CVC4SY for SyGuS-COMP 2019

Andrew Reynolds, Haniel Barbosa, Cesare Tinelli
The University of Iowa

Andres Nötzli, Clark Barrett
Stanford University

***Abstract*—CVC4SY is a syntax-guided synthesis (SyGuS) solver based on bounded term enumeration and, for restricted fragments, quantifier elimination. The enumerative strategies are based on encoding term enumeration as an extension of the quantifier-free theory of algebraic datatypes and on a highly optimized brute-force algorithm. The quantifier elimination strategy extracts solutions from unsatisfiability proofs of the negated form of synthesis conjectures. It uses recent counterexample-guided techniques for quantifier instantiation that make finding such proofs practically feasible. CVC4SY implements these strategies by extending the satisfiability modulo theories (SMT) solver CVC4. The strategy to be applied on a given problem is chosen heuristically based on the problem's structure. This document gives an overview of these techniques and their implementation in the SyGuS Solver CVC4SY, an entry for SyGuS-Comp 2019.**[1]

I. A Refutation-Based Approach for Synthesis

We consider the synthesis problem in the context of some theory T. In this setting, a *synthesis conjecture* is expressed as a formula of the form

$$\exists f \forall x_1 \cdots \forall x_n . P[f, x_1, \ldots, x_n] \tag{1}$$

where the second-order variable f represents the function to be synthesized and P is a formula encoding properties that f must satisfy for all possible values of the input tuple $\bar{x} = (x_1, \ldots, x_n)$. In this setting, finding a witness for this satisfiability problem amounts to finding a function for f in some model of T that satisfies $\forall \bar{x}. P[f, \bar{x}]$.

To determine the satisfiability of $\exists f \forall \bar{x}. P[f, \bar{x}]$ an SMT solver [6] can consider the satisfiability of the (open) formula $\forall \bar{x}. P[f, \bar{x}]$ by treating f as an uninterpreted function symbol. We instead use an approach to synthesis geared toward establishing the *unsatisfiability of its negation*:

$$\forall f \exists \bar{x}. \neg P[f, \bar{x}] \tag{2}$$

A syntactic solution for (1) can be constructed from a refutation of (2), as opposed to being extracted from the valuation of f in a model of $\forall \bar{x}. P[f, \bar{x}]$. Proving (2) unsatisfiable poses a challenge to current SMT solvers, namely, dealing with the second-order universal quantification of f. We use two specialized methods (Sections I-A and I-B) to refute negated synthesis conjectures like (2) that build on existing capabilities of these solvers. These are described in detail in [10, 12].

[1]This work was partially supported by the National Science Foundation under award 1656926 and by the Defense Advanced Research Projects Agency (award FA8650-18-2-7854).

A. Syntax-Guided Enumeration

The first method is general, and follows the *syntax-guided synthesis* paradigm [2] where the synthesis conjecture is accompanied by an explicit syntactic restriction on the space of possible solutions. Similarly to other SyGuS solvers, the syntax-guided search in CVC4SY [9] is based on counterexample-guided inductive synthesis (CEGIS) [15]: a refinement loop in which a learner proposes solutions, and a verifier, generally a satisfiability modulo theories (SMT) solver, checks them and provides counterexamples for failures. Generally, the learner enumerates some set of terms, while pruning spurious ones [16].

Our syntax-guided synthesis method is based on encoding the syntax of terms as first-order values. We use a deep embedding into an extension of the background theory T with a theory of algebraic data types, encoding the restrictions of a syntax-guided synthesis problem. In detail, a set of syntactic restrictions for a function f may be expressed as an algebraic datatype whose constructors represent the possible operators for f. For instance, consider the case where f has type $\mathsf{Int} \to \mathsf{Int}$. The algebraic datatype:

$$\mathsf{S} \;:=\; \mathsf{x}_1 \mid \mathsf{zero} \mid \mathsf{one} \mid \mathsf{plus}(\mathsf{S}, \mathsf{S})$$

encodes a term signature for f that includes nullary constructors for the variable x_1 (the first input argument of f), and constructors for the symbols of the arithmetic theory T. Terms of sort S refer to terms of sort Int. We extend the background theory with an *evaluation operator* e_{S} or a function of type $\mathsf{S} \to (\mathsf{Int} \to \mathsf{Int})$, whose semantics evaluates the analog of its first argument on the second argument. For instance, $e_{\mathsf{S}}(\mathsf{plus}(\mathsf{x}_1, \mathsf{one}), 3) = 4$. Now, consider the (negated) synthesis conjecture $\forall f \exists x. \neg P[f, x]$, where solutions for f are restricted to the signature described by datatype S. This conjecture may be phrased as the (first-order) formula:

$$\forall z \exists \bar{x}. \neg P[e_{\mathsf{S}}(z), \bar{x}] \tag{3}$$

where z has type S. The solver may show (3) unsatisfiable by finding a (single) term t of type S for which $\exists \bar{x}. \neg P[e_{\mathsf{S}}(t), \bar{x}]$ is unsatisfiable in an extension of T. When this is the case, the solution for $\forall f \exists \bar{x}. \neg P[f, \bar{x}]$ is constructed by traversing the structure of t, while replacing constructor symbols with their corresponding symbols from the signature of T.

