ARC: An Educational Project on Automated Reasoning in the Class

Isabela Drămnesc, Tudor Jebelean, Erika Ábrahám, Gábor Kusper, Sorin Stratulat

ARC: An Educational Project on Automated Reasoning in the Class

I. Dramnesc[1], E. Abraham[2], T. Jebelean[3], G. Kusper[4], S. Stratulat[5]

[1]West University of Timisoara
Romania
isabela.dramnesc@e-uvt.ro

[2]RWTH University of Aachen
Germany
abraham@cs.rwth-aachen.de

[3]Johannes Kepler University
Austria
Tudor.Jebelean@jku.at

[4]Eszterhazy Karoly Catholic University of Eger
Hungary
gkusper@aries.ektf.hu

[5]University of Lorraine
France
sorin.stratulat@univ-lorraine.fr

Abstract: The *international Erasmus+ European Project: "ARC – Automated Reasoning in the Class", running from 2019 to 2022* is a partnership of universities from Austria, France, Germany, Hungary, and Romania, and has the purpose of developing advanced material for teaching subjects related to Computational Logic by using Automated Reasoning. The material includes a comprehensive textbook treating the necessary theoretical background (selected topics in Mathematical Logic), but mostly the practical methods from Automated Theorem Proving, as well as the description of the basic programming paradigms and the associated languages, in relation to their logical aspects. Furthermore, we address the most important applications, like program verification and testing, semantic representation of information, algorithm synthesis, etc. One of the main goals of the approach is to improve the logical background of the software professionals in order to motivate them to use formal methods for certification of complex systems and thus to avoid costly failures.

Keywords: computer science, automated reasoning, education, problem based learning

1. Introduction

The human society is exposed to increased risks due to the fast increase in the complexity of software systems. The consequences of software failures range from small nuisances (blocked smartphone, laptop shut down, virus attacks), to big nuisances (failure of an online submission system just before the deadline, chaos in an airport due to failure of luggage handling system, breakdown of the electrical power for a whole region), to important loss of value (loss of a space rocket, after-sale replacement of millions of defect devices) and even to fatalities (failure of an anti-rocket military system, train or airplane crash). One of the main reasons for the failures of software systems (and in general of more complex systems including software) is the relatively low training of software professionals in Computational Logic and in Automated Reasoning, which prevents them to use formal methods in the process of software development. In our experience, there exists a big gap between the learning offer from universities and the requests on the labor market. Our target groups are teachers and students from higher educational institutions.

In the frame of an ongoing international European project, we address this problem because of: the lack of training experts in the educational process, online misinformation, fake news, cyberbullying, and unsecured systems; the lack of interaction and exchange of good practices with experts in the field; and limited access to licensed

professional software. If nothing is done, this big gap becomes increasingly bigger and leads to many dramatic effects: students adapting hard at work, the lack of entrepreneurs, falling living standards, teachers leaving their universities and moving to other universities which offer them support in developing their professional skills and competences in the field, if the curriculum in this field is not updated, the students will not be interested to enroll to such old courses and it will be a dramatic reduction of the number of students, if universities are losing good teachers and students, the quality of the educational process and research, and, implicitly, the ranking of the universities decrease, no possibility to attract foreign teachers and students at the universities.

The main goal of our project (Project 2019-2022), referred as ARC in the rest of the paper, is to improve the education of Computer Science students in fields related to Computational Logic both by developing learning material as well as by training the academic staff. We aim to increase the knowledge and skills of Computer Science graduates in the fields related to Computational Logic, because we believe that this will have a positive impact on the safety and reliability of software. For achieving this objective we want to improve in a significant manner the teaching of computational logic topics for students in computer science, by using computer support based on automated reasoning. More concretely we develop specific interactive exercises for the training of students, and we also demonstrate practically the notions and the methods from computational logic topics.

The main activities of the project consist in: the ARC book: "Computational Logic, a Practical Approach", development of various interactive learning tools based on Automated Reasoning, 5 international training events for the academic staff, training of more than 5000 students in more than 20 courses over 6 semesters (2019 - 2022), as well as the dissemination of the final project results through an international summer school for students and an international workshop for academics (summer 2022).

2. The ARC Book and Tools

One of the most important outcomes of the project is the book "Computational Logic: A Practical Approach". The purpose of the book is threefold:

- to provide supporting material for academics that prepare courses related to computational logic,
- to constitute a tutorial introduction to the theoretical and practical aspects of computational logic, for individuals who prefer to pursue an individual study,
- to help the reader to understand the basic proving techniques that can be used by himself in order to develop formal proofs.

