes the redundancy is maintained even with different subterms
- Blocking minimal term skeleton that determines rewritten form
  - Replace each subterm in given term by fresh variable
  - Rewrite
  - Check if rewritten form stays the same

Example

$\mathsf{ite}(x \simeq 0 \wedge y \geq 0,\, 0,\, x)\!\downarrow = x\!\downarrow$

Blocking via Theory Rewriting with Generalization

- Pervasive goal: enumerate fewer terms!
- Terms equivalent up to rewriting are redundant
  - Blocking constraints added to discard redundancies
- Sometimes the redundancy is maintained even with different subterms
- Blocking minimal term skeleton that determines rewritten form
  - Replace each subterm in given term by fresh variable
  - Rewrite
  - Check if rewritten form stays the same

Example

$\text{ite}(x \simeq 0 \wedge y \geq 0, 0, x){\downarrow} = x{\downarrow}$ but the subterm $y \geq 0$ is irrelevant:
$\text{ite}(x \simeq 0 \wedge w, 0, x){\downarrow} = x{\downarrow}$.

Other optimizations

- Blocking via *CEGIS with Generalization*
  - Generalize failed candidate solutions
    Ex.: If ite$(x \geq 0,\, x,\, y+1)$ fails on point $(3,3)$ and $f(x,y) \leq x-1$ then we can block all ite$(x \geq 0,\, x,\, _)$

- Blocking via *Evaluation Unfolding*
  - Encode relationships between datatype and theory terms
  - Partially evaluates candidates on counterexamples during enumeration

Fast Strategy

- Smart strategy generates a large number of blocking constraints and effectiveness of optimizations depends on grammar
- Alternative: brute-force enumeration rather than constraint solving
  - Incompatible with generalizations and evaluation unfolding

Algorithm

Given an upper bound on term size k, for all $k_1 + \ldots + k_n + \text{ite}(n > 0, 1, 0) = k$:

- Enumerate terms of size k_i of type τ_i, store in $S_{\tau_i}^{k_i}$
- Add $\mathsf{C}(t_1, \ldots, t_n)$ to $S_{\tau_i}^{k}$ with $t_i \in S_{\tau_i}^{k_i}$ for all constructors

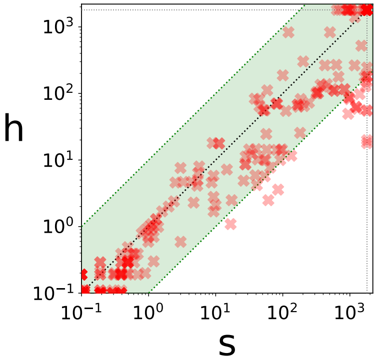
- Cache terms globally, only add terms unique up to rewriting

Evaluation

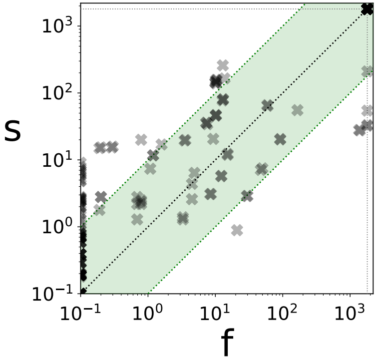
- Benchmark sets:
  - SyGuS-COMP 2018: all five tracks
  - Lustre: invariant synthesis problems for the verification of Lustre models
  - IC-BV: invertibility conditions for bit-vector operators
  - CegisT: bit-vector synthesis problems
- Comparison against EUSolver
- Evaluated the impact of different enumeration strategies and each of the optimizations

Comparisons

- Sometimes better to be smart
  - **s**: smart, **f**: fast, **h**: hybrid



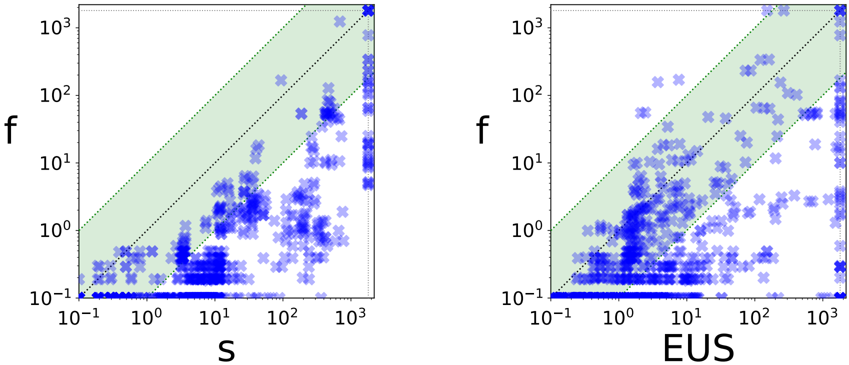
Lustre set (invariant synthesis)
1800s timeout, 485 benchmarks



CrCi set (cryptography circuits)
1800s timeout, 214 benchmarks

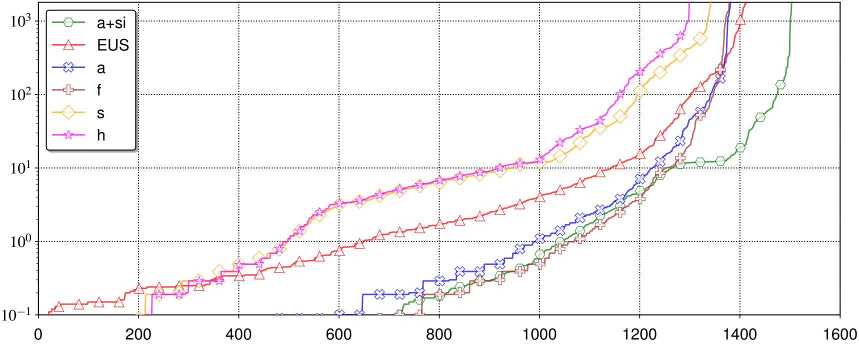
Comparisons

- Sometimes better to just be fast
  - **f**: fast, **s**: smart, **EUS**: EUSolver



PBE-Bitvectors and PBE-Strings sets
1800s timeout, 862 benchmarks

Comparisons



1800s timeout, 1704 problems from SyGuS-COMP'18

- **a**: auto mode picks best enumeration strategy depending on problem
- **si**: single-invocation solver used when quantifier-elimination can be applied to an input (only 16% of benchmarks)

Conclusions

- CVC4Sy is a state-of-the-art SyGuS solver
- SyGuS-COMP'15-18: won CLIA track
- SyGuS-COMP'18-19: won General and PBE tracks
- SyGuS-COMP'19: won Invariant track for the first time
  - New Unif+PI enumeration [Barbosa, Reynolds et al. FMCAD'19]
- Recent improvements include
  - Extensions to the theory of datatypes [Reynolds et al. IJCAR'18]
  - Better rewrites in the underlying SMT solver
    - SyGuS for rewrite rule enumeration [Nötzli, Reynolds, Barbosa et al. SAT'19]
    - Better string rewrites [Reynolds, Nötzli et al. CAV'19]

CVC4Sy: Smart and Fast Term Enumeration for Syntax-Guided Synthesis

https://github.com/CVC4/CVC4

Andrew Reynolds
Haniel Barbosa
Cesare Tinelli

THE UNIVERSITY OF IOWA

Andres Nötzli
Clark Barrett

STANFORD

CAV 2019

2019–07–17, New York, USA

Artifact ⋆ Consistent ⋆ Complete ⋆ Well Documented ⋆ Easy to Reuse ⋆ Evaluated ⋆ CAV ⋆ AEC