Experiments with Automated Reasoning in the Class

Isabela Drămnesc[1][0000−0003−4686−2864], Erika Ábrahám[4][0000−0002−5647−6134], Tudor Jebelean[2][0000−0002−2247−2151], Gábor Kusper[5][0000−0001−6969−1629], and Sorin Stratulat[3][0000−0002−2124−8246]

[1] West University of Timişoara, Romania
isabela.dramnesc@e-uvt.ro
[2] RWTH Aachen University, Germany
abraham@cs.rwth-aachen.de
[3] Johannes Kepler University, Linz, Austria
Tudor.Jebelean@jku.at
[4] Eszterhazy Karoly Catholic University, Eger, Hungary
gkusper@aries.ektf.hu
[5] Université de Lorraine, CNRS, LORIA, 57000, Metz, France
sorin.stratulat@univ-lorraine.fr

Abstract. The European *Erasmus+* project *ARC – Automated Reasoning in the Class* aims at improving the academic education in disciplines related to Computational Logic by using Automated Reasoning tools. We present the technical aspects of the tools as well as our education experiments, which took place mostly in virtual lectures due to the COVID pandemics. Our education goals are: to support the virtual interaction between teacher and students in the absence of the blackboard, to explain the basic Computational Logic algorithms, to study their implementation in certain programming environments, to reveal the main relationships between logic and programming, and to develop the proof skills of the students. For the introductory lectures we use some programs in C and in Mathematica in order to illustrate normal forms, resolution, and DPLL (Davis–Putnam–Logemann–Loveland) with its Chaff version, as well as an implementation of sequent calculus in the Theorema system. Furthermore we developed special tools for SAT (propositional satisfiability), some based on the original methods from the partners, including complex tools for SMT (Satisfiability Modulo Theories) that allow the illustration of various solving approaches. An SMT related approach is natural-style proving in Elementary Analysis, for which we developed and interesting set of practical heuristics. For more advanced lectures on rewrite systems we use the Coq programming and proving environment, in order on one hand to demonstrate programming in functional style and on the other hand to prove properties of programs. Other advanced approaches used in some lectures are the deduction based synthesis of algorithms and the techniques for program transformation.

Keywords: Automated Reasoning · Computational Logic · Computer Aided Teaching.

1 Introduction

The international Erasmus+ European Project *ARC - Automated Reasoning in the Class*, running from 2019 to 2022, is a strategic partnership between 5 prestigious universities from Austria, France, Germany, Hungary, and Romania. This consortium of universities represented by the authors is developing the ARC project, which aims at improving the academic education in disciplines related to Computational Logic by using Automated Reasoning tools.

The activities of the project include: the development of the intellectual output as the ARC book that consists in advanced and motivating teaching and learning material dedicated to Computational Logic, five international trainings for academic staff (one at each partner institution), training of more than 5000 students over 8 semesters, an international summer school for students to learn important notions from the ARC book and how to use the developed tools, as well as an international symposium on ARC for academic staff outside of the partnership for disseminating the final results of the project.

The ARC book contains course material that corresponds to the actual and planned lectures at the partner institutions, accompanied and synchronized with interactive exercises and demonstrations using the software tools developed by the partners especially for the purpose of this book, as well as best-practice recommendations to run the lectures based on advanced pedagogical principles as Problem Based Learning [18]. The students addressed are at all levels: bachelor, master, and PhD. In general there are no lecture prerequisites, when necessary the advanced lectures start with an introduction to the needed notions and algorithms.

The purpose of this paper is to present the most important technical aspects of the tools as well as some aspects of our education experiments, which are also somewhat different from the usual ones because they took place in the period of the COVID pandemics, thus mostly in virtual lectures. The tools are used for several education goals:

- to support the interaction between the teacher and the students in a virtual environment, in particular in the absence of the blackboard or other physical devices;
- to illustrate certain computational logic algorithms that are implemented by some of these tools;
- to demonstrate the implementation principles of mathematical logic notions and methods in certain programming environments;
- to help students understand various relationships between logic and programming, in particular the logical basis of computation, program verification, and algorithm synthesis;
- to develop the proof skills of the students and to reveal the relationship between human proving and automated reasoning.

For the theoretical introductory lectures related to the basics of Mathematical Logic we use some programs in C and in Mathematica [50] in order to illustrate

basic operations like: simple representation and parsing of formulae, transformation into normal form, resolution, and DPLL (Davis–Putnam–Logemann–Loveland [15]), in particular with an animated graphical interface for explaining the Chaff [35] version of the DPLL algorithm.

For illustrating proof methods in sequent calculus [8] we present a tutorial implementation of this method in the Theorema system [7,49,45], including and original version of sequent calculus that uses unit propagation.

We developed special tools for SAT (propositional SATisfiability [29]), some based on the original methods from some of the partners. The name of the new SAT solver is CSFLOC, which states for Count Subsumed Full-Length Clauses [34]. This solver is an iterative version of the inclusion-exclusion principle, which is used to solve the #SAT problem (that is the problem of counting the solutions of a SAT instance [6]). Thus, unusually, we use idea from the field of #SAT to solve SAT, that is we confront the students with a novel specific approach to this problem.

Furthermore, we address SMT (Satisfiability Modulo Theories [5]) by presenting the principles and the usage of two main tools for SMT solving: *Z3* (a state-of-the-art solver used in numerous applications) and SMT-RAT (the solver developed at the partner institution in Aachen), with the latter having the advantage of being able to present and discuss in detail the most important implementation issues.

An SMT related approach is natural-style proving in Elementary Analysis (the so called ϵ δ proofs for statements with alternating quantifiers), for which we developed and interesting set of heuristics that are very useful in practice, both for training the natural-style human proving, as well as for proof automation.