In order to fulfill these goals, the book contains:

- a short introduction to the most important theoretical aspects of mathematical logic,
- a practically oriented presentation of the most relevant algorithms in automated theorem proving, including those inspired from human proving techniques,
- associated tools with tutorials and examples that facilitate the understanding of automated reasoning techniques and their implementation.

2.1. Book content

The book starts with an introduction that motivates the reader by explaining the *purpose of mathematics, mathematical logic, and computational logic*, as well as their role in the current and future landscape of science and technology. In this context, we also reveal the essential role of the software professional and the importance of computational logic for his activities.

In the sequel the book introduces the most important aspects of *mathematical logic*, as the theoretical basis for automated reasoning, as well as the classical methods for formula manipulation. These start with the syntax and semantics of propositional logic, followed by basic equivalences and their use in rewriting, as well as the main techniques for natural style proving and sequent calculus. After that we present the main notions related to first-order predicate logic and the further development of rewriting, natural style proving, and sequent calculus for this logic. This chapter ends with a presentation of the basics of higher-order logic. This logic is presented in comparison with the first-order logic. The formalization of the Noetherian induction principle in higher-order logic is given as an example of a higher-order formula that can be instantiated in order to be applied to first-order reasoning.

The next chapter is dedicated to the presentation of the main techniques for *automated reasoning*, that is those formula manipulation methods that have been developed specifically for being implemented on computers. For propositional logic we describe the transformations into conjunctive normal form (CNF) and disjunctive normal form (DNF), we discuss the practical aspects of implementing natural style proving and sequent calculus, and the resolution principle including its special variant - the DPLL algorithm (Davis 1961, Harrison 2009) for solving the SAT (satisfiability) problem. A review of the most important techniques for SAT is also included.

These are then further generalized to first-order logic, leading to prenex normal form and Skolem normal form, as well as special techniques for natural style proving and sequent calculus for predicate logic formulae. The automated reasoning algorithms for propositional and for first-order logic are illustrated practically by implementations in Mathematica (Wolfram 2003) and Theorema (Buchberger 2016). In particular we illustrate the possibility of mechanizing proofs in complex theories, as well as the principles of proof based algorithm synthesis, by presenting some practical experiments regarding the synthesis of sorting algorithms for lists (Drămnesc 2021-b) and binary trees (Drămnesc 2021-a). A special place is awarded to SMT (satisfiability modulo theories), which combines logical methods with techniques from Computer Algebra, especially with respect to reasoning on numerical domains, like integers and reals. Concerning higher-order logic, two first-order instances of the Noetherian induction principle introduced in the theoretical part are discussed: formula-based induction and term-based induction - they are compared and most of their implementations are detailed. Implicit induction is a rewrite-based induction technique that implements the formula-based Noetherian induction principle. Different two examples of implicit induction inference systems are presented and shown to be sound. This part ends by the description of SPIKE, an implicit induction theorem prover. The SPIKE proofs can be automatically certified using the Coq proof assistant (Coq 2022). The certification process applies a general methodology for certifying formula-based Noetherian induction reasoning (Stratulat 2015). Coq was also used to manually certify synthesized sorting algorithms on binary trees, as those produced using the Theorema system. This chapter ends with an overview of the most important applications of automated reasoning: verification and testing of software and of complex systems, deductive databases, algorithm synthesis, semantic storage and retrieval of information, automation of tutoring systems, smart artifacts, etc.

The following chapter discusses the *logic of computation,* namely the relation between computational logic and programming. This starts with an introduction to reduction systems and (conditional) term rewriting systems, as well as to critical pair completion for reduction to normal form, which constitute the theoretical basis of computation. Next we give an overview of the programming paradigms and then we investigate each of them in detail: imperative (procedural), object oriented, functional, pattern-based, and logic programming. For each paradigm we present the relation to logic and we investigate the consequences of this relation for defining the semantics of programs and the usage of this for verification and testing of programs. The chapter ends with a detailed discussion on the main techniques for program verification: symbolic execution, Hoare logic, model checking, proof carrying code, algorithm and proof certification, etc.

