Towards Learning Infinite SMT Models (Work in Progress)

1st Mikoláš Janota
Czech Technical University in Prague
Prague, Czech Republic
https://orcid.org/0000-0003-3487-784X

2nd Bartosz Piotrowski
Czech Technical University in Prague
Prague, Czech Republic
https://orcid.org/0000-0002-1699-018X

3nd Karel Chvalovský
Czech Technical University in Prague
Prague, Czech Republic
https://orcid.org/0000-0002-0541-3889

***Abstract*—This short paper proposes to learn models of satisfiability modulo theories (SMT) formulas during solving. Specifically, we focus on infinite models for problems in the logic of linear arithmetic with uninterpreted functions (UFLIA). The constructed models are piecewise linear. Such models are useful for satisfiable problems but also provide an alternative driver for model-based quantifier instantiation (MBQI).**

***Index Terms*—SMT, infinite models, piecewise linear**

I. Introduction

Formulas with quantifiers remain a major challenge for *satisfiability modulo theories* (SMT) [1] solvers. This is especially true when quantifiers are combined with uninterpreted functions. Predominantly, SMT solvers tackle quantifiers by gradually instantiating them by ground terms. Such terms may be chosen by a variety of techniques, which themselves have further degrees of freedom that are addressed by specific heuristics in the concrete implementations of the solvers.

One of the oldest techniques is *e-matching* [2], which is mainly syntactic and does not guarantee any completeness. Refutation completeness under certain conditions can be achieved by simple *enumerative instantiation* [3], [4]. Term generation can also be further focused by syntactic guidance [5] or conflicts [6].

A quantifier instantiation technique that stands out is *model-based quantifier instantiation (MBQI)* [7], which unlike the above-mentioned enables showing a formula SAT. This involves constructing a sequence of candidate models that are checked against the formula and drive further instantiations. In their earlier work, Bradley and Manna identify the *array property* [8], [9], for which they show that a finite set of instantiations is always complete—MBQI can be seen as an instantiation generalization of this approach [10].

An interesting question arises in the context of MBQI and that is how to construct the candidate models based on the models of the current ground part (which is gradually being strengthened by further instantiations). In this paper, we propose to attempt to learn piecewise linear functions to represent the candidate models. We use simple, fast algorithms that construct such functions based on the values obtained from the ground model. In that sense, our approach is purely semantic—it ignores the syntactic form of the formula, which contrasts with the current techniques that are mainly syntactically driven [10, Sec. 5].

We implement the proposed techniques in the state-of-the-art SMT solver cvc5 [11] and report encouraging results, where number of SAT responses increases and does not incur a slowdown for UNSAT problems.

To the best of our knowledge such learning had not been attempted in the context of MBQI. Some related approaches appear in the literature, finite models have been learned in the context of finite model finding [12]. Invariants and termination conditions in the context of software verification have been constructed in an analogous fashion [13]–[15]. Another related research direction is function and program *synthesis*. Notably, Barbosa et al. [16] learn syntactically driven decision trees to construct functions—while this is in the context of the SMT solver cvc5 [11], it is separate from the main solver so for instance it is not meant for UNSAT problems. In recent work, Parsert et al. show that state-of-the-art synthesis approaches often fail when faced with SMT problems [17].

II. Learning Models in MBQI

MBQI works in iterations, where in each iteration there is a candidate model $\mathcal{M}$ satisfying the ground part ϕ_g. Let us assume that the formula being solved is of the form $(\forall \mathbf{x}\, \phi)$ for some quantifier-free ϕ and a vector of variables $\mathbf{x}$. A sub-SMT call is issued to check weather $\mathcal{M}$ satisfies $(\exists \mathbf{x}\, \neg\phi)$. If it does not, the formula is satisfied and the model $\mathcal{M}$ is also a model of the original formula. If it is satisfied by some vector $\mathbf{c}$, then the ground formula ϕ_g is strengthened by an instantiation of $(\forall \mathbf{x}\, \phi)$ based on $\mathbf{c}$—for simplicity one may imagine that $\mathbf{c}$ is plugged into $\mathbf{x}$. Hence, the process either terminates by finding a model of the original formula, or when the ground part becomes unsatisfiable (meaning that the original formula is also unsatisfiable), or goes on indefinitely. This contrasts with other quantifier instantiation techniques that are only able to prove unsatisfiability.

