EPiC Series in Computing
Volume 51, 2017, Pages 69–75
ARCADE 2017. 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements

Challenges for Fast Synthesis Procedures in SMT

Andrew Reynolds[1]

Department of Computer Science, The University of Iowa

Abstract

A number of synthesis applications are made possible by automated tools for synthesis. Recent work has shown that SMT solvers, instead of just acting as subroutines for automated software synthesis tasks, can be instrumented to perform synthesis themselves. This paper focuses on some of the current challenges for developing fast and useful procedures in SMT solvers for the benefit of synthesis applications.

1 Introduction

A particularly challenging problem for modern automated reasoners is the task of *synthesis*. Abstractly, a synthesis problem asks whether there exists a structure for which some property holds universally. Examples of structures to synthesize include correct-by-construction program snippets, invariants for establishing program safety, and policy plans for controllers. The synthesis problem is particularly difficult for automated reasoners like SMT solvers, which excel at deductive reasoning, but have limited ability to invent new constructs, an ability inherently critical for synthesis. As a result, many current tools use enumerative brute-force techniques for exploring the space of possible solutions.

Modern synthesis tools, e.g. [26, 3], commonly act a separate exterior layer which periodically makes quantifier-free satisfiability calls to an underlying SMT solver. As a consequence, synthesis tools are not privy to the low-level inferences made by the SMT solvers, nor do SMT solvers actively participate in high-level decisions made by synthesis tools. Conversely, SMT solvers have largely been developed without specific consideration towards advancing the state-of-the-art in synthesis, and often lack a coherent vision of the constraints they are solving.

Recent theoretical and empirical results indicate that procedures for quantified formulas over certain background theories in SMT solvers have reached a level of maturity that makes them viable in real world applications. The use of such procedures is limited in current verification and synthesis tools. The development of general-purpose SMT solvers for synthesis that leverage these procedures has the potential to meet the needs of a large audience and have a broad impact on software applications.

We consider the synthesis problem in the context of some background theory T. We say a *synthesis conjecture* is a formula of the form $\exists f \, \forall \vec{x} \, P[f, \vec{x}]$ where the second-order variable f represents the function to be synthesized and P is a first-order formula encoding properties that f must satisfy for all values of $\vec{x}$. We consider a conjecture of this form in the remainder of the paper, and assume f has sort $\sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma$.

In principle, to find a solution for f in a synthesis conjecture, an SMT solver supporting the theory T can consider the satisfiability of the open formula $\forall \vec{x} \, P[f, \vec{x}]$ by treating f as an uninterpreted function

symbol. This is challenging because it requires the solver to construct internally (a finite representation of) an interpretation of f that is guaranteed to satisfy $P[f, \vec{x}]$ for every value of $\vec{x}$ [15, 25].

More traditional SMT solver designs for handling universally quantified formulas have focused on instantiation-based methods to show *un*satisfiability [13]. While these techniques are incomplete in general, they have been shown to be quite effective in practice. For this reason, we advocate approaches to synthesis geared toward establishing the *unsatisfiability of the negation* of the synthesis conjecture:

$$\forall f \, \exists \vec{x} \, \neg P[f, \vec{x}] \tag{1}$$

The immediate challenge is how to deal with the second-order universal quantification of f, which current SMT solvers have little or no direct support for. In previous work [23], we presented two methods for tackling this problem that build on existing capabilities of SMT solvers:

1. If all occurrences of f in $P[f, \vec{x}]$ are of the form $f(\vec{x})$, then conjecture (1) is equivalent to a *first-order* formula of the form $\exists \vec{x} \, \forall y \, Q[\vec{x}, y]$, where y is a first-order variable of sort σ. Such a formula can be solved by first-order quantifier instantiation techniques specialized for theory T.

2. More generally, conjecture (1) is equivalent to $\forall d \, \exists \vec{x} \, \neg P[\lambda \, \mathsf{eval}(d, \vec{x}), \vec{x}]$, where d of a datatype sort δ that encodes the term space of solutions for f, and eval is a function that acts as an evaluator for terms in T. Such a conjecture can be solved via the syntax-guided synthesis paradigm [1] by extending an SMT solver that supports T and inductive datatypes [21].[1]

The advantage of the first method is that is highly efficient, while the advantage of the second method is that it covers a wider class of synthesis conjectures and generally produces more concise solutions. We present the current challenges in the former of these methods in this paper.