The last chapter of the book is dedicated to computer supported problem solving, containing practical examples related to certain applications of logic that are central to the expertise of the authors: model construction, automation of theory exploration applied to some important domains (numerical domains, lists, binary trees), as well as algorithm synthesis in these theories.

2.2. Tools

We developed various tools or adapted some existing ones in order to support the teaching process.

SAT Solvers. We developed a SAT (satisfiability) solver (Biere 2009), called CSFLOC (Kusper 2018), and we use several other SAT solvers, like MiniSAT (Eén 2004). We use the common representation language, called DIMACS, which is a very simple text based file format. Since SAT is NP complete (Cook 1971), we can efficiently convert NP hard problems into SAT, solve it by SAT solver and convert back the solution to be the solution of the original problem.

SMT-RAT. We have started the development of our satisfiability-modulo-theories (SMT) solver (Kröning 2008) named SMT-RAT (Corzilius 2015) more than a decade ago, with the goal to have a modular program that offers the SMT-compliant implementation of numerous decision procedures for real algebra, and allows their strategic combination for checking the satisfiability of quantifier-free real algebraic formulas.

Mathematica. We developed several illustrative implementations in Mathematica (Wolfram 2003) of some basic methods for logical operations: interactive and automatic construction of the truth table, transformation into various normal forms, resolution, the DPLL algorithm (Harrison 2009), and the Chaff (Moskewicz 2001) version of DPLL.

Plain C. We show how to implement the representation of propositional logic formulae as trees and how to transform efficiently an arbitrary formula into negation normal form (NNF) by operating on this tree.

Theorema. We develop various tutorial examples on using the Theorema system (Buchberger 2006) for developing and testing algorithms, and for natural style proofs. Furthermore, we present a tutorial implementation of sequent calculus, as well as an implementation of a proof based algorithm synthesizer for sorting algorithms and a program transformation tool.

COQ. The Coq proof assistant (Coq 2022) is a powerful and widely used tool for certifying software. It is a combination of two components: i) a programming language for writing programs, by defining data structures, functions and predicates, and ii) a set of tools for stating assertions (including properties about the behavior of programs) and for proving them. It has been used to certify non-trivial mathematical results, as the `four color' and the `odd order' theorems, and applications, as a C compiler. In 2013, it received the prestigious ACM Software System award.

SPIKE. SPIKE (Stratulat 2020) is an automated theorem prover that can reason on conditional equational specifications by using (mathematical) induction. It implements an instance of the formula-based Noetherian induction principle applied on a Noetherian poset of (equational) clauses. Several clauses can be simultaneously tested to verify whether they are (logical) consequences of a given set of axioms, written as conditional equalities. SPIKE is able to apply different induction techniques as implicit, term rewriting, cyclic, mutual, simultaneous and lazy induction. The proofs can be converted to Coq scripts and certified using the Coq proof assistant. The specifications are sorted and constructor-based, and should satisfy some properties, like the ground convergence and sufficient completeness. SPIKE has been used to verify some non-trivial applications, such as the JavaCard platform (Barthe 2003) and the conformity algorithm for a telecommunication protocol (Rusinowitch 2003).

3. Training activities

At every training participated two persons from every partner institution, therefore, after each training, the trained teachers shared the information through their colleagues in the partner universities by organizing local “trainings after trainings”.

3.1. Training in Mathematica and Theorema

The purpose of this training was to understand the tools developed by the Austrian partner and to learn how to use them in the class. The first part of the training referred to the C implementation of NNF (negation normal form). This included the demonstration of the program on some examples, followed by presentation of the principles and the implementation of the representation of propositional formulae in tree form, and the linear transformation in NNF by traversing this tree.

The second part of the training was dedicated to the tools implemented in the Mathematica system. After a general presentation of the principles and the relevant characteristics of the system, for every tool the corresponding demo was followed by the description of the implementation. The following tools were presented: syntax adaptation, truth table, normal form, resolution, DPLL, and Chaff.

The third part of the training consisted in the description and usage of the Theorema system. The principles of the Theorema system and of its usage have been presented in detail by the main author of Theorema 2.0 (Windsteiger 2014).

After that we approached the implementation and usage of a special prover for sequent calculus in propositional logic, followed by the illustration of the use of the Theorema system in the theory of lists and of binary trees for: designing algorithms, testing them, transforming them automatically, and synthesizing sorting methods.