Example 2.1: The following is a possible (non-terminating) run of MBQI on the formula $(\forall x : \mathbb{Z}\, f(x) > x)$. ITE

The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project *POSTMAN* no. LL1902. This article is part of the *RICAIP* project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306.

Algorithm 1: Greedy function construction

Function `Build` (P)
input : list of function points $P \subset \mathbb{Z}^n \times \mathbb{Z}$
output : function $\mathbb{Z}^n \mapsto \mathbb{Z}$

$C \leftarrow \{\}$ `// set of constraints`
$S \leftarrow \{\}$ `// set of covered points`
while $P \neq \emptyset$ **do**
  $\mathbf{a}, v \leftarrow \text{head}(P)$
  $C' \leftarrow C \cup \{\mathbf{a}^T\mathbf{y} + c = v\}$
  if $\mathit{SAT}(C')$ **then**
    $C \leftarrow C'$
    $P \leftarrow \text{tail}(P)$
    $S \leftarrow S \cup \{(\mathbf{a}, v)\}$
  else
    break
$\mathbf{s}, c \leftarrow$ `Solve`(C)
segment $\leftarrow \lambda\mathbf{x}.\, \mathbf{s}^T\mathbf{x} + c$
if $P = \emptyset$ **then** **return** segment
else **return** ITE(`Split`(S, P), segment, `Build`(P))

Algorithm 2: Recursive predicate splitting

Function `Build` (P)
input : positive and negative points $P \subset \mathbb{Z}^n \times \mathbb{B}$
output : predicate $\mathbb{Z}^n \mapsto \mathbb{B}$

if P *all negative* **then return** $\lambda\mathbf{x}$.false
if P *all positive* **then return** $\lambda\mathbf{x}$.true
$C \leftarrow \{\}$ `// set of constraints`
foreach $(\mathbf{a}, b) \in P$ **do**
  if b **then** $C' \leftarrow C \cup \{\mathbf{a}^T\mathbf{y} \geq c\}$
  else $C' \leftarrow C \cup \{\mathbf{a}^T\mathbf{y} < c\}$
  if $\mathit{SAT}(C')$ **then** $C \leftarrow C'$
$\mathbf{s}, c \leftarrow$ `Solve`(C)
$P^+ \leftarrow$ `Build`$(\{(\mathbf{a}, b) \in P \mid \mathbf{s}^T\mathbf{a} \geq c\})$
$P^- \leftarrow$ `Build`$(\{(\mathbf{a}, b) \in P \mid \mathbf{s}^T\mathbf{a} < c\})$
return $\lambda\mathbf{x}.\,\text{ITE}(\mathbf{s}^T\mathbf{x} \geq c, P^+, P^-)$

abbreviation below signifies an *if-then-else* expression.

iteration	ϕ_g	$f(x)$	c
0	true	0	0
1	$f(0) > 0$	1	1
2	$f(0) > 0 \wedge f(1) > 1$	ITE$(x = 0, 1, 2)$	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Example 2.1 shows that for satisfiable formulas MBQI can easily diverge, even though it could terminate quickly if it guessed a different candidate model, for instance $f(x) = x+1$. Neither Z3 nor cvc5 solves this formula. In this work, we focus specifically on simple model candidates that can be composed of linear functions.

