Selecting Quantifiers for Instantiation in SMT

Jan Jakubův[1], Mikoláš Janota[1], Bartosz Piotrowski[1], Jelle Piepenbrock[1], and Andrew Reynolds[2]

[1] Czech Technical University in Prague, Czechia
{jan.jakubuv,mikolas.janota,jelle.piepenbrock}@cvut.cz
[2] University of Iowa, USA

Abstract

This work attempts to choose quantifiers to instantiate based on a feedback from the SMT solver. This tackles the challenge that if a problem contains many quantifier expressions, it eventually gets flooded by too many generated ground terms. Therefore, we instantiate only some of the quantifiers but the question is, which are the useful ones?

The SMT solver is modeled as a multi-armed bandit, where each quantifier is a lever producing a positive reward if an instantiation of the quantifier was useful. The issue is that we do not know whether an instantiation was useful or not until the given problem instance is actually proven (and then it's too late). In this paper we explore several heuristics that attempt to assess the usefulness of quantifier instantiations.

1 Introduction

Satisfiability modulo theories (SMT) solvers abstract quantifiers as propositions that are sources of instantiations. So a quantified sub-formula $\forall \vec{x} \phi$ is abstracted as a Boolean variable Q and instantiations are added in the form of an implication $Q \Rightarrow \phi[\vec{x} \mapsto \vec{t}]$ for some ground terms $\vec{t}$. These ground terms are chosen from the set of ground terms already present in the ground formula. Consider the following toy formula:

$$f(3) < 0 \wedge ((\forall x f(x) > 0) \vee (\forall x f(x) > x)),$$

where $f\colon \text{int} \to \text{int}$. To obtain a refutation, we abstract and instantiate as follows:

abstraction:	$Q_1 \equiv (\forall x f(x) > 0), Q_2 \equiv (\forall x f(x) > x)$
abstracted formula:	$f(3) < 0 \wedge (Q_1 \vee Q_2)$
instantiation 1:	$Q_1 \Rightarrow f(3) > 0$
instantiation 2:	$Q_2 \Rightarrow f(3) > 3$

Most approaches instantiate quantifiers *gradually*, meaning, the new instantiations are added after testing that the old ones do not already yield a contradiction in the ground solver. In this work, we propose to use dynamic machine-learning based heuristic to decide which quantifiers should be instantiated during solving. We model the problem as *multi-armed bandits (MAB)*, which is one of the simplest form of *reinforcement learning* [9]. A multi-armed bandit is equipped with a set of *levers*, which are pulled one at a time. Pulling a lever leads to a *reward* (typically a real number). The multi-armed bandit problem is to find a strategy that leads to the highest total reward overall. Our scenario is more complicated because quantifiers influence each other and the SMT solver has a state, which changes with each instantiation. Hence, here we apply the MAB model as a simplification of the problem as a first step in this area of research.

The overall setup is shown in Figure 1. Quantifiers are instantiated in a loop that terminates if the ground solver determines that the ground part is unsatisfiable. Our guidance picks the

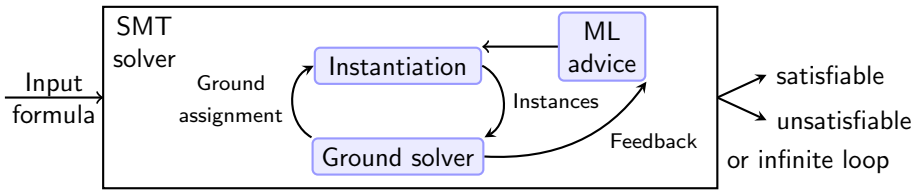

Figure 1: Schematic of the SMT solver with ML guidance for quantifier instantiation.

set of quantifiers to be instantiated in each round. This guidance is based on feedback obtained from the ground solver.

In this paper, we only consider the simplest strategy for picking the ground terms that we instantiate with: the *enumerative instantiation* [5,8]. There the ground terms are enumerated from the Herbrand universe, while almost completely ignoring the semantics of the formula at hand. However, more advanced techniques exist [2,3] and similar approach could be integrated into them. In the standard setting, these techniques would instantiate *all* quantifiers that are true in the current ground model. This can rapidly bloat the ground part, which means that it is harder to solve but also that the space of possible instantiations is getting unwieldy. We also remark that we had applied machine learning to ordering terms for instantiation within cvc5 [4]—this could also be combined with the approach presented here.