3.2. Training in SAT

The goal of the training held at the Hungarian partner was to learn how to use SAT solvers to teach basic logic concepts such as satisfiability, logical contradiction, etc. We studied how to represent a conjunctive normal formula (CNF) as a set, as an inequality, as a matrix, as well as a text in DIMACS file format. Each representation has its own strength, for example the set based representation is used in the articles. While the time complexity of the general SAT problem is O(pow(2,n)), some restricted SAT problems are polynomial or even linear. We studied the 2-SAT problem (Aspvall 1979), the Horn-SAT problem (Dowling 1984), which are well known restricted SAT problems. Then we studied the so called Black-and-White SAT problem (Biró 2018), which is suitable to represent a directed graph as a CNF formula. We studied also a new SAT solver, called CSFLOC (Kusper 2018), which is suitable to solve large Black-and-White SAT problems. Then we studied the Bounded Model Checking (BMC) problem (Biere 1999), which can represent a time depended system as a SAT problem by introducing a time bound. We studied how to use Python programming language to call a SAT solver, and how to represent some well known BMC problems. Finally, we studied contract based programming (Meyer 1991), where each method has a pre- and post-condition. The contract says that if the caller fulfills the pre-condition of the called method, then the called method guarantees that its post-condition will be true. We learned how to add these logical formulas as an assert statement to Java programs.

3.3. Training in SMT

This training organized by the German partner was dedicated to SMT solvers (Kröning 2008), which perform the automated check of quantifier-free real and integer arithmetic formulas for satisfiability. Thanks to their wide functionality and impressive efficiency, they enjoy a wide variety of application areas. However, their practical usage is not trivial. Surely, they can be used as black-box, but when we need to solve hard problems, the problem encoding needs to be done in a smart way, adapted to the working mechanisms of SMT solvers in general, as well as the employed decision procedures. Therefore, in our SMT training we explained the most widely used decision procedures for real algebra, some aspects of their implementation in SMT solvers and the usage of these tools for solving practical problems.

For the theoretical part, we covered:

- the traditional eager approach, transforming logical formulas over some suitable theories into propositional logic;
- lazy SMT solving, using a SAT solver to analyze the Boolean structure of the problem, with theory solvers as back-ends to check consistency in the theory,
- model-constructing satisfiability calculus (MCSAT) (de Moura 2013), which uses the ideas of DPLL+CDCL SAT solving (Davis 1961, Marques Silva 1999) and transfers them also to the theory.

For practical aspects, we introduced SMT-LIB, the standard input language for SMT solvers, and looked at two SMT solvers in more detail: Z3 (de Moura 2008), Microsoft's popular SMT solver and our own tool SMT-RAT (Corzilius 2015). Z3 is easy to install and comes with APIs for Python and C++. In the training, we use both SMT-LIB inputs as well as the Python interface to demonstrate how to use Z3. We have selected SMT-RAT as a second solver because it provides nearly all available complete as well as incomplete real-algebraic decision procedures and their strategic combination. Besides the traditional DPPL(T) approach, both Z3 and SMT-RAT offer also MCSAT-based checks.

3.4. Training in Problem Based Learning

The training was organized physically by the academic staff from the partner institution in Romania and was centered on PBL (Problem Based Learning) and PCL (Project Centered Learning) (Dolmans 2005), which consist in a collection of student centered pedagogical techniques that help students to develop advanced thinking skills. The goal was to allow participants to learn novel, student oriented teaching methods, which also help them in designing teaching material that is more motivating for students.

The training consisted in sessions about PBL student centered approaches - the approach to curriculum and pedagogy. These included general aspects of PBL: what is PBL, the history of PBL, what is PCL, the history of PCL, how these approaches help in developing skills, character qualities and knowledge, maintenance of PBL, why PBL is an effective teaching method. Moreover, we discussed the differences between TL - Traditional Learning and PBL, memory and learning principles, the CCCS (Contextual, Constructive, Collaborative, Self directed), how to identify the needs for creating the context of CCCS learning environment. We also presented the experiences with PBL and

PCL at some universities from America and Europe, how teachers become tutors, how tutorial meetings are organized, how to design the teaching material oriented in real-life problems. From the practical point of view we looked at case studies and analysis on "A Model of Teaching", Bloom's Taxonomy, concrete exercises in PBL style and in PCL style; the reasons to use PBL/PCL as alternative teaching methods, what are the main challenges in implementing PBL/PCL as new approaches, and how can PBL/PCL be adapted to a conventional educational system, how to apply/adapt PBL/PCL in our universities.

