APPLYING AI METHODS IN AUTOMATED THEOREM PROVING FOR SOFTWARE VERIFICATION

Filip M. Marić[1]

[1] Faculty of Mathematics
University of Belgrade, Studentski trg 16, 11000 Belgrade, Serbia
Email: filip@matf.bg.ac.rs

Abstract

Ensuring hardware and software correctness is vital in all computer applications. Software verification is a well-developed discipline of software engineering that relies both on the experimental approach (software testing) and on static program analysis (formal, mathematical proofs of software correctness). This brief survey will present some state-of-the-art results in applying deductive and inductive artificial intelligence methods in software verification.

Key words: software verification, machine learning, automated theorem proving, interactive theorem proving

1 Introduction

Software verification can be either dynamic, examining the runtime program behavior by testing, or static, examining the program source code. Static program analysis usually employs some formal methods and tools. Formal software verification is the act of mathematically proving the algorithm and implementation correctness with respect to its formal specification. Therefore, software verification has always been relying on automated, deductive reasoning techniques and tools, such as SAT and SMT solvers, automated theorem provers (ATPs), and interactive theorem provers (ITPs). Such automated reasoning tools and techniques are based on mathematical logic (propositional logic, first order logic, higher-order logic, modal logics etc.) and are considered to be traditional artificial intelligence (AI) systems.

On the other hand, in the last decade we have seen an amazing development of artificial intelligence tools based on inductive reasoning. Machine learning (ML) and data-driven software development might have become a dominating software development paradigm. Although such techniques are inexact and cannot be directly applied to determine software correctness, there are many scenarios when they are applied in conjunction with exact, deductive methods yielding more powerful, hybrid theorem proving and software verification tools.

In this survey we shall briefly describe some of the most prominent results in applying modern AI and ML methods in theorem proving and software verification, specially focusing on results produced by researchers from Faculty of Mathematics, University of Belgrade.

2 Applications

Artificial intelligence and machine learning techniques have been successfully applied for software testing, algorithm selection and parameter tuning, and in automated theorem proving. On the other hand, there have been some reports on applying software verification techniques for specifying and checking correctness of machine learning system.

2.1 Software testing

Traditional software testing requires manually writing test-scripts, manually analyzing their results (failed tests) and manually maintaining tests when requirements and software changes (e.g., when the UI is updated). AI techniques have been successfully applied in various ways to reduce manual work, to improve test automation and to make the testing workflow much more convenient for testers. There are many research papers describing specific techniques, and there are several survey studies giving a meta-analysis and comparison of individual proposed techniques (e.g., a systematic mapping study[2]).

A key aspect of software testing is designing test cases and there has been a growing interest in applying ML to automate test case generation (i.e., test case design). Given a program P and a set of programs P', ML techniques can generate test cases that are adequate in sense that they are able to distinguish P from all programs in P'. In modern GUI apps, ML can be used to learn an abstraction of the model of the GUI during testing, so that the learned model can be used generate user inputs that visit unexplored states of the app.

Another task where ML methods might be beneficial is designing test-oracles. Namely, when there are no explicit specification it is hard to tell when a test-case should pass, and ML is used to generate test oracles for determining whether a test has passed or failed. For example, in regression testing, learning from the previous successful runs can trigger warnings about potential performance degradation in future system versions.

Testers need to be able to assess the quality of a given test suite. Branch coverage and mutation coverage are usually used as quality measures, but recently the idea of behavioral coverage is introduced, which infers a model from a system by observing its behavior (i.e., outputs) during the execution of a test suite. ML has also been applied to remove infeasible test cases from a test suite, saving testing time (for example, an infeasible test in a GUI application would be the one that would produce a push-button event in a context where that button is disabled).

For large software projects test case suits might require too much time to execute (several days or even weeks), and ML techniques have been used to prioritize tests and improve the effectiveness of the testing effort when testing resources are limited.

2.2 Automated programming grading

In the age of online education and massive open online programming courses it is very desirable to automate grading of students' exams and assignments. Software verification techniques based on both deductive and inductive reasoning have been successfully applied for this task[5, 6]. Techniques based on regression verification detection can be used for comparing student's and teacher's solutions and proving full or k-equivalence between them (two program are k-equivalent if they yield same results when loops are executed at most k times). Techniques based on graph similarity of control flow graphs

can be used to compare solutions for structural similarity and to indicate potential errors in programming style (in introductory programming courses, students solutions are considered to be better if they better match the coding style of official teacher's solutions).

2.3 Algorithm selection and parameter tuning

Algorithm selection is a meta-algorithmic technique to choose an algorithm from a portfolio on an instance-by-instance basis. In theorem proving and software verification there are often many competing systems that solve the same problems with different efficiency. For example, there are many propositional satisfiability (SAT) solvers that show different performance on various SAT instances. SAT portfolio systems such as SATZilla[14], ArgoSmArT[8] choose one of many available SAT solvers based on syntactic and other characteristics of the SAT instance being solved. Various ML techniques are used to build the model used to the select a solver that would show good performance. Random forests of cost sensitive decision trees decision trees are often used (in SATZilla, ISAC, 3S or CHSC), but even simpler techniques such as k nearest neighbors show good performance (e.g. in ArgoSmArT). Similar techniques can be used for choosing between different systems within a portfolio or for choosing different configurations of parameters of a single system. Apart from SAT, this approach has been applied in various hard combinatorial problems (e.g., MIP, CSP, AI planning, QBF, ASP, ...). For example, the system MeSAT[12] offers different methods of encoding CSP into SAT and uses algorithm selection to select the best one. The approach has also been applied on automated theorem proving (e.g., in geometry[9]).