2 Picking Quantifiers

In the standard MAB setting, only one lever is pulled at a time. This is not desirable in our setting, because this could cause an unwieldy number of instantiation rounds. Therefore we always pick a *set* of quantifiers (levers) to instantiate in each round. This is always done based on some *score* assigned to each quantifier, where the score is calculated from the observations made so far (Section 3). There are multiple ways of how one could use such scores. In our experiments, we use a simple strategy where the top p percent of quantifiers is selected in each round, where p is a fixed parameter.

3 Scores and Reward Heuristics

The objective is to assign to each quantifier a score that determines how likely that quantifier is to be instantiated. This score combines two components. The first component estimates how useful the quantifier has been in the past. The second component pushes the search towards quantifiers that have been underused so far. This falls within the well-known paradigm in reinforcement learning: *exploration versus exploitation*.

The current implementation uses as a source of usefulness of a quantifier two values obtained from the SAT solver, namely *difficulty measure* and *active lemma statistics*. Both of these sources provide an integer statistic for each quantifier, with values evolving in time. We further normalize the values, dividing them by the number of current quantifier instantiations. Hence we obtain the utility per quantifier instance. We use these values to estimate usefulness of quantifiers and to guide quantifier instantiation.

In our implementation, *difficulty* is mapping from formulas (those from input or lemmas generated during solving) to natural numbers, where a formula with higher difficulty contributes more towards a proof of unsatisfiability. This measure is incrementally updated during the

course of SMT solving. In particular, when a theory solver generates a conflict clause or lemma φ, we compute the literals ℓ that are propositionally entailed by the negation of φ (e.g. the negation of all literals in a clause), call this set $\mathbf{ilits}(\varphi)$. For each such ℓ, we increment the difficulty measure of the (oldest) formula ψ such that the negation of ℓ is in $\mathbf{ilits}(\psi)$. We subsequently track this difficulty measure on the generated lemma φ. Notice that quantifier instantiation lemmas $Q \Rightarrow \phi[\vec{x} \mapsto \vec{t}\,]$ increment difficulty on the oldest formula that implies Q. The difficulty measure of the lemma itself will be incremented whenever literals in the instantiated body of the quantified formula participate in further conflicts and lemmas. Similarly, we measure *activity*, which tracks how many times a formula participated in propositional propagation.

As a basic estimate of the quantifier quality, we train a simple *linear regression* model to obtain reasonable weights α and β for the linear combination of normalized activity and difficulty $\alpha \cdot \mathit{nact} + \beta \cdot \mathit{ndiff}$. As training data, we collect statistics of the values of *nact* and *ndiff* from runs on several thousand problems evaluated with 60 seconds time limit. The statistics contain the values of *nact* and *ndiff* for each quantifier in every instantiation round, thusly recording the progress of the values in time. The quantifiers whose instantiation was needed to solve the problem are marked as positive, other quantifiers as negative. From this training data, we train a linear regression classifier. The linear regression training yielded the coefficients $\alpha = 0.04$ and $\beta = 0.1$ and these shall be used for the experimental evaluation in Section 4. The prediction computed by the linear formula can be directly used as the quality estimate, or it can be used as a *reward* in the MAB setting as follows.

To include the exploration factor we use the *Upper Confidence Bound (UCB)* formula, where the quality $Q_t(q)$ of the quantifier q at time step t (quantifier selection round) is defined as

$$Q_t(q) = R_t(q) + c\sqrt{\frac{\log(t)}{N_t(q)}}$$

with $R_t(q)$ is the mean reward for the quantifier q so far, and $N_t(q)$ is the number of times q was selected for instantiation up to this point. As rewards, we use predictions of quantifier qualities obtained from the linear model described above. Finally, c is the confidence value controlling the level of exploration with respect to exploitation. The idea behind UCB is that the quality is primarily governed by the reward. The second summand will be higher for the quantifiers that were less often selected in the previous instantiation rounds (quantifiers with smaller $N_t(q)$). Hence the quality of unselected quantifiers keeps increasing and they will eventually become selected in some future round. How fast this happens is controlled by the parameter c where higher values favor exploration.

