Quantifier Instantiations: To Mimic or To Revolt?

Jan Jakubův, Mikoláš Janota

Czech Technical University in Prague, Prague, Czech Republic

Abstract

Quantified formulas pose a significant challenge for Satisfiability Modulo Theories (SMT) solvers due to their inherent undecidability. Existing instantiation techniques, such as e-matching, syntax-guided, model-based, conflict-based, and enumerative methods, often complement each other. This paper introduces a novel instantiation approach that dynamically learns from these techniques during solving. By treating observed instantiations as samples from a latent language, we use probabilistic context-free grammars to generate new, similar terms. Our method not only mimics successful past instantiations but also explores diversity by optionally inverting learned term probabilities, aiming to balance exploitation and exploration in quantifier reasoning.

Keywords

Satisfiability Modulo Theories, Quantifier Instantiation, Probabilistic Term Generation

1. Introduction

Solving formulas involving quantifiers is notoriously hard due to the undecidabilty of the problem. Satisfiability Modulo Theories (SMT) solvers tackle quantifiers by instantiating quantified variables by ground terms. A series of techniques exists that attempt to find instantiations that will most likely lead to a contradiction. Prominently, the techniques used in modern SMT solvers are syntax-driven (*e-matching* [1] or *syntax-guided instantiation* [2]), semantic-driven (*model-based* [3, 4]), *conflict-based* [5], and *enumerative instantiation* [6, 7]. These techniques tend to give highly complementary results.

In this paper we propose a quantifier instantiation technique that seeks inspiration in the other techniques on the fly—while solving the formula. The idea is as follows. Observe which instantiations were made by other techniques and then, attempt to make similar ones. Now, how to find similar instantiations? We look for inspiration in *probabilistic context free grammars* [8]. We imagine the instantiations made so far as a language, where each ground term is a sentence in the language. Since we do not have an explicit definition of the language, we take our observations as samples from it.

Figure 1 illustrates the overall idea of the approach. Instantiation is made by several instantiation modules, e.g. e-matching. Each instantiation module produces some instantiations that are sent to the ground solver. In our approach, these are also intercepted and collected in the ProbGen generator, which also produces its own instantiations sent to the ground solver.

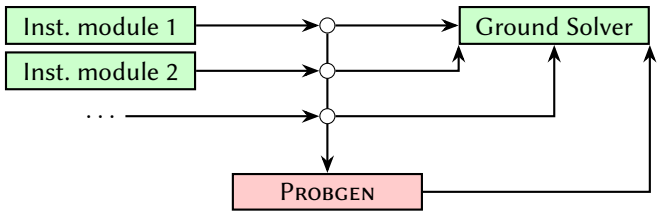

Figure 1: Schematic depiction of the overall idea, where Probgen is our contribution

The idea of probabilistic grammars can be simplified even further. Let's say that the constant c appears frequently under the function symbol f, then $f(c)$ should also be a likely term in our language.

SMT 2025: 23rd International Workshop on Satisfiability Modulo Theories, August 10–11, 2025, Glasgow, UK
https://people.ciirc.cvut.cz/~jakubja5/ (J. Jakubův); https://people.ciirc.cvut.cz/~janotmik (M. Janota)
0000-0002-8848-5537 (J. Jakubův); 0000-0003-3487-784X (M. Janota)

Algorithm 1 Make Term

Input: τ - range type, d - term depth, initially 0
Output: term of type τ
Using: $\textsc{Symbols}(\tau)$ - symbols constructing terms of type τ
$\textsc{Depth}$ - maximum allowed term depth
$\textsc{Pick}(S)$ - pick one symbol out of S according to the selection method
$\textsc{Arity}(s)$ - the arity of symbols s