For more advanced lectures on rewrite systems we used the Coq [44] programming and proving environment, in order on one hand to design programs in functional style and on the other hand to prove their properties. Users interact with Coq via a web interface that mixes lecture content with Coq code that does not require the local installation of the Coq distribution.

Another advanced approach that is used in some of the lectures is the deduction based synthesis of algorithms and special programming transformation techniques that increase the efficiency of algorithms by transforming logical based (pattern matching) programs into functional ones and even into tail recursive and subsequently into loop-based imperative programs.

2 Tools

The tools developed for the project are available online [46] to all partners and to any other interested institutions.

The tools presented in sections 2.1 to 2.3 have been used in teaching mainly for the lectures *Automated Theorem Proving*, *Mathematical Logic*, *Automated Reasoning*, *Computational Logic*, *Algorithm Synthesis*, and *Mathematical Theory Exploration*.

2.1 Automated Reasoning in Theorema

The Theorema system is a software system based on Mathematica which offers a flexible and intuitive user interface, allows the construction of theories, the development of automated and interactive proofs in natural style (that are similar to human proving), the development of provers specific to several domains, as well as the direct execution of algorithms that are synthesized by proving.

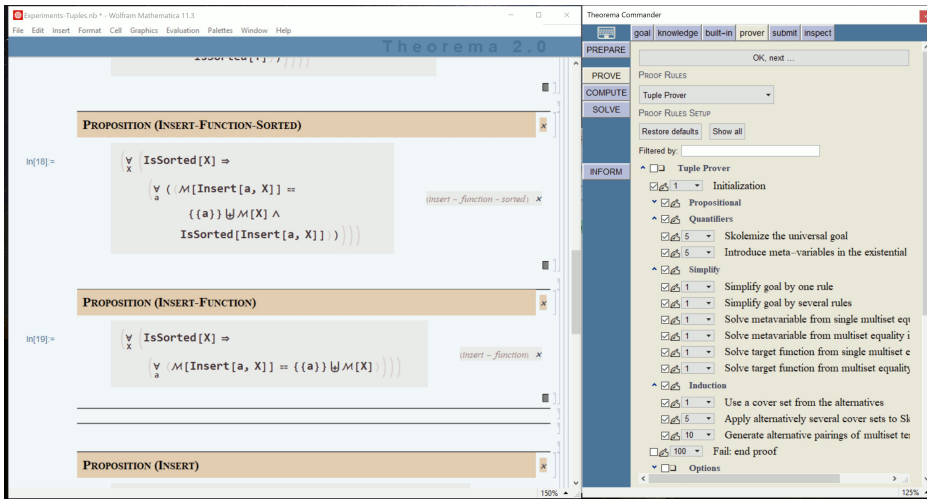

Fig. 1. Theorema notebook and Theorema Commander

In figure 1 we can see two windows: the left hand side represents the Theorema notebook where the user introduces the knowledge base (formulae as propositions, conjectures, definitions, algorithms), and the right hand side is the Theorema Commander in Prove mode, from where the user can choose the prover to be applied, can select/deselect inference rules, can change the priority of inference rules, can set the proof depth and time, can choose how to generate the proof (automatically or interactively), can choose to show the simplified version of the proof (that includes only the proved branches), etc.

The proofs in Theorema are developed as proof trees of proof situations (assumptions and one goal). The proof is extended at every transformation of a proof situation into other proof situation(s) by applying a certain inference rule. The generated proof is shown in a separate window displaying the proof information and in the commander it is displayed the generated proof tree as illustrated in Figure 2.

The user has the possibility of computing with definitions and algorithms. Figure 3 illustrates some computations with an algorithm that has been previously synthesized by proving.

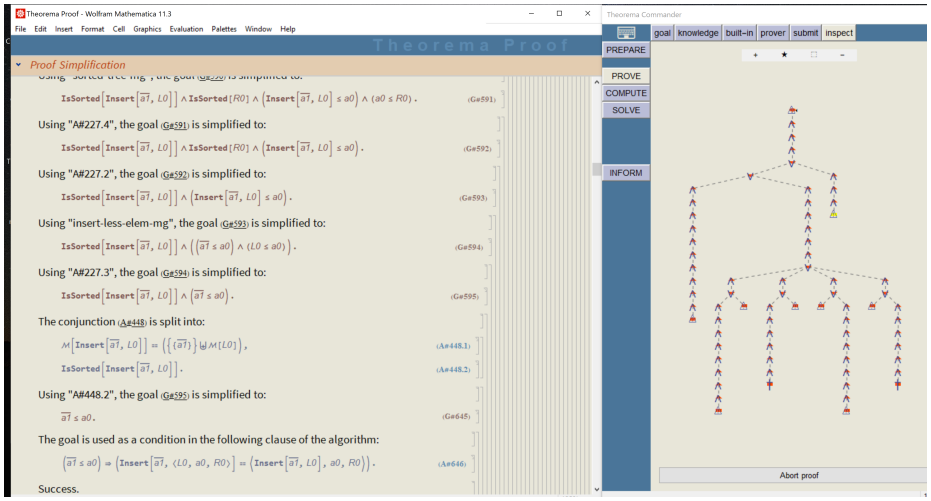

Fig. 2. Theorema proof and the generated proof tree

As a support for teaching proof methods in sequent calculus we developed a Theorema prover that implements sequent calculus, including an original version that uses unit propagation. This prover is used on one hand for presenting automated proofs in natural style that are based on sequent calculus, and on the other hand to study the implementation of such provers in Theorema, namely it shows that, in essence, one only has to specify the inference rules as rewriting steps.