2 First-Order Quantifier Instantiation for Single Invocation Properties

Previous work [23] has shown that SMT solvers with for support quantifier instantiation already have many of the functionalities required for synthesis. In particular, [23] describes a method to efficiently solve a common class of synthesis conjectures whose properties are *single-invocation*. The conjecture $P[f, \vec{x}]$ is is *single-invocation* if it is equivalent to a quantifier-free formula of the form $Q[\vec{x}, f(\vec{x})]$ where f does not occur in $Q[x, y]$. Single invocation properties are of interest to software synthesis applications, where properties of a function-to-synthesize are often stated as a relation between the return value of that function and its inputs. Synthesis conjectures whose properties are single-invocation can be reduced to a first-order satisfiability problem by noting that a second-order conjecture of the form $\exists f \, \forall \vec{x} \, Q[\vec{x}, f(\vec{x})]$ is logically equivalent to the first-order formula:

$$\forall \vec{x} \, \exists y \, Q[\vec{x}, y] \tag{2}$$

For example, the feasibility of the synthesis conjecture $\exists f \, \forall x \, f(x) > 0$ stating that the return value of $f : \mathsf{Int} \to \mathsf{Int}$ is always positive can be equivalently stated as the first-order conjecture $\forall x \, \exists y \, y > 0$, where y is a first-order variable of sort Int. Assuming that the background theory of (2) admits quantifier elimination, i.e. there exists a decision procedure for establishing its satisfiability, the feasibility of the synthesis problem can thus be decided by such a procedure. In the spirit of recent approaches for lazy quantifier elimination [18], we consider an abstract instantiation-based procedure, shown in Figure 1, for establishing the satisfiability of a first-order quantified formula $\exists \vec{x}, \forall \vec{y} \, Q[\vec{x}, \vec{y}]$ in theory T.

[1] For more details, see Section 4 of [23].

Given input $\exists \vec{x} \forall \vec{y} Q[\vec{x}, \vec{y}]$.

Let $\Gamma := \emptyset$, and $\vec{k}$ and $\vec{e}$ be tuples of fresh constants of the same type as $\vec{x}$ and $\vec{y}$.

Repeat

If Γ is T-unsatisfiable, then return "unsat".

If $\Gamma' = \Gamma \cup \{\neg Q[\vec{k}, \vec{e}]\}$ is T-unsatisfiable, then return "sat".

Otherwise,

Let $\mathcal{I}$ be a model of T and Γ' and let $\vec{t} = \mathcal{S}_T(\mathcal{I}, \Gamma')$.

$\Gamma := \Gamma \cup \{Q[\vec{k}, \vec{t}]\}$.

Figure 1: An abstract procedure for determining the T-satisfiability of $\exists \vec{x} \forall \vec{y} Q[\vec{x}, \vec{y}]$, parameterized by a selection function $\mathcal{S}_T$.

The procedure introduces a set of constants $\vec{k}$ and $\vec{e}$ of the same type as $\vec{x}$ and $\vec{y}$, and maintains an evolving set Γ of ground instances of Q. It terminates when either Γ is unsatisfiable, in which case the input is unsatisfiable, or when $\Gamma \cup \{\neg Q[\vec{k}, \vec{e}]\}$ is T-unsatisfiable. In the latter case, Γ is satisfiable and the entailment $\Gamma \models \forall \vec{y} Q[\vec{k}, \vec{y}]$ holds, which together imply that the input is satisfiable. Both satisfiability checks in this loop are accomplished by an SMT solver with support for quantifier-free reasoning. If neither of these conditions hold, the procedure selects a tuple of terms $\vec{t}$ returned by sub-procedure $\mathcal{S}_T$, which is based on the current model $\mathcal{I}$ for Γ and $\neg Q[\vec{k}, \vec{e}]$. It subsequently adds the instance $\{Q[\vec{k}, \vec{t}]\}$ to Γ and the procedure repeats.

We refer to $\mathcal{I}$ as a *candidate model*, and $\mathcal{S}_T$ as the *selection function* of the instantiation procedure.[2] The efficiency of the procedure relies on having a good selection function, namely one that leads to a satisfiable or unsatisfiable response using a small number of instances. The termination of the procedure is guaranteed if its corresponding selection function is *finite*, in that it only returns tuples of terms taken from a finite set, and *monotonic*, in that it returns a unique tuple of terms on each successive instance it is invoked. It can be shown that a selection function for quantified formulas over linear arithmetic can be devised that meets these criteria [24], and provides an efficient approach for simulating classic approaches for quantifier elimination, e.g. [12, 17].

