A Heuristic Prover for Elementary Analysis in *Theorema*

Tudor Jebelean

RISC–Linz, JKU
www.risc.jku.at

Abstract. We present a plug-in to the *Theorema* system, which generates proofs similar to those produced by humans for theorems in elementary analysis and is based on heuristic techniques combining methods from automated reasoning and computer algebra. The prover is able to construct automatically natural-style proofs for various examples related to convergence of sequences as well as to limits, continuity, and uniform continuity of functions. Additionally to general inference rules for predicate logic, the techniques used are: the S-decomposition method for formulae with alternating quantifiers, use of 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, usage of equal arguments to arbitrary functions, and automatic reordering of proof steps in order to check the admissibility of solutions to the metavariables. The problem of proving such theorems directly without using refutation and clausification is logically equivalent to the problem of satisfiability modulo the theory of real numbers, thus these techniques are relevant for SMT solving also.

Keywords: Satisfiability Checking · Natural-style Proofs · Computer Algebra · Symbolic Computation · Satisfiability Modulo Theories.

1 Introduction

In this paper we present our results on a class of proof problems which arise in elementary analysis, namely those involving formulae with alternating quantifiers. We implement the following heuristic techniques, which extend our previous work [9, 5, 6]: the S-decomposition method for formulae with alternating quantifiers [8], use of Quantifier Elimination by Cylindrical Algebraic Decomposition [4], analysis of terms behavior in zero, bounding the ϵ-bounds, semantic simplification of expressions involving absolute value, polynomial arithmetic, usage of equal arguments under unknown functions, and automatic reordering of proof steps in order to check the admissibility of solutions to the metavariables.

Our prover, implemented in the frame of the *Theorema* system [3], aims at producing natural-style proofs for simple theorems involving convergence of sequences and of functions, continuity, uniform continuity, etc. The prover does not need to access a large collection of formulae (expressing the properties of the domains involved). Rather, the prover uses techniques from computer algebra in order to discover relevant terms and to check necessary conditions, and only needs as starting knowledge the definitions of the main notions involved. The size of this short paper does not allow an overview of the relevant literature, so we only mention [2], which in contrast to our prover is based mostly on rewriting of logical terms and does not handle alternating quantifiers.

2 Application of Special Techniques

Example: Product of Convergent Sequences. We illustrate the heuristics by the proof of the theorem *"The product of two convergent sequences is convergent"*, which is presented in detail together with other examples and explanations of the techniques in [7]. The proof starts from the definitions of product of two functions and of convergence of a function $f : \mathbb{N} \longrightarrow \mathbb{R}$:

$$\underset{a\in\mathbb{R}}{\exists}\ \underset{\substack{e\in\mathbb{R}\\ e>0}}{\forall}\ \underset{M\in\mathbb{N}}{\exists}\ \underset{\substack{n\in\mathbb{N}\\ n\geq M}}{\forall} |f[n]-a|<e$$

After introducing Skolem constants f_1, f_2 for the arbitrary convergent sequences and expansion of the goal and of the assumptions by the definitions of convergence and of product of functions, the prover is left with two main assumptions and one goal (instances of the formula above), which have parallel alternating quantifiers.

The S-decomposition method. The main structure of the proof (see [8]) is as follows: the quantifiers are removed from the 3 statements in parallel, using a combination of inference steps which decompose the proof into several branches. When the 3 formulae are existential, first introduce the Skolem constants for the assumptions, and then introduce a witness for the goal. The proof branches into: a main branch with the new goal, and secondary branches for proving the subgoals stating that the type and the condition of the existential variable hold for the witness. When the 3 formulae are universal, first introduce Skolem constants for the goal, and then introduce the instantiation terms for the assumptions. Similarly to above, separate secondary branches are created for the type and condition checking of the instantiation terms.

Thus in this proof the prover produces, in order: Skolem constants a_1, a_2, witness $a_1 + a_2$; Skolem constant e_0, instantiation term Min $\left[1, \frac{e_0}{|a_2|+|a_1|+1}\right]$; Skolem constants M_1, M_2, witness Max$[M_1, M_2]$; Skolem constant n_0, instantiation term n_0. (The names are similar to the corresponding variables in the definition.)