1: **function** $\textsc{MakeTerm}(\tau, d = 0)$
2: $S \leftarrow \textsc{Symbols}(\tau)$
3: **if** $d \geq \textsc{Depth}$ **then**
4: $S \leftarrow \{s \in S \mid s \text{ is a constant}\}$
5: $s \leftarrow \textsc{Pick}(S)$
6: **if** s is a constant **then**
7: **return** s
8: **else**
9: **for all** $0 < i \leq \textsc{Arity}(s)$ **do**
10: $t_i \leftarrow \textsc{MakeTerm}(\tau_i, d + 1)$ **where** τ_i is the type of the i-th argument of s
11: **return** $s(t_1, \ldots, t_n)$

However, this suggests a different question. If the new terms are generated according to the ones that were already used in the past, will they be useful? Should we not rather do the opposite, i.e., *generate different terms*? Effectively, this means *inverting* the probabilities when generating new terms for instantiation.

2. Preliminaries

Basic understanding of SMT [9] is assumed. In special cases, such as linear arithmetic, decision procedures exist for theories with quantifiers [10, 11, 12, 13]. In general, however, presence of quantifiers can easily make the problem undecidable and in SMT, quantifier instantiations are used to tackle this problem. A quantified subformula $\forall \bar{x}.\phi$, with a vectors of variables $\bar{x}$ is abstracted as a proposition Q and instantiations are realized as implications of the form $Q \rightarrow \phi[\bar{x} \mapsto \bar{t}]$, where t is a vector of ground terms. This implication is called a *lemma*. A plethora of instantiation techniques appear in the literature. These typically exhibit complementary behavior and do not interact directly with one another, although indirect interaction through a shared solver state can occur.

Probabilistic grammars [14, 8], extend traditional formal grammars by associating probabilities with their production rules. This probabilistic framework allows for modeling uncertainty and variation in natural language and other structured data. In a probabilistic context-free grammar (PCFG), each rule of the form $A \rightarrow \alpha$ is assigned a probability $P(A \rightarrow \alpha)$, such that the sum of probabilities of all rules with the same left-hand nonterminal A is 1. These models are particularly useful in computational linguistics and natural language processing tasks, where ambiguity and multiple possible interpretations are common. PCFG have been used in autoformalization tasks [15].

3. Term Generation

Full-fledged learning of PCFG is likely to be to expensive for our purposes and we take a Markov-like approach instead. New ground terms are generated recursively, as a tree, and the probability of a symbol s being generated in a node n is determined by the frequency of s in the instantiations so far.

Terms are generated recursively in a straightforward fashion with fixed maximum depth using function $\textsc{MakeTerm}$ in Algorithm 1. The function $\textsc{MakeTerm}(\tau)$ generates a term of type τ by selecting a term head symbols and recursively generating argument terms, possibly resorting to constants to

limit the depth of generated terms. The head symbol is always picked from the set $\textsc{Symbols}(\tau)$ which contains all symbols of type τ encountered in the terms generated by other instantiation modules. The maximum allowed term depth is controlled by parameter Depth. With $\textsc{Depth} = 0$, the algorithm generates constants only.

The key point of the algorithm is the selection of the term head symbol using the function Pick. Naturally, there is a lot of flexibility in this choice, and that is where our statistical approach comes into play. We implement three different selection methods. The first one, denoted random, selects a symbol uniformly at random, independent of any symbol statistics. The next method, weights, maintains a statistic for each symbol s, counting the number of times s occurs in terms generated by other instantiation techniques. These counts are interpreted as probabilistic weights, and the head symbol is then sampled from a categorical distribution derived from them. In practice, this means treating the weights as proportional intervals on a number line, generating a random number in the range from 0 to the total weight sum, and selecting the symbol corresponding to the interval in which the number falls. The third and final selection method we experiment with is paths, which extends weights by taking term positions into account. It maintains separate weight vectors for each term position, where a position is determined by the list of symbols occurring above the term in the syntax tree. For example, the position of both a and b in $f(g(a, b))$ is given by the path (f, g). Since the weight vectors for paths can be quite sparse and contain only few symbols of the given type, we additionally consider all other compatible symbols with the default weight 1 at each position.