2.2 Synthesis and Transformation of Algorithms

The Algorithm Synthesis Problem consists in: starting from the specification of a problem, given as a pair of input and output conditions, how can we automatically discover an algorithm which satisfies the specification? The concern of algorithm synthesis is to develop methods and tools for mechanizing and automatizing (parts of) the process of finding an algorithm that satisfies a given specification.

As a support for teaching subjects related to algorithm synthesis by proving and mathematical theory exploration we developed a prover [20,19] in the Theorema system that synthesizes a large number of algorithms operating on lists and on binary trees, including sorting algorithms as rewrite programs. A prover in Theorema consists of a collection of inference rules (as rewrite rules). In these related lectures we discuss: the use of multisets in the problem specification (as an easier way to express that two objects have the same elements), we analyse the inference rules, we study the use of several techniques for synthesizing algorithms, the use of different knowledge base (different techniques and different knowledge base can lead to the synthesis of different algorithms), the principles

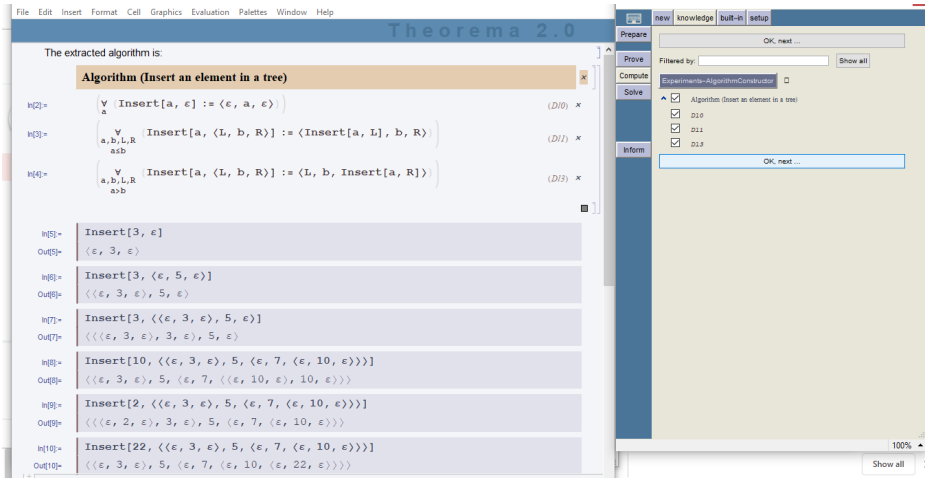

Fig. 3. Theorema Compute

of extracting an algorithm from a proof, as well as the principles of systematic exploration of theories. In fig. 1 we can see on the LHS the specification of the function *Insert* that inserts at the appropriate place an element into a sorted list and on the RHS the list of the prover inference rules. In fig. 2 we can see a part of the synthesizing proof for this algorithm (LHS) and the structure of the proof tree (RHS). The computation with the synthesized algorithm is presented in fig. 3.

As a support for teaching subjects related to algorithm transformations, we developed a Theorema tool [21] that transforms recursive algorithms into more efficient ones, which are then transformed into tail recursive, then into functional programs, and then into their corresponding imperative version. In these lectures we expose the systematic principles for transforming an algorithm into the tail recursive version, how to improve the efficiency using a special flag for avoiding the unnecessary recursion, and then how to transform the algorithm into its functional and iterative version.

2.3 Sample Implementations

Computational Logic in C. This teaching experiment presents the design and the implementation in C language of an algorithm that transforms propositional formulae into Negation Normal Form (NNF – only conjunctions and disjunctions, with negation occurring only inside literals). Furthermore the program generates commands for a TeX interface that displays the formula tree and its evolution. This is in fact a starting point for the implementation and visualization of various rewriting and proving methods, as it demonstrates:

– the reading and writing of propositional formulae in reversed Polish notation,

- the tree representation of formulae,
- traversal of the tree in recursive but also in imperative way (for efficiency),
- the efficient transformation into NNF in linear time by destructive update of the formula tree.

The program also generates TeX commands that are used to generate the sequence of pages for the visualization of the algorithm effect.

Mathematical Logic in Mathematica. The Mathematica Computer Algebra system has a programming language that is based on logical rewriting of terms using a powerful pattern matching concept, therefore one can use it for demonstrating the possibility of programming by using essentially logical formulae. Its usage in the class has a double advantage: on one hand the implementation is relatively easy because low level functions like parsing, term matching, term rewriting, etc. are already available, and on the other hand most of the program text is very similar to logical formulae.

For the lectures related to our project we developed various functions implementing: truth table computation (interactive and also automatic), formula rewriting into Negation Normal Form (NNF), Conjunctive Normal Form (CNF), and Disjunctive Normal Form (DNF), the DPLL method (see the definition in subsection 2.5), and finally the Chaff implementation of DPLL [35] – see Fig.4.

On one hand the students have been able to use these demo programs in order to check practically the properties studied in the theoretical part of the lecture, and on the other hand they could understand the implementations and also the close relationship between the logic of term rewriting and programming.

Proof Heuristics in Elementary Analysis. We developed several heuristic techniques for such proofs:

- the S-decomposition method for formulae with alternating quantifiers,
- quantifier elimination by Cylindrical Algebraic Decomposition,
- analysis of terms behavior in zero,
- bounding the ϵ-bounds,
- semantic simplification of expressions involving absolute value,
- polynomial arithmetic and solving,
- usage of equal arguments to arbitrary functions,
- reordering of proof steps in order to insure the admissibility of solutions to meta-variables.

Some of these combine logic with domain specific methods (which is basically equivalent to SMT solving). Moreover, in our teaching context they are very useful for producing proofs that can be easily understood by human readers. These techniques allow to produce natural-style proofs for many interesting examples [27], like convergence of sum and product of sequences, continuity of sum, product and compositions of functions, etc. that have been discussed with the students.