Here we focus on problems from linear integer arithmetic with uninterpreted functions (UFLIA). In each iteration of the MBQI loop, we assume that we are given for each n-ary function or predicate f finitely many points $(\mathbf{a}, v)$, where $\mathbf{a}$ is a vector of n values for the arguments and v is the value of f for these arguments, i.e., $f(\mathbf{a}) = v$. The value is either an integer (for functions) or Boolean (for predicates). These function points appear naturally in MBQI as a model of the ground part. So for instance, in iteration 2 of Example 2.1, the model for the ground part might assign $f(0) = 2$ and $f(1) = 3$, allowing us to propose $f(x) = x + 2$.

Currently, we consider each function or predicate in the formula separately. Hence, the objective is fitting a piecewise linear function to these points, which is essentially a synthesis task. Here we also need to take into account that the points are in the space of integers and they must be matched perfectly, not just approximately as in some methods used in statistical learning, *cf.* [18]. Further, the process needs to be efficient since it may be invoked many times during individual iterations of MBQI.

A straightforward approach to synthesizing a piecewise linear function is a greedy one, where we first sort the points according to some criterion (e.g., lexicographically) and then try to greedily connect adjacent points into a single hyperplane. This approach is outlined in Algorithm 1. The points are organized in a list and to check if the current point fits onto the hyperplane under construction, we add a corresponding constraint to the set of equations C. These equations are of the form $\mathbf{a}^T\mathbf{y} + c = v$, where $\mathbf{a} \in \mathbb{Z}^n, c \in \mathbb{Z}$ and $\mathbf{y}$ is a vector of integer variables. These are linear Diophantine equations solvable in polynomial time [19, pgs. 343–345] [20].

Once the set of constraints C becomes unsatisfiable, a new segment (hyperplane) needs to be started. In order to construct an SMT term, we use an if-then-else (ITE) expression. The splitting condition must be such that the points already covered and the points yet to be covered become disjoint under this condition. How exactly this split is done is represented by the function `Split` in the pseudocode. In our implementation, we use a lexicographic order on P, which lets us also easily split the points as follows. If the last covered point is $a_0, a_1, \ldots$ and the point yet to be covered is $a'_0, a'_1, \ldots$, the condition is

$$x_0 < a'_0 \vee (x_0 = a'_0 \wedge x_1 < a'_1) \vee \ldots$$

Additionally, this condition is simplified so that we consider x_i only if $a_j = a'_j$ for $j < i$. If all the given points already fit on the hyperplane under construction, no splitting is needed.

To treat predicates, rather than equations, as a primitive we use inequalities of the form $\mathbf{s}^T\mathbf{a} \geq c$, $\mathbf{s} \in \mathbb{Z}^n, c \in \mathbb{Z}$. An analogous greedy algorithm could be used to split a list of points into segments. However, only very simple predicates can be learned by this approach, since each segment is only able to separate points by a single hyperplane. So for instance equality cannot be learned.

An alternative is to use decision trees but here the question is what should be the predicates used in the internal nodes of the tree? In statistical machine learning, decision trees split on the value of a single feature (variable) and this seems to be too limiting since this enables capturing only limited interactions between variables—again, equality would not be learnable. Hence, we apply a hybrid approach where the points are split into two parts by *some* hyperplane and the rest is classified recursively. Effectively, we are building a decision tree where each branch corresponds to a convex polyhedron and each polyhedron should only contain points of one value.

This approach is outlined in Algorithm 2, which looks greedily for a hyperplane splitting the encountered points into positive and negative. Since this hyperplane might not split the points perfectly, each "half" is further refined recursively.

For the Algorithm 2 to terminate, the chosen hyperplane must split at least one positive and one negative point. Further, the algorithm is sensitive to the order in which the points are added to the constraints C. We use a simple heuristic for this order. We first focus on a pair of points with different values (one false, one true). Since there may be many such pairs, we first sort by lexicographic order and pick a pair of adjacent points (p_i, p_{i+1}) with different values. There still may be multiple such pairs and we pick such pair that maximizes information gain [21] by looking at the points left and right of the pair—i.e., by looking at the subsets $\{p_1, \ldots, p_i\}$ and $\{p_{i+1}, \ldots, p_n\}$, as if they were split by the hyperplane currently constructed (even though this might not eventually be true, since we only guarantee that p_i and p_{i+1} are split). Other points are added into the constraints by going first right and then going left from the splitting pair—this order was chosen arbitrarily.