The above symbol selection methods weights and paths generates terms mimicking terms generated by other instantiation modules. In order to introduce diversity and generate different terms, we introduce additional parameter Flip. Its value is the probability under which the weights should be inverted, setting all weights to $1/w$ instead of w. This probability is applied in every call to function Pick. Hence different calls to Pick within one call to MakeTerm can work with different weights.

The final parameter we introduce to our probabilistic instantiation module concerns when the module is activated. In cvc5, several *effort* levels are defined for instantiation modules. These modules are tried sequentially, with the effort level increasing if no lemma is produced at the current level. The currently implemented effort levels in cvc5, in order of increasing effort, are *conflict*, *standard*, *model*, and *last call*. Conflict-based instantiation modules are tried first (*conflict*), followed by heuristic instantiation modules (*standard*, e.g., e-matching), then model-based techniques (*model*), and finally full effort instantiation (*last call*) are launched.

We follow the behavior of the enumerative instantiation modules and introduce a parameter Effort, which supports two values: lastcall and interleave. With lastcall, our probabilistic instantiation module is executed only at the last call effort level, that is, when no module has produced a lemma at a lower effort. With interleave, instantiations are performed already at the standard effort level, meaning the module runs more frequently than with lastcall.

4. Experimental Evaluation

We evaluate[1] our approach on a benchmark from SMT-LIB [16, 17], namely on $8,024$ problems from the `2019-Preiner` directory of the UFNIA logic. This benchmark was chosen based on a preliminary experiment on a larger subset of SMT-LIB since our methods seemed to perform well therein. We perform an extensive grid search for various parameters of the probabilistic instantiation module, namely all of the following combinations.

$$\begin{aligned} \textsc{Effort} &\in \{\textsc{lastcall}, \textsc{interleave}\} \\ \textsc{Pick} &\in \{\textsc{random}, \textsc{weights}, \textsc{paths}\} \\ \textsc{Depth} &\in \{0, 1, 2, 3, 4\} \\ \textsc{Flip} &\in \{0.0, 0.2, 0.5, 0.8, 1.0\} \end{aligned}$$

[1] On a server with two AMD EPYC 7513 32-Core processors @ 3680 MHz and with 514 GB RAM.

Effort	lastcall						interleave					
Pick	random		weights		paths		random		weights		paths	
Depth	*total*	*ref (+/-)*	*total*	*ref (+/-)*	*total*	*ref (+/-)*	*total*	*ref (+/-)*	*total*	*ref (+/-)*	*total*	*ref (+/-)*
0	**3523**	**+499**/-53	**3497**	**+477/-57**	**3445**	**+427**/-59	**3575**	**+571/-73**	**3570**	**+571/-78**	**3498**	**+497/-76**
1	3293	+270/-54	3245	+230/-62	3273	+247/-51	3309	+321/-89	3284	+300/-93	3327	+349/-99
2	3221	+195/-51	3231	+215/-61	3220	+190/-47	3242	+259/-94	3239	+256/-94	3211	+235/-101
3	3220	+193/-**50**	3192	+182/-67	3219	+188/-**46**	3276	+290/-91	3171	+211/-117	3230	+256/-103
4	3212	+197/-62	3181	+169/-65	3215	+184/-**46**	3236	+258/-99	3161	+206/-122	3228	+255/-104

Figure 2: Grid evaluation of strategies with different Depth values (with fixed Flip $= 0.0$).

Since the parameter Flip is applicable only when Pick is either weights or paths, we obtain altogether 110 different strategies. We evaluate all of them with a 10 second time limit per strategy and problem. Our probabilistic instantiation module is launched together with the default cvc5 setting which uses e-matching and conflict-based quantifier instantiation (cbqi). The instantiation terms introduced by these two default modules are intercepted and used to collect symbol statistics utilized by our weights and paths symbol selection functions. The set of asserted quantifiers is randomly shuffled in each round. This is because we terminate the instantiation round after a first successfully generated lemma.