We may obtain solutions for f in the conjecture $\exists f \forall \vec{x} Q[\vec{x}, f(\vec{x})]$ by using this procedure to obtain a proof of *unsatisfiability* on the negated form of (2):

$$\exists \vec{x} \forall y \neg Q[\vec{x}, y] \tag{3}$$

Say the procedure finds that $\{\neg Q[\vec{k}, t_1], \ldots, \neg Q[\vec{k}, t_n]\}$ is T-unsatisfiable. Then the function:

$$\lambda \vec{x}.\, \mathsf{ite}(Q[\vec{x}, t_1], t_1, \mathsf{ite}(Q[\vec{x}, t_2], t_2, \cdots \mathsf{ite}(Q[\vec{x}, t_{n-1}], t_{n-1}, t_n) \cdots)) \tag{4}$$

is a solution for f in the synthesis conjecture $\exists f \forall \vec{x} Q[\vec{x}, f(\vec{x})]$. To see the intuition, the function (4) returns t_1 when our specification holds for t_1. It returns t_2 when it holds for t_2, and so on. The specification holds for t_n since the procedure has discovered that $\neg Q[\vec{x}, t_1] \wedge \ldots \wedge \neg Q[\vec{x}, t_{n-1}]$ entails $Q[\vec{x}, t_n]$.

[2] For details on the formal requirements of a selection function, see [24].

The above technique provides an efficient end-to-end method for solving the synthesis problem in the restricted case when the property in question is single-invocation. It is often orders of magnitude faster than enumerative syntax-guided techniques, as reported in [23] and corroborated by the 2015 edition of the syntax-guided synthesis competition [2], where its implementation in the SMT solver CVC4 won both the general and conditional linear arithmetic tracks.

2.1 Challenge: Quantifier Instantiation for New Theories

A number of instantiation-based approaches in the style of Figure 1 have been developed for the theory of linear arithmetic [6, 14, 24]. While the behavior for procedures for quantified linear arithmetic is fairly well understood, quantifier instantiation for other theories, notably for bit-vectors, remains a challenge. Current methods for quantified bit-vectors construct streams of instantiations based on candidate models [27], and use aggressive rewriting techniques to increase the likelihood that the problem can be solved as a consequence of preprocessing. The limitation of current procedures is that they are often unable to construct useful symbolic instantiations required for deriving short refutations. Consider a synthesis conjecture whose first-order formulation is:

$$\exists x \, \forall y \, \mathsf{bvshl}(y, \#\mathsf{x0001}) \not\approx \mathsf{bvand}(x, \#\mathsf{xFFFE}) \tag{5}$$

where x and y are bit-vectors of length 32. The body of this formula says that shifting y left by one bit is not equal to the taking the bit-wise conjunction of x with $\#\mathsf{xFFFE}$, which is equivalent to setting the least significant bit of x to 0. This formula can be proven unsatisfiable with a single instantiation $y \mapsto \mathsf{bvshr}(x, \#\mathsf{x0001})$, in other words the result of shifting x right by one bit. However, most current SMT solvers are unable to find this instantiation, and will instead enumerate a stream of *value* instantiations of the form $y \mapsto \#\mathsf{x0000}$, $y \mapsto \#\mathsf{x0001}$, and so on.

To address this challenge, one can develop techniques for algebraic reasoning in the context of quantifier instantiation, or use enumerative techniques for synthesizing useful quantifier instantiations. Recent work by Preiner et al [19] uses a syntax-guided approach for constructing instantiations for quantified bit-vector formulas. An approach is used for constructing Skolem functions for QBF formulas by Rabe et al [20] that is also similar in spirit.

A unique challenge to solving quantified bit-vectors is the wide array of logical subdomains in its signature. This signature includes bit-precise versions of arithmetic operators, making the insights from a procedure for quantified linear arithmetic applicable. A number of operators in bit-vectors, such as those for bit shifting above, are (fully or partially) invertible, making an algebraic procedure for finding symbolic instantiations like the one above example feasible.

Beyond bit-vectors, a number of SMT solvers have been extended with theories outside the standard canon of traditional theories, such as unbounded strings and regular expressions in SMT [16], finite sets [4], and floating points [10]. In parlance of Figure 1, devising selection functions for each of these theories remains an open challenge.