Compared to Algorithm 2, the current implementation in fact stops the greedy search on the first UNSAT response from the sub-solver. The reason is that we want to avoid inefficiencies in the sub-solver, which is currently used in an incremental setting without push and pops.

Fig. 1 visualizes how equality can be found using this algorithm. The example shows the evolution of models for the formula $(\forall xy : \mathbb{Z}\, R(x, y) \Rightarrow x = y) \wedge (\forall xy : \mathbb{Z}\, x = y \Rightarrow R(x, y))$, which unambiguously defines R as the equality. Also, this instance is not solved by neither Z3 nor cvc5.

In the final iteration of the MBQI loop, the learning algorithm is given the positive points $\{(0, 0), (1, 1), (-1, -1)\}$ and the negative points $\{(-1, 0), (0, 1), (1, 0), (1, 2)\}$. The recursive algorithm learns the term $\text{ITE}(x - y \geq 0, -x + y \geq 0, \text{false})$, which can be seen as the intersection of $x \geq y$ and $y \geq x$, as expected.

III. Experiments

The proposed algorithms were implemented in cvc5 [11] in the main branch. The experiments were run with the time limit of 30 s. We evaluate on two sets of benchmarks. The first set was obtained by taking SMT-LIB benchmarks from UFLIA that do not contain any uninterpreted sorts; these tend to be unsatisfiable. The second set is obtained by taking fragments of UFLIA benchmarks—we describe the process in the following subsection. The solver cvc5 is run with e-matching turned off, i.e. MBQI only. We compare two versions: one where predicates are synthesized by Algorithm 1 (*non-smart*) and Algorithm 2 (*smart*).

solver	solved: UNSAT
standard MBQI	3831
ours non-smart MBQI	3831
ours smart MBQI	3835
Z3	5809

TABLE I: Problems solved within 30 s time limit by four solvers in a benchmark being a set of these UFLIA problems that does not have sort declarations. There were no SAT results in this benchmark. The total number of problems is 6288.

solver	solved: SAT	solved: UNSAT	solved: total
standard MBQI	18843	7863	26706
ours non-smart MBQI	29456	7863	37319
ours smart MBQI	31977	7863	39840
Z3	28380	7482	35862

TABLE II: Problems solved within 30 s time limit by four solvers in a benchmark based on UFLIA problems modified to make most of the problems SAT by removing some assertions, and where all declared sorts were substituted with `Int`. The total number of problems is 69692.

The results for these two sets of problems are summarized in Tables I and II, respectively. The results indicate that on satisfiable instances, our semantic-guided syntactic outperforms the syntactic approach of Z3. Our implementation does not incur any slowdown on the unsatisfiable instances. In fact, a handful of instances are solved on top of the default MBQI.

A. Generation of satisfiable problems

We consider a simple technique to generate interesting satisfiable fragments from a given SMT formula—understood as a conjunction of assertions. The idea is to consider some parameter k and extract fragments that contain k uninterpreted functions. Given an SMT formula, we choose k uninterpreted function symbols that appear in the formula and filter out the conjuncts that are only weakly related to these k symbols. Let us describe the process for $k = 2$. Consider an SMT formula ϕ and two uninterpreted function symbols f and g that occur in ϕ. Consider subformula ψ in ϕ (one of the conjuncts) of the form $(\forall \mathbf{x}\, \psi')$, where $\mathbf{x}$ is an arbitrary set of variables. We will say that ψ is *in the* f, g *fragment of* ϕ if it contains at least one f or g and no other uninterpreted functions; there is no limitation on constants. *The global* f, g *fragment of* ϕ is defined as the conjunction of all the f, g fragments in ϕ. We remark that in the current implementation, we only consider subformulas that are denoted by the users as separate assertions—this could be relaxed.