3.5. Training in Coq

This was the last of five training sessions and it was presented physically by academic staff from the partner institution from France.

The goal was to introduce the participants to the use of the Coq proof assistant (Coq 2022) to model and verify algorithms. The participants were supposed to have no prior experience with Coq but some basic knowledge of propositional and first-order logic, and functional programming.

The training consisted of 8 lectures that lasted between 50 and 100 minutes each. The introduction part from Lecture 1 gave the motivations for certifying critical software, then presented the certification flow in four steps: i) the definition of data structures, ii) the definition of algorithms in terms of a collection of functions, iii) the definition of properties to be satisfied by the algorithms by the means of predicates, and iv) the proof of the properties. During Lecture 2, it was shown a basic usage of Coq for proving statements about propositional and first-order logics. Lecture 3 gave an overview of the basic proof strategies and tactics in Coq, while Lecture 4 introduced some examples of proofs about programs using recursive functions. Another way to write algorithms is by using inductive predicates, as shown in Lecture 5. The inductive predicates can also be used in the definitions of properties. Lectures 6 and 7 presented different proof techniques adapted to reason on properties involving inductive predicates. The last lecture detailed different techniques for the definition and manipulation of recursive functions and recursive inductive predicates during the proof process, as well as to the definition and usage of explicit induction schemas issued from these recursive definitions. The content of the lectures, the Coq environment as well as the exercises related to them have been available online, via a web browser.

The participants had an overview of the main features of Coq by experimenting with many examples in a convenient way, by the means of the jsCoq web interface (Gallego Arias 2016).

4. Dissemination activities

4.1. The ARC International Summer School on Computational logic

The summer school will be organized physically in July 2022 by academic staff from the partner institution in Austria. Students of all levels: Bachelor, Master, and PhD were selected, based on their applications, from all the partner institutions to participate to the summer school. The goal of the summer school is that students learn Computational Logic based on the ARC book (Computational Logic: A Practical Approach) produced in the ARC project and use the tools developed in Theorema, Coq, Satisfiability Modulo Theories, and SATisfiability solvers in practical exercises based on PBL principles.

There will be 5 full days of courses taught by the authors consisting in:

- a presentation of the basic notions of Mathematical Logic, an overview of the proof methods, the main principles of program verification, exercises based on the Theorema system;
- For SAT/SMT solving, the training covers both theoretical aspects of the solving algorithms (DP, DPLL, CDCL, CSFLOC, DPLL(T), MCSAT) as well as the design and usage of SAT/SMT solvers for checking the satisfiability of logical formulas;
- a presentation of techniques for algorithm synthesis on lists, and binary trees, how to extract different algorithms from proofs, how to explore a theory, exercises using the PBL principle;
- an intensive training in Coq. Basically, this will be a rerun of the presentations done during the `Training in Coq' lectures, but more student-oriented.

The results of the summer school will be that students will learn important notions from the ARC book and they will acquire skills for using developed tools in Theorema, Coq, SMT and SAT solvers in practical exercises based on PBL principles. Participants will receive an attendance certificate indicating the number of hours of courses (equivalent to 3 ECTS credits).

4.2. International Symposium on Automated Reasoning in the Class

The symposium, designed as a multiplier event, will be organized in a hybrid format in July 2022 at the partner institution from Romania. The goal of the symposium is to disseminate the results of the ARC project, mainly to familiarize the participants with the ARC book. There will be 2 full days of sessions consisting in: the history and the purpose of the ARC project, the main goal of the ARC book and its innovative approach on teaching, on each chapter the general presentation of the contents, interactive testing of the software and of the exercises, the recommendations and best practice learning scenarios, as well as the possibilities for future cooperation.

5. Conclusions

This project creates the possibility to significantly improve the teaching of computational logic by exploiting in a cross fertilization manner the experiences of 5 excellent computer science departments from European universities. Furthermore, it shows how to use automated theorem prover tools in the classroom.

Currently most of the objectives of the project have been achieved: training of 45 teachers in computational logic related areas, creating the context of interaction and exchange of good practices (in new and innovative pedagogical skills, including multidisciplinary approaches, new curriculum design) of teachers with experts in the field, providing a wider variety of courses to lifelong learning students, linking education with research and innovation, designing and developing learning-outcomes curricula that meet the learning needs of students and also being relevant to the labor market, giving access to licensed professional software for more than 1000 students and 20 teachers, and increasing the visibility of the Erasmus+ activities for the general European public.