4 Experiments

The above-presented ideas were implemented in the SMT solver cvc5 [1], while turning off all the other instantiation techniques and focusing only on the enumerative instantiation [5, 8]. A subset 5000 of SMTLIB benchmark problems is used for the experiments. We deliberately select problems with a large number of quantifiers. The experiments were run on a server with four AMD EPYC 7513 32-Core Processor @ 2.6GHz and with 504 GB of memory.

The default technique where all available quantifiers are instantiated in each round is used as the *baseline*. To assess the performance of our techniques for quantifier selection, we evaluate several *random selection* strategies. In each strategy, k% of randomly selected quantifiers is instantiated with k ranging from 10%, 20%, up to 90%. Apart from these control strategies, we evaluate our *linear model* from Section 3 where the quality of each quantifier is computed as

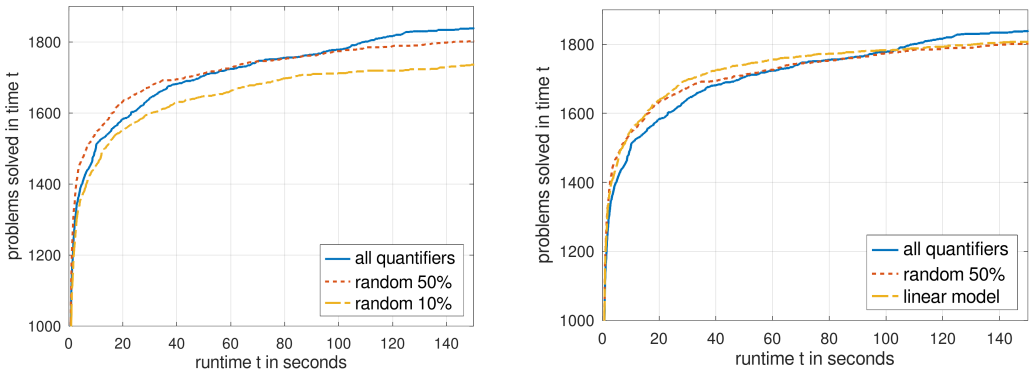

Figure 2: Problems solved by various quantifier selection methods.

a linear combination of two utility values *nact* and *ndiff*. The linear model is evaluated in two modes, namely, with and without the use of the UCB formula. As with the random strategies, we evaluate several variants with different selection ratios k, selecting $k\%$ of the quantifier with the highest quality. In the case of the UCB formula, additional *confidence* coefficient c is required to control the level of exploration. We evaluate several values for c between 0.001 and 10.

The ML approach performs well in short timeouts. With the timeout of 1 s the *baseline* strategy, which instantiates all available quantifiers, solved 1107 problems. The worst of the *random selection* strategies for $k = 10\%$ solved 1016 problems, and the best one for $k = 50\%$ already outperformed the baseline with 1171 solved problems. The *linear model* scored slightly higher with 1186 problems, in particular, its best variant with $k = 50\%$. The UCB formula led to an almost identical performance with 1190 solved problems, namely, its variant with $k = 80\%$ and $c = 3$. Our techniques perform slightly better than the random selection. The two random strategies solved 1212 problems together, while the two ML strategies solved 1264.

The experiment progress is depicted in Figure 2, where the graphs show the number of problems solvable by each of the strategies in time t. Both graphs use the baseline strategy as the reference. The left graph compares the baseline to the random selection, while the right graph compares it to the best ML-based selection technique, namely the linear model with the UCB formula. We see that the line for random 10% performs more poorly than the baseline. On the other hand, the random 50% line stays above the baseline up till ≈ 60 s. The experiment shows the versions with and without the UCB formula to be of comparable performance. The right graph, which compares our best ML selection method with best random selection, supports our previous observation that the ML selection slightly outperforms the random one. The ML approach stays above the baseline up till ≈ 100 s. The methods are slightly complementary, the version without UCB adds 6.4% to the problems solved by the UCB version.

While our strategies outperform the random selection, the improvement is rather small. This is further supported by Figure 3, which compares the strategies in a scatter plot. For each solved problem p, a point is plotted at the position (t_1, t_2) where t_i is the runtime of strategy s_i on problem p. In both graphs, s_1 is the baseline strategy, while s_2 is the random selection of 50% in the left graph and the linear model with UCB on the right. Hence the points below the diagonal designate the runtime improvement, that is, problems solved faster than the baseline. For the ML approach we observe a cluster of points below the diagonal with time below 10 s. For larger times we observe both improvement and worsening of performance.