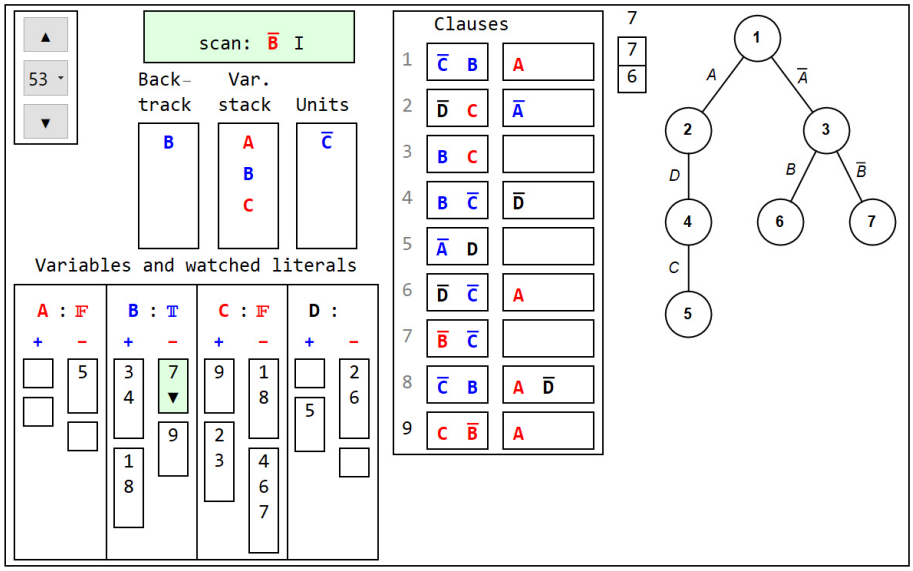

Fig. 4. Visualization of the Chaff implementation of DPLL.

We presented these heuristics in the class, both in lectures related to automated reasoning, but also in lectures dedicated to the training of students in formal mathematical techniques (formalization of mathematical notions, formal proving, etc.).

2.4 The Coq Theorem Prover

The Coq proof assistant can be seen as a combination of:

1. a simple but extremely expressive, programming language, and
2. a set of tools for stating logical assertions (including assertions about the behaviour of programs) and giving evidence of their truth.

Started in 1984 by Thierry Coquand and Gérard Huet, it has been extended in 1991 with inductive types by Christine Paulin. Since then, more than 40 people contributed to Coq. It has been used to certify non-trivial theorems, as the 'four colour' and 'odd order' theorems, and applications, as a C compiler. In 2013, it received the prestigious ACM Software System award.

We developed a Coq tutorial based on the material presented at an INRIA summer school [26]. It consists in the following parts, also publicly accessible [11]:

1. Programming with natural numbers and lists (lecture, exercises)
2. Propositions and predicates (lecture, exercises)

3. Making proofs in Coq (lecture, exercises)
4. Proofs about programs (lecture, exercises)
5. Inductive data types (lecture, exercises)
6. Inductive properties I (lecture, exercises)
7. Inductive properties II (lecture, exercises)
8. Recursive functions in Coq (lecture, exercises)

The links above allow to access the material interactively, by the means of *JsCoq* [2], a Jupyter Notebook interface for Coq.

The JsCoq interface is split in two, with its left (Figure 5) and right (Figure 6) sides.

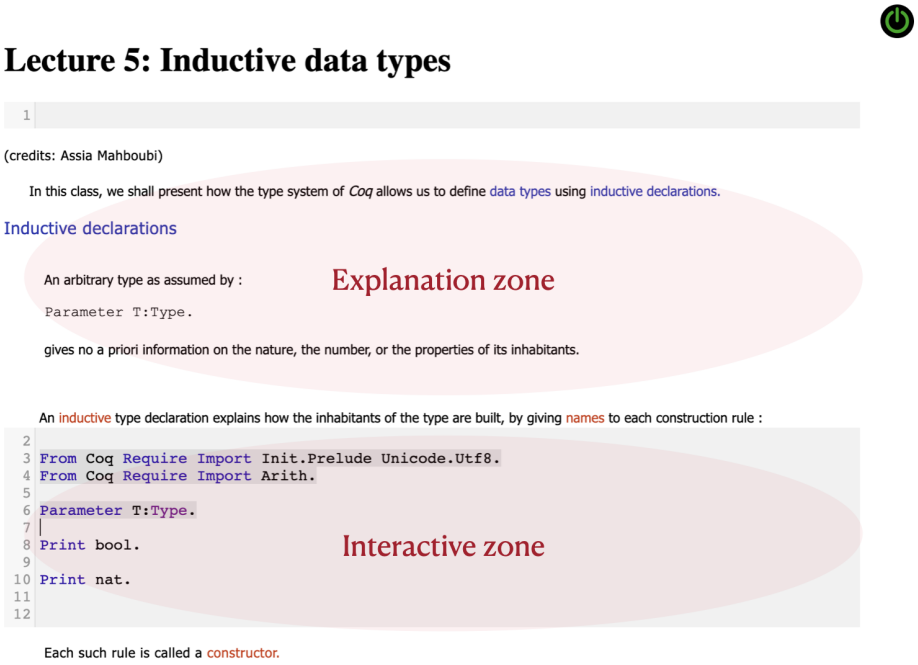

Fig. 5. The left side.

One can distinguish the following zones. On the left hand side:

– The *explanation* zone. It contains the explanations given by the lecturer.
– The *interactive zone*. The user can execute the Coq code interactively. Its content can be modified by the user.