We always generate 20 terms for each encountered type and remove duplicates if necessary. Within one instantiation round, only one set of terms is generated for every encountered type, that is, for every type of a $\forall$-bound variable from asserted quantifiers. This gives us a non-zero probability that variables of the same type will be instantiated by the same term. A new set of terms is generated in the next instantiation round.

The results for different values of Depth with Flip fixed to 0.0 are depicted in Figure 2. This allows us to evaluate the effect of various term depths. For each strategy we present three numbers: (1) the total number of solved problems (column *total*), and (2-3) the number of solutions gained ($+$) and lost ($-$) on the baseline reference strategy (columns *ref*). As the baseline strategy we consider cvc5 with the default option setting. **The reference strategy solves 3,077 problems.** To ease orientation, the best value in each column is **highlighted** and the best values within the table are additionally underlined. We can draw several observations from Figure 2:

1. All strategies significantly outperform the reference strategy, which solves 3,077 problems.
2. In every case, increasing the maximum term depth leads to a decline in performance. The best results are achieved when instantiating with constants only (Depth $= 0$), likely due to the reduced complexity of the term space at lower depths.
3. Strategies using the interleave effort, where our instantiation module is invoked more frequently, tend to perform better than those using lastcall. This suggests that there is a potential advantage in our approach.
4. Each strategy both solves some problems that the reference does not (*ref+*) and fails on some that the reference does solve (*ref-*). The lastcall strategies lose fewer solutions because the probabilistic instantiation is applied less frequently, resulting in behavior closer to the reference. The fewest losses occur with the paths variants at higher term depths, where the generated terms more closely resemble those of the reference strategy.
5. There appears to be no significant advantage of probabilistic generation over random term generation in this case. Since the depth is fixed to 0, the paths variant does not fully benefit from using different weight vectors for different positions. Moreover, the weights variant instantiates using all constants from the other instantiation modules, guided by probabilities based on their occurrence counts. In contrast, paths in this case only tracks statistics for constants that appear at the top level while all other constants are assigned a default weight of 1.

In the next Figure 3 we fix Depth to 0 and we evaluate various values for Flip. Higher values force our module to generate terms more different than term used by default instantiation modules. The table

Effort	lastcall				interleave			
Pick	weights		paths		weights		paths	
Flip	*total*	*ref (+/-)*	*total*	*ref (+/-)*	*total*	*ref (+/-)*	*total*	*ref (+/-)*
0.0	3497	+477/-57	3445	+427/-59	3570	+571/-78	3498	+497/-76
0.2	**3509**	+**500**/-68	3479	+463/-61	3579	+578/-76	3510	+504/-**71**
0.5	3474	+475/-78	3488	+467/-56	**3613**	+**612**/-76	3510	+508/-75
0.8	3507	+493/-63	**3515**	+**496**/-58	3579	+575/-**73**	3517	+516/-76
1.0	3335	+314/-**56**	3495	+473/-**55**	3429	+426/-74	**3532**	+**530**/-75

Figure 3: Grid evaluation of strategies with different Flip values (with fixed Depth $= 0$).

strategy parameters				*performance in greedy cover*			
Effort	Depth	Pick	Flip	*solves*	*+new*	*adds*	$=$ *total*
interleave	0	weights	0.5	3613	+3613	–	= 3613
standard	0	weights	0.2	3509	+141	+3.90%	= 3754
standard	0	weights	0.0	3497	+82	+2.18%	= 3836
standard	0	random	–	3523	+52	+1.36%	= 3888
interleave	0	paths	0.0	3498	+42	+1.08%	= 3930
standard	0	paths	0.2	3479	+29	+0.74%	= 3959
interleave	0	weights	0.8	3579	+24	+0.61%	= 3983
standard	0	weights	0.8	3507	+21	+0.53%	= 4004
interleave	0	weights	0.0	3570	+17	+0.42%	= 4021
interleave	0	random	–	3575	+14	+0.35%	= 4035
interleave	0	paths	0.5	3510	+12	+0.30%	= 4047
standard	3	random	–	3220	+11	+0.27%	= 4058
interleave	0	weights	0.2	3579	+10	+0.25%	= 4068
interleave	0	weights	1.0	3429	+9	+0.22%	= 4077
standard	0	paths	1.0	3495	+8	+0.20%	= 4085
standard	0	weights	0.5	3474	+7	+0.17%	= 4092
standard	1	paths	0.5	3124	+7	+0.17%	= 4099
standard	0	paths	0.0	3445	+6	+0.15%	= 4105
standard	1	random	–	3293	+6	+0.15%	= 4111
interleave	0	paths	0.8	3517	+5	+0.12%	= 4116