The enumeration performed by CVC4SY is parameterized by an enumeration strategy chosen before solving: it either applies a constraint-based (*smart*) enumeration, which allows for numerous optimizations [9, Section 2]; a new approach

for (*fast*) enumerative synthesis [9, Section 3], which applies a highly optimized brute-force algorithm and has significant advantages with respect to the smart enumeration; and a *hybrid* approach combining smart and fast enumeration [9, Section 4]. All enumeration strategies rely on techniques that ensure subterms in candidate solutions are unique up to theory-specific rewriting. For example, since terms x_1 and $x_1 + 0$ are equivalent, a *symmetry breaking clause* such as $\neg\mathsf{isplus}(t) \vee \neg\mathsf{iszero}(t.2)$ can be used to avoid enumerating solutions where the second child of t, denoted $t.2$, is zero.

B. Quantifier Instantiation for Single-Invocation Properties

The second method applies to a restricted, but fairly common, case of synthesis conjectures. When axiomatizing properties of a desired function f, a particularly well-behaved class are *single-invocation properties*. Similar classes of conjectures have been studied in recent work [3, 7].

A *single-invocation property* is any formula of the form $Q[\bar{x}, f(\bar{x})]$ obtained as an instance of a quantifier-free formula $Q[\bar{x}, y]$ not containing f. In other words, the only occurrences of f in $Q[\bar{x}, f(\bar{x})]$ are in subterms of the form $f(\bar{x})$ with the *same* tuple $\bar{x}$ of *pairwise distinct* variables. The conjecture $\forall f\, \exists \bar{x}.\, \neg Q[\bar{x}, f(\bar{x})]$ is equivalent to the *first-order* formula:

$$\exists \bar{x}\, \forall y.\, \neg Q[\bar{x}, y] \tag{4}$$

To prove the unsatisfiability of (4), it suffices to find a T-unsatisfiable finite set Γ of ground instances of $\neg Q[\bar{\mathsf{k}}, y]$, where $\bar{\mathsf{k}}$ is a set of fresh uninterpreted constants. To find such a Γ, we use a specialized new technique, which we refer to as *counterexample-guided quantifier instantiation*. This technique is similar to CEGIS, but with the difference of being built directly into the SMT solver. The idea is to choose instantiations for $\forall y.\, \neg Q[\bar{\mathsf{k}}, y]$ based on models for $Q[\bar{\mathsf{k}}, \mathsf{e}]$, where e is a fresh constant, using a *selection function*. Selection functions are specific to the background theory. Recent work has shown selection functions for linear real and integer arithmetic [11] and implemented them in CVC4SY. As a consequence of this work, CVC4SY is a complete synthesis procedure for single-invocation synthesis conjectures over linear arithmetic.

Assuming the solver finds such a T-unsatisfiable Γ, say $\{\neg Q[\bar{\mathsf{k}}, t_1[\bar{\mathsf{k}}]], \ldots, \neg Q[\bar{\mathsf{k}}, t_p[\bar{\mathsf{k}}]]\}$, it can be shown that:

$$\lambda \bar{x}.\, \mathsf{ite}(Q[\bar{x}, t_p], t_p, (\cdots \mathsf{ite}(Q[\bar{x}, t_2], t_2, t_1) \cdots)) \tag{5}$$

is a solution for the synthesis conjecture $\forall f\, \exists \bar{x}\, Q[\bar{x}, f(\bar{x})]$. In the case that f has syntactic restrictions, we use heuristic enumerative techniques to find a function equivalent to (5) that meets such syntactic restrictions.

II. Enhancements

A. Optimizations for Enumerative Search in SMT

Since last year, we have added a number of optimizations to CVC4SY's syntax-guided enumerative search to make it faster [9]. A key element of the *smart* enumerative search in CVC4SY is the use of a theory of datatypes with *shared selectors* [14]. At a high level, this ensures a maximal number of (e.g., symmetry breaking) clauses are shared across multiple contexts of the search. The new *fast* enumerative search leads to gains in most problem categories, but is especially impactful on PBE problems, where it outperforms the smart strategy by several orders of magnitude. Such gains are significant given that CVC4SY won this track at SyGuS-COMP 2018 by employing the smart technique alone. We have also made significant improvements to CVC4SY's theory rewriting [8, 13], which is used to infer when a term is equivalent to a previous one and hence can be discarded. This benefits both smart and fast enumerations.