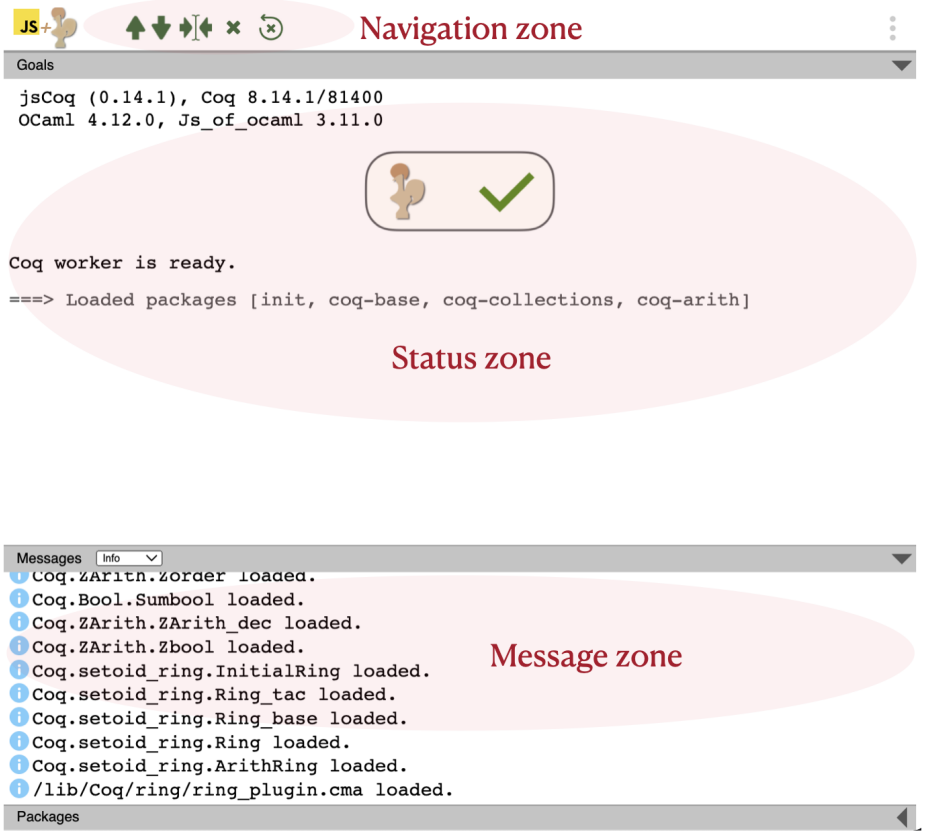

Fig. 6. The right side.

On the right hand side:

- The *navigation* zone. The user can navigate through the Coq code using the *up* and *down* arrows. The executed code is grayed in the navigation zone. The user can also execute the code up to the current position of the cursor in the navigation zone by clicking on the 'face-to-face arrows' button. The *cross* button stops the execution of a Coq command, which is useful when it takes too much time. The Coq session can be reset with the rightmost button.
- The *status* zone. It displays the current status of the prover as well as the current goals during the proof sessions.
- The *message* zone. It shows the messages given by the Coq prover while executing code from the interactive zone.

The tutorial has been taught remotely to Master students in Computer Science at the West University of Timisoara. They appreciated that Coq scripts are

mixed with the related explanations, as well as the way to directly change the Coq scripts, execute them and view the results in the same spot. On the other hand, the web interface does not allow yet to save the current state of the scripts in one step; alternatively, the users can manually copy the blocks of Coq script one by one.

2.5 SAT Solvers

In order to keep close contact with the research and with the industrial community, we used in our practical exercises a very popular file format for SAT solvers, namely the DIMACS format [16]. It represents a boolean variable by its 1-based index. A positive number corresponds to a positive literal, a negative number to a negative literal. Each line is a clause, and each line is terminated by a "0".

A DIMACS CNF file must contain a problem line which starts by *p cnf* which tells the number of variables and the number of clauses information:

```
p cnf #variables #clauses
```

It can contain also comments, which start by the letter "c". An example for DIMACS CNF file:

```
c This is a SAT problem with 3 variables and 5 clauses.
c It is unsatisfiable.
p cnf 3 5
-1 2 0
-2 3 0
1 -3 0
1 2 3 0
-1 -2 -3 0
```

SAT solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) [15] algorithm. This algorithm performs Boolean Constraint Propagation (BCP) and backtrack search, i.e., at each node of the search tree it selects a decision variable and assigns a truth value to it, then steps back when a conflict occurs.

Conflict-driven clause learning (CDCL), see Chapter 4 in [6], is based on the idea that conflicts can be exploited to reduce the search space. If the method finds a conflict, then it analyzes this situation, determines a sufficient condition for this conflict to occur, in form of a learned clause, which is then added to the formula, and thus avoids that the same conflict occurs again.

Besides clause learning, lazy data structures are one of the key techniques for the success of SAT solvers, such as "watched literals" as pioneered in 2001, by the CDCL solver Chaff [35].

Since multi-core architectures are common today, the need for parallel SAT solvers using multiple cores has increased considerably.

In essence, there are two approaches to parallel SAT solving. The first group of solvers typically follow a divide-and-conquer approach. They split the search space into several subproblems, sequential workers solve the subproblems. This

first group uses relatively intensive communication between the nodes. They do for example load balancing and dynamic sharing of learned clauses.

The second group apply portfolio-based SAT solving. The idea is to run independent sequential SAT solvers with different restart policies, branching heuristics, learning heuristics.

In case of the SAT problem we can speak about the so called threshold phenomenon, or phase transition phenomenon, which is the following observation. If we have a uniform random 3-SAT problem (i.e. each clause contains 3 literals) with n boolean variables and m clauses, then around the threshold $m/n = 4.27$ there are difficult problems for the DPLL algorithm based SAT solvers [23]. Under-constraint problems (where $m/n << 4.27$) are usually very easy for SAT solvers. Again, over-constraint problems (where $m/n >> 4.27$) are in general very easy for them. But there is a subclass of over-constraint problems which is very difficult. This is the class of those problems which are over-constraint and minimal unsatisfiable (UNSAT) at the same time. A SAT problem is minimal UNSAT if and only if all of its clauses are needed in the proof that the problem is UNSAT.