At every iteration of the proof cycle one needs a witness for the existential goal and an one or more instantiation terms for the universal assumptions: these are the difficult steps in the proof, for which we use special proof techniques based on computer algebra.

Reasoning about terms behavior in zero: by polynomial arithmetic the prover infers the value of the witness for a by equating all the expressions under the absolute value to zero.

Use of metavariables: the existential variable in the goal (or the universal variable in an assumption) is replaced by a new symbol (metavariable), which is a name for the term (solution of the metavariable) which we need to find. This term is determined later in the proof, and the subgoals stating the type and the condition are checked on the secondary branches. Also, one must ensure that the solution to the metavariable does not contain Skolem constants which are introduced later in the proof. If this condition is not fulfilled, the prover tries to **reorder the steps of the proof**.

Quantifier Elimination is used in order to find the solution of the metavariable in relatively simple situations – as for instance in this proof for $Max[M_1, M_2]$, as described in [1].

Identification of equal terms under unknown functions. This is used for finding the instantiation term n_0.

Since f_1 and f_2 are arbitrary, we do not know anything about their behaviour. In the goal f_1 and f_2 have argument n_0, therefore the prover uses the same argument in the assumptions, otherwise it would be impossible us the assumptions in the proof of the goal.

Algebraic Manipulations. The most challenging part in this proof is the automatic generation of the instantiation term $\mathrm{Min}\left[1, \frac{e_0}{|a_2|+|a_1|+1}\right]$, which is performed by a heuristic combination of solving, substitution, and simplifying, as well as rewriting of expressions under the absolute value function, and it is realized at the end of the proof. The goal in this moment is[1]:

(16) $|\,(f_1\,[n_0] * f_2\,[n_0]) - (a_1 * a_2)\,| < e_0$

and the main assumptions are:

(18) $|f_1\,[n_0] - a_1| < e$

(19) $|f_2\,[n_0] - a_2| < e$

Internally the prover replaces $f_1[n_0]$ and $f_2[n_0]$ by x_1 and x_2, respectively, both in the goal and in the assumptions. The argument of the absolute value in the transformed goal is $E_0 = x_1 * x_2 - a_1 * a_2$ and in the transformed assumptions $E_1 = x_1 - a_1$ and $E_2 = x_2 - a_2$.

First we use the following heuristic principle: transform the goal expression E_0 such that it uses as much as possible E_1 and E_2, because about those we know that they are small. In order to do this we take new variables y_1, y_2, we solve the equations $y_1 = E_1$ and $y_2 = E_2$ for x_1, x_2, we substitute the solutions in E_0 and the result simplifies to: $a_1 * y_2 + a_2 * y_1 + y_1 * y_2$. This is the internal representation of the absolute value argument in the goal (20).

(20) $|a_1 * (f_2[n_0] - a_2) + a_2 * (f_1[n_0] - a_1) + (f_1[n_0] - a_1) * (f_2[n_0] - a_2)| < e_0$

[1] Formula labels are as in [7].

Note that the transformation from (16) to (20) is relatively challenging even for a human prover.

$$(21)\ |a_1 * (f_2[n_0] - a_2) + a_2 * (f_1[n_0] - a_1) + (f_1[n_0] - a_1) * (f_2[n_0] - a_2)| \leq$$

$$\leq |a_1 * (f_2[n_0] - a_2)| + |a_2 * (f_1[n_0] - a_1)| + |(f_1[n_0] - a_1) * (f_2[n_0] - a_2)| =$$

$$= |a_1| * |f_2[n_0] - a_2| + |a_2| * |f_1[n_0] - a_1| + |f_1[n_0] - a_1| * |f_2[n_0] - a_2| <$$

$$< |a_1| * e + |a_2| * e + e * e \leq |a_1| * e + |a_2| * e + e = e * (|a_1| + |a_2| + 1) =$$