Figure 4: Greedy cover from the evaluated strategies helps to measure complementarity.

format is the same as in Figure 2 with the exception that the random case is omitted since the Flip parameter does not effect it. The line for Flip $= 0$ has the same values as in the line Depth $= 0$ in the previous figure.

The best results are obtained with the weights variant using the interleave effort and Flip $= 0.5$. This strategy solves 3,613 problems and produces 612 new solutions compared to the baseline. Overall, we observe that better performance is achieved when Flip is strictly greater than 0.0, suggesting that occasionally generating complementary terms is beneficial. By tuning the Flip parameter, we outperform the random term generation results shown in Figure 2.

The final experiment in this work aims to evaluate the mutual complementarity of the tested strategies. A greedy cover sequence is constructed from all evaluated strategies. The sequence is initialized by selecting the most effective individual strategy, and the problems it solves are marked. At each subsequent step, the strategy that solves the largest number of remaining unsolved problems is selected. This process is repeated iteratively. In this way, the greedy cover reveals strategies that complement one another.

The first 20 strategies in the greedy cover are presented in Figure 4. The first four columns specify the

strategy parameters, while the next four columns summarize their performance within the greedy cover. The column *solves* shows the number of problems each strategy solves individually. The remaining columns reflect each strategy's contribution to the overall portfolio performance. The column *+new* indicates how many additional problems the strategy contributes to the portfolio. The column *total* gives the cumulative number of problems solved by the portfolio up to that point, computed as the sum of *+new* and the previous row's *total*. Finally, the column *adds* expresses *+new* as a percentage of the current portfolio size.

We observe that the two most complementary strategies switch the effort mode from INTERLEAVE to STANDARD and use different values for FLIP. Since the second strategy contributes 3.90% to the portfolio, this indicates a meaningful level of complementarity among the strategies. Furthermore, we observe that the majority of strategies use a term depth of 0, meaning they instantiate using constants only. This suggests that the strategies not shown in Figure 2 and Figure 3, specifically those with DEPTH > 0 and FLIP > 0.0, do not yield any significant improvement.

5. Conclusions and Future Work

We have presented preliminary experiments on the probabilistic generation of instantiation terms for quantified formulas in cvc5. The results indicate potential in our approach, as we were able to improve upon the reference baseline strategy. The best performance was achieved by strategies that generate terms complementary to those produced by other instantiation techniques. This highlights the importance of diversity in instantiation strategies and suggests that probabilistic generation can effectively fill gaps left by more deterministic methods. Leveraging this complementarity may be key to further improving solver performance on quantified benchmarks. In contrast, guided generation of more complex or deeper terms has not yet proven to be effective.

Future work will focus on improving the guided generation of complex terms. Given the exponential growth of the instantiation term space, an effective reduction of the search space is essential. One possible direction is to employ a learning-based approach to guide term generation. We would like to refine the interaction between probabilistic instantiation and existing instantiation techniques, exploring how to better coordinate or prioritize between them during solving. Additionally, we plan to investigate adaptive mechanisms that adjust term generation parameters dynamically based on solver progress or formula structure. Another direction is to explore richer probabilistic models, beyond simple frequency-based grammars, that can better capture structural patterns in useful instantiations. Finally, we intend to evaluate our approach on broader and more diverse benchmarks to better understand its strengths and limitations in different theory combinations.