B. Strategy-Based Solution Construction

CVC4SY uses divide-and-conquer techniques inspired by [3, 4] to construct solutions for synthesis conjectures. At a high level, the approach enumerates a stream of terms and independently devises a strategy for combining them into candidate solutions. When applicable, CVC4SY uses strategies for building solutions involving ite-terms (to create decision trees), string concatenation concat in the PBE strings division (to sequence solutions), and combinations of the two.

C. Invariant Synthesis

For the invariant synthesis track, CVC4SY uses several static preprocessing passes to make the conjecture easier to solve. Most importantly, this includes rephrasing the invariant synthesis problem as finding a *strengthening of the post-condition* (resp. weakening of the pre-condition). In other words, given an invariant synthesis problem for predicate I, instead of synthesizing $I(\bar{x})$, CVC4SY may choose to instead synthesize a predicate $I'(\bar{x})$ such that $post(\bar{x}) \wedge I'(\bar{x})$ is the overall solution, where $post(\bar{x})$ is the post-condition of the invariant synthesis problem. Additionally, CVC4SY uses a new divide-and-conquer approach, Unif+PI, for function synthesis that works especially well for invariant synthesis [5]. It accumulates refinement lemmas, synthesizes partial solutions for each point in these lemmas independently, and uses a decision tree algorithm to combine these partial solutions into an overall solution.

D. Constant Repair

Since last year, CVC4SY implements new techniques [1] for using its theory solvers to synthesize constants to repair candidate solutions. To do so, occurrences of $(\mathsf{Constant}\ T)$ in grammars are treated as a constructor const containing a single field of type T. If a candidate solution is enumerated that involves this constructor, a subcall to CVC4 is made to find the appropriate constant, if one exists, such that the candidate solution (universally) satisfies the overall conjecture. For example, in terms of the deep embedding, if our candidate solution is $\mathsf{plus}(\mathsf{x}, \mathsf{const}(1))$ and our specification is $\exists f \forall x.\, f(x) > x + 100$, then we check the satisfiability of the quantified conjecture $\exists c \forall x.\, x + c > x + 100$, which is satisfiable with a model where, e.g., $c = 101$.

E. Inferring Equivalent Single-Invocation Properties

CVC4 uses techniques for recognizing when synthesis conjectures can be rewritten into a form that is single invocation.

This includes normalizing the arguments of invocations across conjunctions, and applying quantifier elimination to variables for which the function to synthesize is not applied.

F. Supporting New SyGuS Language

This year CVC4SY has been extended to support the new SyGuS language[2], which is a lightly-modified version of the SyGuS input format that was used in previous competitions. The new format is more compliant with SMT-LIB version 2.6, includes minor changes to the concrete syntax for commands, and eliminates several deprecated features of the previous format.

III. Configurations for SyGuS-COMP 2019

CVC4SY is entering all tracks of SyGuS-COMP 2019. For all tracks, CVC4SY performs the following steps, given a background theory T:

1) Parse the input and phrase the problem in terms of a (negated) synthesis conjecture of the form $\forall f \exists \bar{x}. \neg P[f, \bar{x}]$, and a set of syntactic restrictions R.
2) Do one of the following:
   a) Determine a single-invocation property $Q[\bar{y}, f(\bar{y})]$ that is equivalent to $P[f, \bar{x}]$. Let φ be $\forall \bar{z}. Q[\mathsf{k}, \bar{z}]$ and let T' be T.
   b) Let φ be $\forall z \exists \bar{x}. P[e_{\mathsf{S}}(z), \bar{x}]$, where S is an algebraic datatype that encodes R, and let T' be an extension of T whose signature includes S and e_{S}, as described in Section I-A.
3) Using the appropriate technique (either the one from Section I-A or I-B), show that φ is T'-unsatisfiable.
4) If successful, reconstruct a solution for the original synthesis conjecture $\forall f \exists \bar{x}. \neg P[f, \bar{x}]$.

For the GENERAL and CLIA tracks, CVC4SY prefers executing Step 2a over Step 2b. In the case that CVC4SY executes Step 2a but fails to reconstruct a solution in Step 4, CVC4SY restarts and executes Step 2b instead.