Since the current implementation only supports generation of functions and predicates on integers, we only consider UFLIA problems and replace all of interpreted sorts by `Int`.

IV. Summary

In this short paper we propose algorithms for the construction of piecewise-linear model candidates in the context of model-based quantifier instantiation (MBQI), which is a

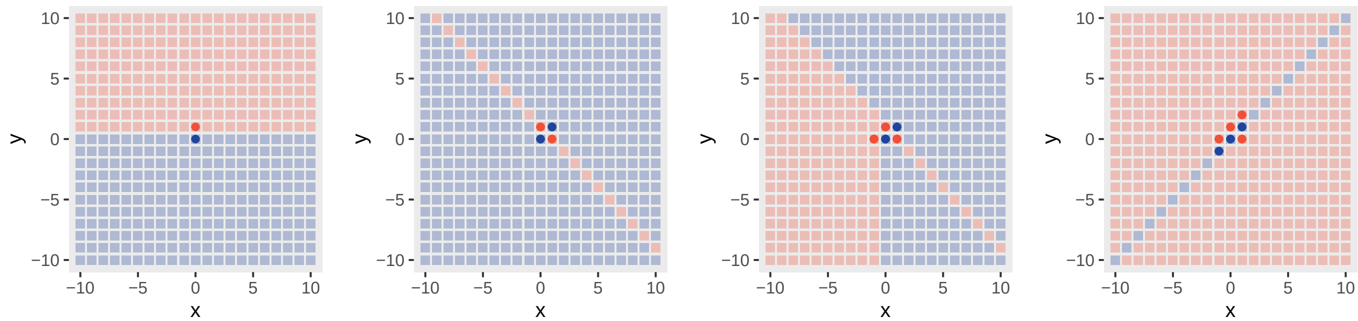

Fig. 1: Four snapshots of the iterative process of finding an infinite model for a binary relation R, as described in Algorithm 2. The relation R is constrained by two assertions: $R(x, y) \Rightarrow x = y$ and $x = y \Rightarrow R(x, y)$. Yellow color signifies *false* and blue color signifies *true*. Dots in the center are points with boolean values assigned by the solver. Rectangles in the background depict the current relation assigned to R. The last figure shows the state in which the algorithm found the desired relation.

powerful instantiation technique for solving SMT problems with quantification. In essence, this means learning an *infinite* model based on *finite* information. The experimental evaluation shows that many new satisfiable problems can be solved by the proposed approach and at the same time it does not slow down the solver for unsatisfiable problems. In the future, we would like to explore closer collaboration with the sub-solvers and the main algorithm, *cf.* [22].

Acknowledgments

We would like to thank Chad Brown for numerous discussions on the topic and Andrew Reynolds for giving useful pointers regarding the codebase of cvc5.

References

[1] C. Barrett and C. Tinelli, “Satisfiability modulo theories,” in *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[2] D. Detlefs, G. Nelson, and J. B. Saxe, “Simplify: A theorem prover for program checking,” *J. ACM*, vol. 52, no. 3, pp. 365–473, 2005. [Online]. Available: https://doi.org/10.1145/1066100.1066102

[3] A. Reynolds, H. Barbosa, and P. Fontaine, “Revisiting enumerative instantiation,” in *Tools and Algorithms for the Construction and Analysis of Systems*, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-89963-3_7

[4] M. Janota, H. Barbosa, P. Fontaine, and A. Reynolds, “Fair and adventurous enumeration of quantifier instantiations,” in *Formal Methods in Computer-Aided Design*, 2021.