Acknowledgements

This work is supported by the Czech MEYS under the ERC CZ project no. LL1902 *POSTMAN*, and by the European Union under the project *ROBOPROX* (reg. no. CZ.02.01.01/00/22_008/0004590), and by the *RICAIP* project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: Grammar and spelling check, Paraphrase and reword. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] D. Detlefs, G. Nelson, J. B. Saxe, Simplify: A theorem prover for program checking, J. ACM 52 (2005) 365–473. doi:10.1145/1066100.1066102.

[2] A. Niemetz, M. Preiner, A. Reynolds, C. W. Barrett, C. Tinelli, Syntax-guided quantifier instantiation, in: Tools and Algorithms for the Construction and Analysis of Systems, TACAS, volume 12652, Springer, 2021, pp. 145–163. doi:10.1007/978-3-030-72013-1_8.

[3] Y. Ge, L. M. de Moura, Complete instantiation for quantified formulas in satisfiabiliby modulo theories, in: Computer Aided Verification CAV, volume 5643, Springer, 2009, pp. 306–320. doi:10.1007/978-3-642-02658-4_25.

[4] A. Reynolds, C. Tinelli, A. Goel, S. Krstić, M. Deters, C. Barrett, Quantifier instantiation techniques for finite model finding in SMT, in: 24th International Conference on Automated Deduction, CADE 2013, 2013, pp. 377–391. doi:10.1007/978-3-642-38574-2_26.

[5] A. Reynolds, C. Tinelli, 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. doi:10.1109/FMCAD.2014.6987613.

[6] M. Janota, H. Barbosa, P. Fontaine, A. Reynolds, Fair and adventurous enumeration of quantifier instantiations, in: Formal Methods in Computer-Aided Design, IEEE, 2021, pp. 256–260. doi:10.34727/2021/ISBN.978-3-85448-046-4_35.

[7] A. Reynolds, H. Barbosa, P. Fontaine, Revisiting enumerative instantiation, in: Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 10806, Springer, 2018, pp. 112–131. doi:10.1007/978-3-319-89963-3_7.

[8] E. Charniak, Statistical parsing with a context-free grammar and word statistics, in: B. Kuipers, B. L. Webber (Eds.), Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA, AAAI Press / The MIT Press, 1997, pp. 598–603. URL: http://www.aaai.org/Library/AAAI/1997/aaai97-093.php.

[9] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021, pp. 1267–1329. URL: https://doi.org/10.3233/FAIA201017. doi:10.3233/FAIA201017.

[10] V. Weispfenning, The complexity of linear problems in fields, Journal of Symbolic Computation 5 (1988) 3–27. doi:10.1016/S0747-7171(88)80003-8.

[11] A. Tarski, A Decision Method for Elementary Algebra and Geometry, Univ. of California Press, Berkeley, 1951. doi:https://doi.org/10.2307/jj.8501420, also in [?].

[12] J. H. Davenport, J. Heintz, Real quantifier elimination is doubly exponential, Journal of Symbolic Computation 5 (1988) 29–35. doi:10.1016/S0747-7171(88)80004-X.

[13] N. Bjørner, M. Janota, Playing with quantified satisfaction, in: International Conferences on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), 2015, pp. 15–27.

[14] D. Jurafsky, J. H. Martin, Speech and Language Processing, 3rd ed., 2023. Draft available at https://web.stanford.edu/~jurafsky/slp3/.

[15] C. Kaliszyk, J. Urban, J. Vyskočil, Automating Formalization by Statistical and Semantic Parsing of Mathematics, Springer International Publishing, 2017, p. 12–27. doi:10.1007/978-3-319-66107-0_2.

[16] C. Barrett, P. Fontaine, C. Tinelli, The SMT-LIB Standard: Version 2.6, Technical Report, Department of Computer Science, The University of Iowa, 2017. Available at www.SMT-LIB.org.

[17] C. Barrett, P. Fontaine, C. Tinelli, The Satisfiability Modulo Theories Library (SMT-LIB), www.SMT-LIB.org, 2016.