For the lectures related to Computational Logic we have developed a SAT solver, called CSFLOC [34], and some problem generators which can generate minimal UNSAT problems [33].

In the lecture *Applied Logic* we studied several SAT solvers, several problem libraries, and even implemented the well-known algorithms using Python, which is a rapid prototyping programming language.

2.6 SMT Solving

Another class of tools for automated reasoning are *Satisfiability Modulo Theories* (*SMT*) *solvers* [5,32]. Also these tools aim at checking the satisfiability of logical formulas in a fully automated manner [6], but in contrast to SAT solvers, they are devoted to formulas from (mostly quantifier-free) first-order logic over different theories. Starting with easier theories (e.g. equality logic with uninterpreted functions) and theories for program verification (e.g. array theory, bit-vector arithmetic), impressive developments have been made also for quantifier-free linear and non-linear arithmetic over the reals and the integers.

SMT solving enjoys an active research community with dedicated conferences [42,39], a SatLive forum [43], and SMT solver competitions [38] between numerous SMT solvers that focus on different theories, algorithms and application areas. One of the main achievements of the community is the SMT-LIB standard input language [3], which is supported by most state-of-the-art SMT solvers. Thus when a problem is encoded in this input language, most SMT solvers (that support the theories used) can be directly feeded with it. Using this standard, a large benchmark library [3] has been collected and is publicly available for tool developers and can be used for competitions. Besides command-line SMT-LIB input, some solvers come with APIs for different programming languages (typically C++ and/or Python) that can be used to specify and solve problems.

For educational purposes, besides giving insights into the theoretical backgrounds, in our project we focussed on two tools: Z3 and SMT-RAT. Both support the SMT-LIB standard and offer a C++ API, whereas Z3 has also an API for Python. We use Z3 to train the *usage* of SMT solvers and SMT-RAT to train *tool development*.

Z3 [36] is one of the most popular, most efficient and easy-to-use SMT solvers that support a large number of different theories. Therefore, Z3 is well suited to train the *usage* of SMT solvers. The tool is accessible under [53]. Extensive documentation and numerous examples are provided, which makes it easy to construct a program for the training.

Firstly, in our project we use Z3 to train how to encode and solve problems using the SMT-LIB standard. The rich variety of Z3-supported theories allows to encode and efficiently solve problems from numerous domains [4]. We used these and other examples from [51] to design encoding exercises.

Secondly, we use the Python API [54] that allows an easy employment of Z3 directly from Python programs. Concrete executable examples are provided under [52].

SMT-RAT (SMT Real-Arithmetic Toolbox) [14,13] is an SMT solver with a focus on solving quantifier-free real algebraic problems. Though competitive, the main aim of SMT-RAT is not to be the fastest, but to offer an optimal platform for *implementing* new SMT solving ideas.

To provide optimal support for extensions, SMT-RAT comes with a solid basic support for arithmetic computations and a clear modular structure. SMT-RAT modules implement different decision procedures that can be connected by user-defined strategies: each module tries to solve its input problem and might consult its backends by delegating sub-problems to them. This way we can e.g. use fast but incomplete procedures and assure completeness by providing a complete backend to them.

The latest release [37] offers, besides some non-arithmetic components, the CaRL [9] library for arithmetic datatypes and basic computations with them, and solver modules for e.g. the simplex and the Fourier-Motzkin variable elimination methods for linear problems, for non-linear real arithmetic different adaptions of the cylindrical algebraic decomposition method [10,30], the virtual substitution method [48,12,1], subtropical satisfiability [22], methods based on Gröbner bases [47,28], as well as interval constraint propagation [24,25] and the branch-and-bound technique [17,31] for finding integer solutions.

Besides installation guide, the SMT-RAT documentation [40] also describes built-in support for benchmarking, debugging and testing.

Teaching SMT solving. Regarding theory, we developed a lecture named *Satisfiability Checking* to convey algorithmic aspects of SMT techniques to students. The lecture covers SAT solving, eager SMT solving for equality logic and

uninterpreted functions as well as for bitvector arithmetic, lazy SMT solving as a framework, and dedicated decision procedures for linear and nonlinear real and integer arithmetic theories. We discuss SMT-adaptions of the Gauss and Fourier-Motzkin variable elimination, the simplex algorithm and the branch-and-bound method for linear arithmetic, and the methods of interval constraint propagation, subtropical satisfiability, virtual substitution and cylindrical algebraic decomposition for non-linear real arithmetic.

To support learning, we developed automatic exercise generators for different algorithms, embedded in a GUI for easy usage. The development of such a training tool is challenging, as the generated exercises should satisfy different quality criteria. For example:

- For each trained algorithm, a large number of exercises of comparable difficulty and time effort should be generated.
- A correct answer should be unique and it should be a good indicator that the student understood the trained algorithm. Numerical answers should be syntactically short and easy to type on the keyboard. For multiple choice answers, the probability to correctly guess the answer should be small.
- Offer different difficulty levels, tips, explanations and a full solution.

Regarding practical aspects and implementation, we made extensive experience using SMT-RAT in teaching, in practical courses as well as in more than 50 Bachelor and Master theses [41] for implementing novel ideas for decision procedures.

In general, we observed that good students are very much attracted to SMT solving, as topics in this area are theoretically challenging, but also practical aspects play an important role for efficient implementations. Several of the students who wrote their thesis in SMT solving have been enrolled in PhD studies afterwards.