The partial results of ARC book were used in more than 15 lectures throughout the project lifetime. Around 240 students participated to the evaluation process. They gave us feedback to the content of the lectures and to the pedagogical teaching methods that we applied. The content of the book will be used in more than 20 lectures at the partner universities. Moreover, novel lectures (that use the material of the book and the developed tools) have been introduced in the curriculum at the partner universities.

The pandemy had a significant impact on the implementation of this project, which was initially designed as an opportunity to collaborate mostly in presence meetings, both among the educators from the partners, as well as with the students. As the travel restrictions in Europe started 5 months after the project start, we extended the project period with 11 months and we rescheduled the in-presence activities for later dates, with the hope that the pandemic will end. However, as the pandemic continued we had to switch to online mode for the training events Theorema and SMT, while the other three have been organized in hybrid mode, but still with the presence of most of the participants, taking advantage of the short periods of lifting of travel restrictions during the pandemics. More importantly, the materials for the lectures were developed for online lectures also, for instance the student exercises that are usually performed at the blackboard have been redesigned using new implemented software that allows to simulate them in online mode. This in fact facilitated the cooperation between the distant partners. For instance, we adapted the system infrastructure such that the software available at the coordinator site (with professional licenses acquired during the project) could now be used remotely by the students from all the partners. Thus, the educators had access to the online exercises during the classes, while the students had independent access between the classes for individual training and for solving part of their homeworks.

The cooperation between 5 different partners with complementary expertize and different teaching methods and environments, also in a pandemic situation, raised various and mostly unexpected challenges that we finally managed to overcome, but this actually raised the quality of our results.

References

Project (2019-2022). www.arc.info.uvt.ro

Aspvall, B., Plass, M. F., Tarjan, R. E. (1979). A Linear-Time Algorithm For Testing The Truth Of Certain Quantified Boolean Formulas. *Information Processing Letters* 8 (3) (pp. 121-123). https://doi.org/10.1016/0020-0190(79)90002-4

Barthe, G. and Stratulat, S. (2003). Validation of the JavaCard Platform with implicit induction techniques. In *RTA'03: Rewriting Techniques and Applications* (pp. 337–351), Springer LNCS 2706. https://doi.org/10.1007/3-540-44881-0_24

Biere, A., Heule, M. van Maaren, H., Walsch, T. (eds) (2009). Handbook of Satisfiability. *Volume 185 of Frontiers in Artificial Intelligence and Applications*, IOS Press. https://dl.acm.org/doi/10.5555/1550723

Biere, A., Cimatti, A., Clarke, E., Zhu, Y (1999). Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, Springer LNCS 1579. https://doi.org/10.1007/3-540-49059-0_14

Biró, Cs., Kusper, G. (2018). Equivalence of Strongly Connected Graphs and Black-and-White 2-SAT Problems, *Miskolc Mathematical Notes,* Vol. 19 (2). (pp. 755-768). https://doi.org/10.18514/MMN.2018.2140

Buchberger, B., Jebelean, T., Kutsia, T., Maletzky, A., Windsteiger, W. (2016). Theorema 2.0: Computer-assisted natural-style mathematics. *Journal of Formalized Reasoning* 9(1) (pp. 149-185). https://doi.org/10.6092/issn.1972-5787/4568

Cook. S. A. (1971). The Complexity of Theorem-Proving Procedures, *Proceedings* of *STOC'71* (pp. 151-158). https://doi.org/10.1145/800157.805047

Coq (2022). *The Coq reference manual*. INRIA. www.coq.inria.fr.

Corzilius, F., Kremer, G., Junges, S., Schupp, S., Abraham, E. (2015). SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. *In Proceedings of SAT'15: 18th International Conference on Theory and Applications of Satisfiability Testing* (pp. 360-368), Springer. https://doi.org/10.1007/978-3-319-24318-4_26

Davis, M., Logemann, G., Loveland, D. (1961). A machine program for theorem proving. *Communications of the ACM*. 5(7): 394–397. https://doi.org/10.1145/368273.368557

Dowling, W. F., Gallier J. H. (1984). Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *J. of Logic Programming*, 1(3), (pp. 267-284). https://doi.org/10.1016/0743-1066(84)90014-1