2.4 Theorem proving

Software verification relies on theorem proving. Automated theorem proving (ATP) has been a long standing goal for AI systems. Theorem provers have traditionaly been based on deductive reasoning in some logical calculi. Theorems in first and higher-order logics are usually proved using some variants of the resolution method, tableaux method in combination with decision procedures for specific theories (e.g., linear arithmetic). Such ATP systems can autonomously prove non-trivial mathematical theorems. For example, an ATP based on coherent logic manages to prove 37% of the theorems from Tarski's seminal book on geometry (with readable and machine verifiable proofs generated) without any guidance[3].

Recently machine learning techniques have been applied to help guide deductive systems. Henri Poincaré argued that we prove by logic, but discover by intuition, and such combination of traditional logic-based methods with machine learning that can give some sort of intuition might give us big advances in computer-based theorem proving. We shall describe several examples of this approach.

The *sledgehammer* tool[10] available within interactive theorem prover Isabelle/HOL finds proofs by running several first-order logic automated theorem provers and SMT solvers. The current goal formulated in higher order logic of Isabelle/HOL is translated into first order logic before being handled to FOL ATPs, along with relevant lemmas and axioms. One of key components of the system is relevance filtering which among several thousand available lemmas and axioms selects a subset that could potentially contribute to proving the goal. Currently this component is learning-based[1] and it outperforms earlier selector that did not apply learning and selected lemmas based on scores assigned by symbols that they contain.

Such hybrid systems start to show quite good performance. Flyspeck project is a large formalization effort culminating with a formal of Kepler's conjecture. An AI system[13] that combined external ATPs and machine-learning premise selection methods trained on the Flyspeck proofs managed to solve 39% of the 14185 theorems in the corpus in a push-button mode (without any high-level advice and user interaction).

The International Mathematical Olympiad (IMO) is perhaps the most celebrated mental competition in the world and as such is among the greatest grand challenges for Artificial Intelligence (AI). The IMO Grand Challenge[1], recently formulated, requires to build an AI that can win a gold medal in the competition, and first steps in that direction have been made. Problems are considered solved only when solutions can be mechanically checked by interactive theorem provers. There have been manual formalizations of large collections of IMO problems[7]. A few months ago a neural theorem prover for Lean has been built[11] that learned to solve a variety of challenging high-school olympiad problems, including two problems adapted from the IMO. The prover uses a language model to find proofs of formal statements. Each time a new proof is found, it is used as new training data, which improves the neural network and enables it to iteratively find solutions to harder and harder statements.

2.5 Verification of Machine learning systems

The current uses of ML systems in verification and theorem proving are such that ML is used for only as guiding and heuristic components, so they need not be verified. However, importance of ML in modern software definitely requires careful and more formal analysis. For example, a very important property of ML systems is robustness to adversarial examples, and ML systems can be used in safety critical applications only when it is shown that there is no combination of inputs that will generate an undesirable output. Verification of such properties is still in its infancy, because methods make assumptions that prevent them from providing absolute guarantees of the absence of adversarial examples [4]. However, this research topic gains a lot of attention recently and we might expect to have future ML which have been formally analyzed and verified.

3 Conclusions

Theorem proving and software verification have been relying on exact, deductive AI techniques, from its very beginning. Automated reasoning has been a field of AI that has both enabled modern software verification system, and has been driven forward by the needs and developments in software verification. State-of-the-art results show that modern AI systems and ML can successfully be applied in conjuction with traditional automated reasoning algorithms to give a new generation of more powerful automated reasoning systems.

[1] https://imo-grand-challenge.github.io/

References

[1] J. C. Blanchette et al., A Learning-Based Fact Selector for Isabelle/HOL, Journal of Automated Reasoning volume 57, 2016.

[2] V. H. S. Durelli et al., "Machine Learning Applied to Software Testing: A Systematic Mapping Study," in IEEE Transactions on Reliability, vol. 68, no. 3, pp. 1189-1212, Sept. 2019.

[3] S. S. Djurdević et al. Automated generation of machine verifiable and readable proofs: A case study of Tarski's geometry. Ann. Math. Artif. Intell. 74(3-4): 249-269 (2015)

[4] I. Goodfellow and N. Papernot. The challenge of verification and testing of machine learning. Cleverhans-blog, 2017.

[5] M. V. Janičić et al: Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments. Information and Software Technology, Elsevier 2013.

[6] M. V. Janičić and F. Marić. Regression Verification for Automated Evaluation of Students Programs. Computer Science and Information Systems, 17(1), 2020.

[7] F. Marić and S. Stojanović-Djurdjević, Formalizing IMO Problems and Solutions in Isabelle/HOL, In proceedings of ThEdu 2020., EPTCS 328, 2020.

[8] M. Nikolić, F. Marić, P. Janičić, Simple algorithm portfolio for SAT, Artificial Intelligence Review, 40, 2013.

[9] M. Nikolić et al. Portfolio theorem proving and prover runtime prediction for geometry, Annals of Mathematics and Artificial Intelligence (85), 2019.

[10] L. C. Paulson and J. C. Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. PAAR-2010: Workshop on Practical Aspects of Automated Reasoning. Edinburgh, Scotland, July 2010.

[11] S. Polu et al., Formal Mathematics Statement Curriculum Learning. preprint, https://arxiv.org/abs/2202.01344, February, 2022.

[12] M. Stojadinović, F. Marić, meSAT: multiple encodings of CSP to SAT. Constraints, 19(4), 2008.

[13] C. Kaliszyk, J. Urban, Learning-assisted automated reasoning with Flyspeck, Journal of Automated Reasoning, 2014.

[14] Xu et al., SATzilla: Portfolio-based Algorithm Selection for SAT, Journal of Artificial Intelligence Research 32, 2008.