3 Conclusion

The lectures from the partner universities that are addressed by the project are: *Mathematical Logic, Automated Reasoning, Computational Logic, Automated Theorem Proving, Techniques for Scientific Work, Functional Programming, Logic Programming, Algorithm Synthesis and Mathematical Theory Exploration, Artificial Intelligence, Satisfiability Checking, Applied Logic, Modeling and Verifying Algorithms in Coq, Satisfiability Modulo Theories.* These took place since the Winter Semester 2019 (thus for 7 semesters) with a total of more than 5000 students. Starting with Summer Semester 2020 the lectures have been gradually switched to online mode. On one hand this created the expected difficulties in the interaction with the students, on the other hand this stimulated the creation of specific approaches, reading material, and tools for ensuring the quality of the lectures and for motivating the students. As now we started to switch back to presence lectures, we can notice that the experience and the material created during the pandemics is still very useful. A powerful help for improving

the teaching is constituted by the various aspects of cooperation between the project partners: besides the possibility to cross-use some material and tools, we also created specific environments in which the students can access computational resources and licensed software remotely at the other partners.

As a general conclusion we can say that the tools and the practical material developed in this project had a significant positive impact on the teaching process, with visible effects on the quality of the student achievements. In particular, in the context of the pandemics, because of the downgrading of the communication between teachers and students, the lack of these tools would have probably lead to a decrease of student motivation, participation, and learning. Evaluation of the project results has been pursued continuously during the project life, including evaluation from the point of view of the students involved. In fig. 7 we present the student answers (percentage) to some of the most relevant issues about the lectures.

Issue	Very good	Good	Neutral
Interesting	26	48	19
Well defined goals	35	50	11
Well structured	43	42	11
Helpful materials	38	42	16
Helpful examples	40	45	11
Helps professional development	21	40	25
Teaching methods	22	47	20

Fig. 7. Some evaluation results (percentage of student answers).

Further work on the objectives of this project is, among other reasons, strongly motivated by the fact that we come now back to in–presence lectures, and also with a strong uncertainty about possible fall back to virtual mode. Thus, on one hand we can now adapt, test, and evaluate the methods that have been used virtually in order to use them in a classical environment, and on the other hand we have to keep improving the virtual techniques in order to be prepared for future pandemic waves.

Acknowledgements. This work is co-funded by the Erasmus+ Programme of the European Union, project ARC: Automated Reasoning in the Class, 2019-1-RO01-KA203-063943.

References

1. Ábrahám, E., Nalbach, J., Kremer, G.: Embedding the virtual substitution method in the model constructing satisfiability calculus framework. In: Proc. of SC-square'17. CEUR Workshop Proceedings, vol. 1974. CEUR-WS.org (2017). `https://doi.org/urn:nbn:de:0074-1974-4`, `http://ceur-ws.org/Vol-1974/EAb.pdf`

2. Arias, E.J.G., Pin, B., Jouvelot, P.: jsCoq: Towards hybrid theorem proving interfaces. In: Proc. of the 12th Workshop on User Interfaces for Theorem Provers, UITP. EPTCS, vol. 239, pp. 15–27 (2016)
3. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org (2016)
4. Barrett, C., Kroening, D., Melham, T.: Problem solving for the 21st century: Efficient solvers for satisfiability modulo theories. Knowledge Transfer Report 3, London Mathematical Society and Smith Institute for Industrial Mathematics and System Engineering (2014)
5. Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 26, pp. 825–885. IOS Press (2009)
6. Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
7. Buchberger, B., Jebelean, T., Kutsia, T., Maletzky, A., Windsteiger, W.: Theorema 2.0: Computer-Assisted Natural-Style Mathematics. Journal of Formalized Reasoning **9**(1), 149–185 (2016). https://doi.org/10.6092/issn.1972-5787/4568
8. Buss, S.R.: An introduction to proof theory. In: Handbook of proof theory, pp. 31–35. Elsevier (1998)
9. CArL: Project homepage. https://github.com/smtrat/carl
10. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Automata Theory and Formal Languages. LNCS, vol. 33, pp. 134–183. Springer (1975)
11. https://members.loria.fr/SStratulat/files/MVA/index.html
12. Corzilius, F.: Integrating virtual substitution into strategic SMT solving. Ph.D. thesis, RWTH Aachen University, Germany (2016), http://publications.rwth-aachen.de/record/688379
13. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Abraham, E.: SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. In: Proc. of SAT'15. LNCS, vol. 9340, pp. 360–368. Springer (2015). https://doi.org/10.1007/978-3-319-24318-4_26, https://doi.org/10.1007/978-3-319-24318-4_26
14. Corzilius, F., Loup, U., Junges, S., Ábrahám, E.: SMT-RAT: an smt-compliant nonlinear real arithmetic toolbox - (tool presentation). In: Cimatti, A., Sebastiani, R. (eds.) Proc. of SAT'12. LNCS, vol. 7317, pp. 442–448. Springer (2012). https://doi.org/10.1007/978-3-642-31612-8_35, https://doi.org/10.1007/978-3-642-31612-8_35
15. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (jul 1962). https://doi.org/10.1145/368273.368557, https://doi.org/10.1145/368273.368557
16. http://www.domagoj-babic.com/uploads/ResearchProjects/Spear/dimacs-cnf.pdf
17. Doig, A.G., Land, B.H., Doig, A.G.: An automatic method for solving discrete programming problems. Econometrica pp. 497–520 (1960)
18. Dolmans, D.H.J.M., De Grave, W., Wolfhagen, I.H.A.P., Van Der Vleuten, C.P.M.: Problem-based learning: future challenges for educational practice and research. Mediac Education **39**(7), 732–741 (2005), https://doi.org/10.1111/j.1365-2929.2005.02205.x
19. Dramnesc, I., Jebelean, T.: Synthesis of sorting algorithms using multisets in Theorema. Journal of Logical and Algebraic Methods in Programming **119**(100635) (2020). https://doi.org/10.1016/j.jlamp.2020.100635