Meyer, B. (1991). Design by Contract. In *Advances in Object-Oriented Software Engineering*, (pp. 1–50), Prentice Hall.

de Moura,L. M., Bjørner, N. S. (2008). Z3: An efficient SMT solver. In *TACAS'08: Tools and Algorithms for the Construction and Analysis of Systems* (pp. 337-340), Springer LNCS 4963. https://doi.org/10.1007/978-3-540-78800-3_24

de Moura, LO. M., Jovanovic, D. (2013). A model-constructing satisfiability calculus. In *VMCAI'13: Verification, Model Checking, and Abstract Interpretation* (pp 1-12), Springer LNCS 7737. https://doi.org/10.1007/978-3-642-35873-9_1

Dolmans, D. H. J. M., De Grave, W., Wolfhagen, I. H. A. P. , Van Der Vleuten, C. P. M. (2005). Problem-based learning: future challenges for educational practice and research. *Mediac Education*, vol. 39, no. 7. (pp. 732 – 741). https://doi.org/10.1111/j.1365-2929.2005.02205.x

Drămnesc, I., Jebelean, T. (2021-a). AlCons: Deductive synthesis of sorting algorithms in Theorema. In *ICTAC'21: Theoretical Aspects of Computing* (pp. 314-333), Springer LNCS 12819. https://doi.org/10.1007/978-3-030-85315-0_18

Drămnesc, I., Jebelean, T. (2021-b). Synthesis of sorting algorithms using multisets in Theorema. *Journal of Logical and Algebraic Methods in Programming*, vol. 119, 100635, Elsevier. https://doi.org/10.1016/j.jlamp.2020.100635

Eén, N., Sörensson, N. (2004). An Extensible SAT-solver. In *SAT 2003: Theory and Applications of Satisfiability Testing.* Springer LNCS 2919. https://doi.org/10.1007/978-3-540-24605-3_37

Harrison, J. (2009). *Handbook of Practical Logic and Automated Reasoning* (pp. 84–90). Cambridge University Press. ISBN 978-0-521-89957-4.

Kröning, D., Strichman, O. (2008). *Decision Procedures: An Algorithmic Point of View*. Springer. https://link.springer.com/book/10.1007/978-3-540-74105-3

Kusper, G., Biró, Cs., Iszály, Gy. B. (2018). SAT solving by CSFLOC, the next generation of full-length clause counting algorithms. In *IEEE International Conference on Future IoT Technologies*. (pp. 1-9), IEEE, https://doi.org/10.1109/FIOT.2018.8325589

Stratulat, S. (2015). Mechanically certifying formula-based Noetherian induction reasoning. *Journal of Symbolic Computation*, volume 80, part I. (pp. 209 – 249), https://doi.org/10.1016/j.jsc.2016.07.014

Stratulat, S. (2020). SPIKE, an automatic theorem prover -- revisited. In *SYNASC 2020: Symbolic and Numeric Algorithms for Scientific Computing* (pp. 93–96), IEEE, https://doi.org/10.1109/SYNASC51798.2020.00025

Rusinowitch, M., Stratulat S., and Klay, F. (2003). Mechanical verification of an ideal incremental ABR conformance algorithm. *Journal of Automated Reasoning*, vol. 30, no. 2. (pp. 153–177). https://doi.org/10.1023/A:1023251327012

Gallego Arias, E. J., Pin, B., and Jouvelot, P. (2016). jsCoq: Towards hybrid theorem proving interfaces. In *UITP'16: User Interfaces for Theorem Provers* (pp. 15–27), Electronic Proceedings in Theoretical Computer Science (EPTCS) 239. https://doi.org/10.48550/arXiv.1701.07125

Marques Silva, J. P., Sakallah, K. A. (1999). GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers* 48(5). (pp. 506-521). https://doi.org/10.1109/12.769433

Moskewicz, M., Madigan, C., Zhao, Y., Lintao, Z., and Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *DAC'01: Design Automation Conference* (pp. 530–535). https://doi.org/10.1145/378239.379017.

Windsteiger, W. (2014). Theorema 2.0: A system for mathematical theory exploration. In *ICMS 2014: International Congress on Mathematical Software*. Springer LNCS 8592. (pp. 49–52). https://doi.org/10.1007/978-3-662-44199-2_9

Wolfram, S. (2003). *The Mathematica Book*. Wolfram Media Inc. ISBN: 978-1-57955-022-6