$$= \frac{e_0}{|a_2|+|a_1|+1} * (|a_1| + |a_2| + 1) = e_0$$

The formula (21) is realized by *rewriting of the absolute value expressions*. Namely, we apply certain rewrite rules to expressions of the form $|E|$ and their combination, as well as to the metavariable e. Every rewrite rule transforms a (sub)term into one which is not smaller, so we are sure to obtain a greater or equal term. The final purpose of these transformations is to obtain a strictly positive ground term t multiplied by the target metavariable (here e). Since we need a value for e which fullfils $t * e \leq e_0$, we can set e to e_0/t. The rewrite rules come from the elementary properties of the absolute value function: (e. g. $|u + v| \leq |u| + |v|$)) and from the principle of *bounding the ϵ-bounds*: Since we are interested in the behaviour of the expressions in the immediate vicinity of zero, the bound e can be bound from above by any positive value. In the case of product (presented here), we also use the rule: $e * e \leq e$, that is we bound e to 1. This is why the final expression of e is the minimum between 1 and the term t found as above.

Proving Simple Conditions. At certain places in the proof, the conditions upon certain quantified variables have to be proven. The prover does not display a proof of these simple statements, but just declares them to be consequences of "elementary properties of $\mathbb{R}$". (Such elementary properties are also invoked when developing formulae (20) and (21).) In the background, however, the prover uses *Mathematica* functions in order to check that these statements are correct. This happens for instance for the subgoal $\underset{e_0}{\forall}(e_0 > 0 \Rightarrow e > 0)$ and will be treated after the instatiation term for e is found, by using QE on this formula (where e has the found value Min[...]), which returns *True* in *Mathematica*.

3 Conclusion and Further Work

When applied to problems over reals, *Satisfiability Modulo Theories (SMT)* solving combines techniques from automated reasoning and from computer algebra. From the point of view of automated reasoning, proving unsatisfiability of a set of clauses appears to be quite different from producing natural-style proofs. Indeed the proof systems are different (resolution on clauses vs. some version of sequent calculus), but they are essentially equivalent, relaying on equivalent transformations of formulae. Moreover, the most important steps in first order proving, namely the instantiations of universally quantified formulae (which in natural-style proofs is also present as the equivalent operation of finding witnesses for existentially quantified goals), are actually the same or very similar.

The full automation of proofs in elementary analysis constitutes a very interesting application for the combination of logic and algebraic techniques, which is essentially equivalent to SMT solving (combining satisfiability checking and symbolic computation). Our experiments show that complete and efficient automation is possible by using certain heuristics in combination with complex algebraic algorithms.

Further work includes a systematic treatment of various formulae which appear in textbooks, and extension of the heuristics to more general types of formulae. In this way we hope to address the class of problems which are usually subject to SMT solving.

References

1. Abraham, E., Jebelean, T.: Adapting Cylindrical Algebraic Decomposition for proof specific tasks. In: Kusper, G. (ed.) ICAI 2017: 10th International Conference on Applied Informatics (2017), in print
2. Bauer, A., Clarke, E., Zhao, X.: Analytica – an experiment in combining theorem proving and symbolic computation. Journal of Automated Reasoning **21**(3), 295–325
3. Buchberger, B., Jebelean, T., Kutsia, T., Maletzky, A., Windsteiger, W.: Theorema 2.0: Computer-assisted natural-style mathematics. JFR **9**(1), 149–185 (2016)
4. Collins, G.E.: Quantier elimination for real closed fields by cylindrical algebraic decomposition. In: Automata Theory and Formal Languages. LNCS, vol. 33, pp. 134–183. Springer (1975)
5. Jebelean, T.: Techniques for natural-style proofs in elementary analysis. ACM Communications in Computer Algebra **52**(3), 92–95
6. Jebelean, T.: Techniques for natural-style proofs in elementary analysis (extended abstract). In: Bigatti, A.M., Brain, M. (eds.) Third International Workshop on Satisfiability Checking and Symbolic Computation, 2018
7. Jebelean, T.: A heuristic prover for elementary analysis in Theorema. Tech. Rep. 21-07, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz (April 2021)
8. Jebelean, T., Buchberger, B., Kutsia, T., Popov, N., Schreiner, W., Windsteiger, W.: Automated Reasoning. In: Buchberger, B., et al. (eds.) Hagenberg Research, pp. 63–101. Springer (2009)
9. Vajda, R., Jebelean, T., Buchberger, B.: Combining logical and algebraic techniques for natural style proving in elementary analysis. Mathematics and Computers in Simulation **79**(8), 2310–2316 (April 2009)