[5] A. Niemetz, M. Preiner, A. Reynolds, C. W. Barrett, and C. Tinelli, “Syntax-guided quantifier instantiation,” in *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. Springer, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-72013-1_8

[6] A. Reynolds, C. Tinelli, and L. M. de Moura, “Finding conflicting instances of quantified formulas in SMT,” in *Formal Methods in Computer-Aided Design, FMCAD*. IEEE, 2014, pp. 195–202. [Online]. Available: https://doi.org/10.1109/FMCAD.2014.6987613

[7] Y. Ge and L. M. de Moura, “Complete instantiation for quantified formulas in satisfiabiliby modulo theories,” in *Computer Aided Verification CAV*, vol. 5643. Springer, 2009, pp. 306–320. [Online]. Available: https://doi.org/10.1007/978-3-642-02658-4_25

[8] A. R. Bradley, Z. Manna, and H. B. Sipma, “What’s decidable about arrays?” in *Verification, Model Checking, and Abstract Interpretation Conference, VMCAI*, vol. 3855. Springer, 2006, pp. 427–442. [Online]. Available: https://doi.org/10.1007/11609773_28

[9] A. R. Bradley and Z. Manna, *The calculus of computation - decision procedures with applications to verification*. Springer, 2007. [Online]. Available: https://doi.org/10.1007/978-3-540-74113-8

[10] L. M. de Moura and N. S. Bjørner, “Bugs, moles and skeletons: Symbolic reasoning for software development,” in *Automated Reasoning, 5th International Joint Conference, IJCAR*, vol. 6173. Springer, 2010, pp. 400–411. [Online]. Available: https://doi.org/10.1007/978-3-642-14203-1_34

[11] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, and Y. Zohar, “cvc5: A versatile and industrial-strength SMT solver,” in *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. Springer, 2022. [Online]. Available: https://doi.org/10.1007/978-3-030-99524-9_24

[12] M. Janota and M. Suda, “Towards smarter MACE-style model finders,” in *22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2018*, vol. 57. EasyChair, 2018, pp. 454–470. [Online]. Available: https://doi.org/10.29007/w42s

[13] A. Adjé and P. Garoche, “Automatic synthesis of piecewise linear quadratic invariants for programs,” in *Verification, Model Checking, and Abstract Interpretation, VMCAI*, vol. 8931. Springer, 2015, pp. 99–116. [Online]. Available: https://doi.org/10.1007/978-3-662-46081-8_6

[14] C. Urban, “The abstract domain of segmented ranking functions,” in *Static Analysis SAS*. Springer, 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-38856-9_5

[15] C. Urban and A. Miné, “A decision tree abstract domain for proving conditional termination,” in *Static Analysis SAS*. Springer, 2014. [Online]. Available: https://doi.org/10.1007/978-3-319-10936-7_19

[16] H. Barbosa, A. Reynolds, D. Larraz, and C. Tinelli, “Extending enumerative function synthesis via SMT-driven classification,” in *2019 Formal Methods in Computer Aided Design, FMCAD*. IEEE, 2019, pp. 212–220. [Online]. Available: https://doi.org/10.23919/FMCAD.2019.8894267

[17] J. Parsert, C. Brown, M. Janota, and C. Kaliszyk, “Experiments on infinite model finding in SMT solving,” in *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*. EasyChair, 2023. [Online]. Available: https://easychair.org/publications/paper/TLbl

[18] N. Landwehr, M. Hall, and E. Frank, “Logistic model trees,” *Machine Learning*, vol. 59, no. 1-2, pp. 161–205, May 2005. [Online]. Available: https://doi.org/10.1007/s10994-005-0466-3

[19] D. E. Knuth, *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998.

[20] A. Griggio, “A practical approach to satisfiability modulo linear integer arithmetic,” *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 8, no. 1-2, pp. 1–27, 2012.

[21] J. R. Quinlan, “Induction of decision trees,” *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[22] A. Abate, H. Barbosa, C. W. Barrett, C. David, P. Kesseli, D. Kroening, E. Polgreen, A. Reynolds, and C. Tinelli, “Synthesising programs with non-trivial constants,” *J. Autom. Reason.*, vol. 67, no. 2, p. 19, 2023. [Online]. Available: https://doi.org/10.1007/s10817-023-09664-4