2.2 Challenge: More General Properties

The methods mentioned in this section so far are limited to single invocation properties. As means of handling more general properties, it is worth investigating methods that decompose synthesis conjectures into parts that are single invocation, and parts that are not. Consider the invariant synthesis problem for transition systems, which can be expressed as a synthesis problem where f is a *predicate* of sort $\sigma_1 \times \ldots \times \sigma_n \to \mathsf{Bool}$, of the following form:

$$\exists f. \forall \vec{x} \vec{y}. \quad (\mathsf{pre}(\vec{x}) \Rightarrow f(\vec{x})) \wedge ((\mathsf{T}(\vec{x}, \vec{y}) \wedge f(\vec{x})) \Rightarrow f(\vec{y})) \wedge (f(\vec{x}) \Rightarrow \mathsf{post}(\vec{x}))$$

where pre is a formula denoting the initial states of the system, T denotes the system's transition relation, post denotes a property to be proven invariant for the system. In this setting, f is an auxiliary inductive invariant used to prove $post$. We may decompose this property into two parts, the single invocation (non-inductive) part $(\mathsf{pre}(\vec{x}) \Rightarrow f(\vec{x}) \wedge f(\vec{x}) \Rightarrow \mathsf{post}(\vec{x}))$, and the non-single invocation (inductive) part $(T(\vec{x},\vec{y}) \wedge f(\vec{x}) \Rightarrow f(\vec{y}))$. The procedure from Figure 1 can be trivially be applied to the former part, resulting in the partial solution $\lambda\vec{x}.\mathsf{post}(\vec{x})$ (alternatively $\lambda\vec{x}.\mathsf{pre}(\vec{x})$), which can in turn be seen as a *candidate* invariant for the system. If this candidate does not satisfy the overall specification, then it can then be strengthened (alternatively, weakened) based on a counterexample to inductive part of the specification. This approach is already exploited by modern model-checkers based on property-directed reachability [9], a successful algorithm for verifying the safety of transition systems. It is yet to be determined if a similar approach can be applied more generally in the context of function synthesis, that is, when the return type of f is a sort different from Bool.

2.3 Challenge: Concise Solutions

As noted in the results of [2], the procedure in this section often returns solutions for synthesis conjectures very quickly but often returns solutions that are very large, often much larger than necessary. This shortcoming is prohibitive for synthesis applications that require optimal or nearly optimal solutions, such as software synthesis. To address this challenge, one can analyze the proof of unsatisfiability of (3) to find more concise solutions. The general idea is the following. Say the proof of unsatisfiability of this formula shows $\neg Q[\vec{k},t_1],\ldots,\neg Q[\vec{k},t_n]$ entail $\bot$. Moreover, say we find that another set of formulas $\neg R_1[\vec{k},t_1],\ldots,\neg R_n[\vec{k},t_n]$ also entail $\bot$ where for each $i=1,\ldots,n$, $\neg Q[\vec{k},t_i]$ entails $\neg R_i[\vec{k},t_i]$, and R_i is smaller or otherwise better by some measure than Q. Then, by this construction:

$$\lambda\vec{x}.\,\mathsf{ite}(R_1[\vec{x},t_1],t_1,\mathsf{ite}(\cdots\,\mathsf{ite}(R_{n-1}[\vec{x},t_{n-1}],t_{n-1},t_n)\,\cdots)) \tag{6}$$

is also a solution for f in the synthesis conjecture $\exists f\,\forall\vec{x}\,Q[\vec{x},f(\vec{x})]$. A promising line of research is to develop methods for constructing $R_1,\ldots,R_n$. We conjecture that analyzing the proof of unsatisfiability of (3) emitted by an SMT solver can be directly leveraged for this task.

3 Conclusion and Future Work

We have presenting several current challenges for developing fast SMT procedures for synthesis. These procedures have the potential to benefit a number of formal applications, including model checkers that use invariant synthesis for verifying safety properties of safety-critical systems [11]. We also foresee the opportunity for SMT solvers to play a prominent role in higher-order theorem provers and model finders as well. Examples of such tools include Nunchaku, a new counterexample generator for higher-logic that follows a line of recent work [8, 22]. Other recent systems that natively support higher-order logic [5, 7] often heavily rely on automated theorem provers and SMT solvers for first-order quantifier formulas. We conjecture that synthesis algorithms can be leveraged in these tools for reasoning about arbitrary higher-order formulas, in particular by instantiating second-order quantified formulas with solutions to synthesis problems.

References

[1] 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.

[2] R. Alur, D. Fisman, R. Singh, and A. Solar-Lezama. Results and analysis of sygus-comp'15. *arXiv preprint arXiv:1602.01170*, 2016.