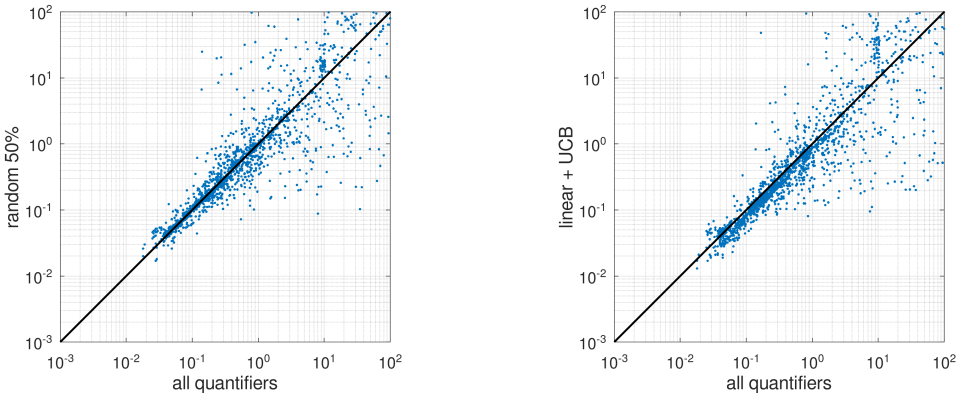

Figure 3: Runtime comparisons of the baseline and quantifier selection methods.

5 Conclusions and Future Work

In this extended abstract we report on our initial experiments that attempt to estimate the usefulness of quantifiers *during* search. This contrasts with standard machine learning techniques that learn from one problem to another (cf. [4, 6, 7, 10]). In our case, the objective is to obtain feedback from the problem being solved. To our best knowledge, this is the first time such an approach has been attempted. Eventually, machine learning across multiple instances could be combined with single-instance learning.

The experiments suggest that a guided quantifier selection can lead to a runtime improvement and hence to more problems solved with very short timeouts. It seems, however, that the improvement over the baseline strategy is decreasing with increasing time limits. In our future research, we plan to investigate whether an improvement over higher time limits can be achieved with more advanced methods for quantifier selection. In particular, we plan to investigate other metrics for the usefulness of quantifiers.

Acknowledgments

The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project *POSTMAN* no. LL1902, the European Regional Development Fund under the Czech project AI&Reasoning no. CZ.02.1.01/0.0/0.0/15_003/0000466, the grant of National Science Center, Poland, no. 2018/29/N/ST6/02903, and Amazon Research Awards. 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.

References

[1] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held*

as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.

[2] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: A theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.

[3] Yeting Ge and Leonardo Mendonça de Moura. Complete instantiation for quantified formulas in satisfiability modulo theories. In *Computer Aided Verification, 21st International Conference, CAV*, pages 306–320, 2009.

[4] Mikoláš Janota, Jelle Piepenbrock, and Bartosz Piotrowski. Towards learning quantifier instantiation in SMT. In *Theory and Applications of Satisfiability Testing – SAT 2022*. LIPIcs, 2022.

[5] Mikoláš Janota, Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds. Fair and adventurous enumeration of quantifier instantiations. In *Formal Methods in Computer-Aided Design*, 2021.

[6] Jelle Piepenbrock, Tom Heskes, Mikoláš Janota, and Josef Urban. Guiding an automated theorem prover with neural rewriting. In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings*, volume 13385 of *Lecture Notes in Computer Science*, pages 597–617. Springer, 2022.

[7] Jelle Piepenbrock, Tom Heskes, Mikoláš Janota, and Josef Urban. Guiding an automated theorem prover with neural rewriting. In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings*, volume 13385 of *Lecture Notes in Computer Science*, pages 597–617. Springer, 2022.

[8] Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. Revisiting enumerative instantiation. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 10806, pages 112–131, 2018.

[9] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. Adaptive Computation and Machine Learning series. Bradford Books, Cambridge, MA, 2 edition, November 2018.

[10] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jirí Vyskocil. MaLARea SG1 – machine learner for automated reasoning with semantic guidance. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2008.