For some tracks, we enter different configurations with different synthesis strategies. We denote the *smart* enumerative strategy by **s** and the *fast* enumerative strategy by **f**. The Unif+PI strategy for invariant synthesis is denoted by **su**. We denote the the auto strategy, which heuristically picks a strategy based on the properties of the problem, by **auto**. It uses the single-invocation solver on problems that are amenable to quantifier elimination, strategy **f** on PBE problems and problems without the Boolean type or the ite operator in their grammar and strategy **s** otherwise.

CVC4SY has the following entries, per track, in SyGuS-COMP 2019:

Track	Configuration
CLIA	**auto**
GENERAL	**auto**, **f**, **s**
INV	**f**, **s**, **su**
PBE_BitVec	**f**, **s**
PBE_Strings	**f**, **s**

[2]https://sygus.org/assets/pdf/SyGuS-IF_2.0.pdf

References

[1] A. Abate, C. David, P. Kesseli, D. Kroening, and E. Polgreen. Counterexample guided inductive synthesis modulo theories. In H. Chockler and G. Weissenbacher, editors, *Computer Aided Verification (CAV), Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2018.

[2] R. Alur, R. Bodík, G. Juniwal, M. M. K. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. Syntax-guided synthesis. In *FMCAD*, pages 1–17. IEEE, 2013.

[3] R. Alur, P. Černý, and A. Radhakrishna. *Synthesis Through Unification*, pages 163–179. Springer International Publishing, Cham, 2015.

[4] R. Alur, A. Radhakrishna, and A. Udupa. Scaling enumerative program synthesis via divide and conquer. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 319–336. Springer, Berlin, Heidelberg, 2017.

[5] H. Barbosa, A. Reynolds, D. Larraz, and C. Tinelli. Extending enumerative function synthesis via SMT-driven classification. In C. Barrett and J. Yang, editors, *Formal Methods In Computer-Aided Design (FMCAD)*. (Accepted for publication). IEEE, 2019.

[6] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability Modulo Theories. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, 2009.

[7] D. Neider, S. Saha, and P. Madhusudan. Synthesizing piece-wise functions by learning classifiers. In M. Chechik and J. Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 186–203. Springer, 2016.

[8] A. Nötzli, A. Reynolds, H. Barbosa, A. Niemetz, M. Preiner, C. Barrett, and C. Tinelli. Syntax-guided rewrite rule enumeration for SMT solvers. In M. Janota and I. Lynce, editors, *Theory and Applications of Satisfiability Testing (SAT)*, (Accepted for publication). Lecture Notes in Computer Science. Springer, 2019.

[9] A. Reynolds, H. Barbosa, A. Nötzli, C. Barrett, and C. Tinelli. cvc4sy: Smart and fast term enumeration for syntax-guided synthesis. In I. Dillig and S. Tasiran, editors, *Computer Aided Verification (CAV), Part II*, volume 11562 of *Lecture Notes in Computer Science*, pages 74–83, Cham, 2019. Springer International Publishing.

[10] A. Reynolds, M. Deters, V. Kuncak, C. W. Barrett, and C. Tinelli. Counterexample guided quantifier instantiation for synthesis in CVC4. In *Computer Aided Verification (CAV)*. Springer, 2015.

[11] A. Reynolds, T. King, and V. Kuncak. Solving quantified linear arithmetic by counterexample-guided instantiation. *Formal Methods in System Design*, 51(3):500–532, 2017.

[12] A. Reynolds, V. Kuncak, C. Tinelli, C. Barrett, and M. Deters. Refutation-based synthesis in smt. *Formal Methods in System Design*, 2017.

[13] A. Reynolds, A. Nötzli, C. W. Barrett, and C. Tinelli. High-level abstractions for simplifying extended string constraints in SMT. In I. Dillig and S. Tasiran, editors, *Computer Aided Verification (CAV), Part II*, volume 11562 of *Lecture Notes in Computer Science*, pages 23–42. Springer, 2019.

[14] A. Reynolds, A. Viswanathan, H. Barbosa, C. Tinelli, and C. Barrett. Datatypes with shared selectors. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *International Joint Conference on Automated Reasoning (IJCAR)*, volume 10900 of *Lecture Notes in Computer Science*, pages 591–608. Springer, 2018.

[15] A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia, and V. A. Saraswat. Combinatorial sketching for finite programs. In J. P. Shen and M. Martonosi, editors, *ASPLOS*, pages 404–415. ACM, 2006.

[16] A. Udupa, A. Raghavan, J. V. Deshmukh, S. Mador-Haim, M. M. Martin, and R. Alur. Transit: Specifying protocols with concolic snippets. In *PLDI*, pages 287–296. ACM, 2013.