[3] R. Alur, A. Radhakrishna, and A. Udupa. Scaling enumerative program synthesis via divide and conquer. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 319–336, 2017.

[4] K. Bansal, A. Reynolds, C. Barrett, and C. Tinelli. A new decision procedure for finite sets and cardinality constraints in SMT. In N. Olivetti and A. Tiwari, editors, *Proceedings of the 8th International Joint Conference on Automated Reasoning*, volume 9706, pages 82–98, 2016.

[5] C. Benzmüller, L. C. Paulson, F. Theiss, and A. Fietzke. Leo-ii-a cooperative automatic theorem prover for classical higher-order logic (system description). In *International Joint Conference on Automated Reasoning*, pages 162–170. Springer Berlin Heidelberg, 2008.

[6] N. Bjørner and M. Janota. Playing with quantified satisfaction. In *20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations, LPAR 2015, Suva, Fiji, November 24-28, 2015.*, pages 15–27, 2015.

[7] J. C. Blanchette, S. Böhme, and L. C. Paulson. Extending sledgehammer with SMT solvers. *Journal of automated reasoning*, 51(1):109–128, 2013.

[8] J. C. Blanchette and T. Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In *International Conference on Interactive Theorem Proving*, pages 131–146. Springer Berlin Heidelberg, 2010.

[9] A. R. Bradley. *SAT-Based Model Checking without Unrolling*, pages 70–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[10] M. Brain, V. DSilva, A. Griggio, L. Haller, and D. Kroening. Deciding floating-point logic with abstract conflict driven clause learning. *Formal Methods in System Design*, 45(2):213–245, 2014.

[11] A. Champion, A. Mebsout, C. Sticksel, and C. Tinelli. The kind 2 model checker. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, pages 510–517, 2016.

[12] D. C. Cooper. Theorem proving in arithmetic without multiplication. In *Machine Intelligence, pages 91100*, 1972.

[13] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical report, J. ACM, 2003.

[14] B. Dutertre. Solving exists/forall problems with yices. In *Workshop on Satisfiability Modulo Theories*, 2015.

[15] Y. Ge and L. M. de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *Computer Aided Verification, 21st International Conference, CAV 2009*, pages 306–320, 2009.

[16] T. Liang, A. Reynolds, C. Tinelli, C. Barrett, and M. Deters. A DPLL(T) theory solver for a theory of strings and regular expressions. In *Computer Aided Verification - 26th International Conference, CAV 2014*, pages 646–662, 2014.

[17] R. Loos and V. Weispfenning. Applying linear quantifier elimination. *Comput. J.*, 36(5):450–462, 1993.

[18] D. Monniaux. Quantifier elimination by lazy model enumeration. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pages 585–599, 2010.

[19] M. Preiner, A. Niemetz, and A. Biere. Counterexample-guided model synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 264–280, 2017.

[20] M. N. Rabe and S. A. Seshia. Incremental determinization. In *Theory and Applications of Satisfiability Testing - SAT*, pages 375–392, 2016.

[21] A. Reynolds and J. C. Blanchette. A decision procedure for (co) datatypes in SMT solvers. In *International Conference on Automated Deduction*, pages 197–213. Springer International Publishing, 2015.

[22] A. Reynolds, J. C. Blanchette, S. Cruanes, and C. Tinelli. Model finding for recursive functions in SMT. In *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, pages 133–151, 2016.

[23] A. Reynolds, M. Deters, V. Kuncak, C. Tinelli, and C. W. Barrett. Counterexample-guided quantifier instantiation for synthesis in SMT. In *Computer Aided Verification - 27th International Conference, CAV 2015, San*

Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II, pages 198–216, 2015.

[24] A. Reynolds, T. King, and V. Kuncak. Solving quantified linear arithmetic by counterexample-guided instantiation. In *Formal Methods in System Design (to appear)*, 2017.

[25] A. Reynolds, C. Tinelli, A. Goel, S. Krstić, M. Deters, and C. Barrett. Quantifier instantiation techniques for finite model finding in SMT. In M. P. Bonacina, editor, *Proceedings of the 24th International Conference on Automated Deduction (Lake Placid, NY, USA)*, volume 7898 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 2013.

[26] A. Solar-Lezama. Program sketching. *STTT*, 15(5-6):475–495, 2013.

[27] C. M. Wintersteiger, Y. Hamadi, and L. De Moura. Efficiently solving quantified bit-vector formulas. *Formal Methods in System Design*, 42(1):3–23, 2013.