20. Dramnesc, I., Jebelean, T., Stratulat, S.: Mechanical Synthesis of Sorting Algorithms for Binary Trees by Logic and Combinatorial Techniques. Journal of Symbolic Computation **90**, 3–41 (2019). https://doi.org/10.1016/j.jsc.2018.04.002
21. Dramnesc, I., Jebelean, T.: Implementation of Deletion Algorithms on Lists and Binary Trees in Theorema. RISC Report Series 20-04, Research Institute for Symbolic Computation, Johannes Kepler University Linz (2020)
22. Fontaine, P., Ogawa, M., Sturm, T., Vu, X.: Subtropical satisfiability. In: Proc. of FroCoS'17. LNCS, vol. 10483, pp. 189–206. Springer (2017). https://doi.org/10.1007/978-3-319-66167-4_11, https://doi.org/10.1007/978-3-319-66167-4_11
23. Friedgut, E., appendix by Jean Bourgain: Sharp thresholds of graph properties, and the -sat problem. Journal of the American Mathematical Society **12**, 1017–1054 (1999)
24. Gao, S., Ganai, M., Ivančić, F., Gupta, A., Sankaranarayanan, S., Clarke, E.M.: Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems. In: Proc. of FMCAD'10. pp. 81–90. IEEE (2010)
25. Herbort, S., Ratz, D.: Improving the efficiency of a nonlinear-system-solver using a componentwise Newton method. Tech. Rep. 2/1997, Inst. für Angewandte Mathematik, University of Karlsruhe (1997)
26. CEA-EDF-INRIA summer school, INRIA Paris-Rocquencourt, Antenne Parisienne, 2011. https://fzn.fr/teaching/coq/ecole11/
27. Jebelean, T.: A heuristic prover for elementary analysis in Theorema. In: CICM 2021. LNAI, vol. 12833, pp. 130–134. Springer (2021)
28. Junges, S., Loup, U., Corzilius, F., Ábrahám, E.: On Gröbner bases in the context of satisfiability-modulo-theories solving over the real numbers. In: Proc. of CAI'13. LNCS, vol. 8080, pp. 186–198. Springer (2013)
29. Knuth, D.E.: The Art of Computer Programming: Satisfiability, Volume 4, Fascicle 6. Addison–Wesley Professional (2015)
30. Kremer, G., Abraham, E.: Fully incremental cylindrical algebraic decomposition. J. Symb. Comput. **100**, 11–37 (2020). https://doi.org/10.1016/j.jsc.2019.07.018, https://doi.org/10.1016/j.jsc.2019.07.018
31. Kremer, G., Corzilius, F., Abraham, E.: A generalised branch-and-bound approach and its application in SAT modulo nonlinear integer arithmetic. In: Proc. of CASC'16. LNCS, vol. 9890, pp. 315–335. Springer (2016). https://doi.org/10.1007/978-3-319-45641-6_21, https://doi.org/10.1007/978-3-319-45641-6_21
32. Kroening, D., Strichman, O.: Decision Procedures: An Algorithmic Point of View. Springer (2008)
33. Kusper, G., Balla, T., Biró, C., Tajti, T., Yang, Z.G., Baják, I.: Generating minimal unsatisfiable SAT instances from strong digraphs. In: SYNASC 2020. pp. 84–92. IEEE Computer Society Press (2020). https://doi.org/10.1109/SYNASC51798.2020.00024
34. Kusper, G., Biró, C., Iszály, G.B.: SAT solving by CSFLOC, the next generation of full-length clause counting algorithms. In: 2018 IEEE International Conference on Future IoT Technologies. pp. 1–9. IEEE Computer Society Press (2018). https://doi.org/10.1109/FIOT.2018.8325589
35. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proc. of the 38th Annual Design Automation Conference. p. 530–535. Association for Computing Machinery (2001). https://doi.org/10.1145/378239.379017, https://doi.org/10.1145/378239.379017

36. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. of TACAS'08. LNCS, vol. 4963, pp. 337–340. Springer (2008)
37. https://github.com/ths-rwth/smtrat/releases
38. https://smt-comp.github.io/
39. https://smt-workshop.cs.uiowa.edu/
40. https://ths-rwth.github.io/smtrat/
41. https://ths.rwth-aachen.de/theses/
42. http://www.satisfiability.org/
43. http://www.satlive.org/smt.html
44. The Coq development team: The Coq Reference Manual. INRIA (2020)
45. https://www.risc.jku.at/research/theorema/software
46. https://arc.info.uvt.ro/?page_id=49
47. Weispfenning, V.: A new approach to quantifier elimination for real algebra. In: Quantifier Elimination and Cylindrical Algebraic Decomposition. pp. 376–392. Texts and Monographs in Symbolic Computation, Springer (1998)
48. Weispfenning, V.: Quantifier elimination for real algebra - the quadratic case and beyond. Appl. Algebra Eng. Commun. Comput. **8**(2), 85–101 (1997)
49. Windsteiger, W.: Theorema 2.0: A System for Mathematical Theory Exploration. In: ICMS'2014. LNCS, vol. 8592, pp. 49–52 (2014). https://doi.org/10.1007/978-3-662-44199-2_9
50. Wolfram Research Inc.: Mathematica, Version 13.0.0, https://www.wolfram.com/mathematica
51. https://ericpony.github.io/z3py-tutorial/guide-examples.htm
52. https://github.com/Z3Prover/doc/tree/master/programmingz3/code
53. https://github.com/z3prover/z3
54. https://theory.stanford.edu/~nikolaj/programmingz3.html