To appear in ACM Communications in Computer Algebra

Techniques for Natural-style Proofs in Elementary Analysis
POSTER ABSTRACT*

Tudor Jebelean
RISC–Linz, Johannes Kepler University, Austria
Tudor.Jebelean@JKU.AT

Abstract

Combining methods from satisfiability checking with methods from symbolic computation promises to solve challenging problems in various areas of theory and application. We look at the basically equivalent problem of proving statements directly in a non–clausal setting, when additional information on the underlying domain is available in form of specific properties and algorithms. We demonstrate on a concrete example several heuristic techniques for the automation of natural–style proving of statements from elementary analysis. The purpose of this work in progress is to generate proofs similar to those produced by humans, by combining automated reasoning methods with techniques from computer algebra. Our techniques include: the S-decomposition method for formulae with alternating quantifiers, quantifier elimination by cylindrical algebraic decomposition, analysis of terms behaviour in zero, bounding the ϵ-bounds, rewriting of expressions involving absolute value, algebraic manipulations, and identification of equal terms under unknown functions. These techniques are being implemented in the *Theorema* system and are able to construct automatically natural–style proofs for numerous examples including: convergence of sequences, limits and continuity of functions, uniform continuity, and other.
Keywords: Satisfiability Checking, Natural-style Proving, Symbolic Computation.

1 Introduction

The need for natural–style proofs (that is: similar to proofs produced by humans – see [2])[1] arises in various applications, as for instance in tutorials, demonstrations, and interactive teaching systems. Some authors argue for the use of natural style when the proof system is not completely automatic (e. g. interactive provers) because this facilitates the interaction with the human user.

When applied to problems over reals, *Satisfiability Modulo Theories (SMT)* 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. a version of sequent calculus), but they are essentially equivalent, relaying on equivalent transformation 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. From the point of view of Computer Algebra, finding these instantiations is the most important operation, thus here again one can use the same techniques in SMT and natural–style proving. Therefore the techniques can be easily moved from one area to the other, because they are essentially equivalent.

*Partially supported by project "Satisfiability Checking and Symbolic Computation" (H2020-FETOPN-2015-CSA 712689).
[1] Here we do not mean *natural deduction.*

In this poster we present our results on a class of proof problems which arise in elementary analysis. These are problems involving formulae with alternating quantifiers, which are difficult to solve by the purely logic approach, because this requires the use of a large number of formulae which express the necessary properties of numbers (naturals, integers, rationals, reals). We use the following techniques, which extend our previous work [6]: the S-Decomposition method for formulae with alternating quantifiers [5], Quantifier Elimination by Cylindrical Algebraic Decomposition [3], analysis of terms behaviour in zero, bounding the ϵ-bounds, rewriting of expressions involving absolute value, algebraic manipulations (solving, substitution, and simplification), and identification of equal terms under unknown functions.

Our prover, implemented in the frame of the *Theorema* system [2], aims at producing natural–style proofs for simple theorems involving limits of sequences and of functions, continuity, uniform continuity, etc. An important aspect of the naturalness of the proof is the fact that the prover does not need to access a large collection of formulae which express the properties of the domains involved. Rather, the prover uses symbolic computation techniques from algebra in order to discover relevant terms and to check necessary conditions, and only needs as starting formulae the definitions of the main notions involved.

An extended version of this poster abstract was submitted with the same title to *3-rd Workshop on Satisfiability Checking and Symbolic Computation* (Oxford, July 2018). Most of the text presented here, as well as full details on the examples and on the techniques used for solving them are presented in the technical report [4].

2 Example: Product of Convergent Sequences

We illustrate our method by the proof of the theorem *"the product of two convergent sequences is convergent"*, which is presented in detail in [4] and partially on the next pages.

Note the specific structure of the quantified formulae: the quantifier has a first underline which declares the *type* of the variable, and possibly a second underline which declares a *condition* upon the quantified variable. These two conditions have a specific role during the decomposition of the proof using our method – see [1]. For space reasons, in this presentation we do not address the treatment of types. Note also that we follow the convention of *Mathematica* and *Theorema*, by denoting function and predicate application by square brackets instead of the traditional round parantheses.

The proof starts from the definition of the notion of convergent sequence: (f is a real function of natural argument, e is real, m, n are naturals):

$$\text{IsConvergent}(f) \;\Leftrightarrow\; \underset{a}{\exists}\; \underset{e>0}{\forall}\; \underset{M}{\exists}\; \underset{n\geq M}{\forall}\; \text{Abs}[f(n)-a] < e$$

After expanding this definition and the definition of product, the prover decomposes the theorem into 3 statements: two assumptions and one goal.

We assume :

(1) $\exists_a \forall_e (e > 0 \Rightarrow \exists_M \forall_n (n \geq M \Rightarrow \mathsf{Abs}[f_1[n] - a] < e))$

(2) $\exists_a \forall_e (e > 0 \Rightarrow \exists_M \forall_n (n \geq M \Rightarrow \mathsf{Abs}[f_2[n] - a] < e))$

and we prove :

(3) $\exists_a \forall_e (e > 0 \Rightarrow \exists_M \forall_n (n \geq M \Rightarrow \mathsf{Abs}[(f_1[n] * f_2[n]) - a] < e))$

The main structure of the proof follows from the S-Decomposition method [5]: the quantifiers are removed from the 3 statements in parallel, using a combination of inference steps which decompose the proof into several branches, introduce Skolem constants, and require special terms (for instantiations or as witnesses). In the background the prover keeps certain quantified formulae which express the general

structure of the proof and which are used at certain moments for finding the witnesses and the instantiation terms (as described in [1]).

3 Application of Special Techniques

We describe here how some of the special techniques mentioned in the introduction are used in the course of the proof. (Details about the techniques which are not presented here are presented in [4].)

Witness for Existential Goal. The assumptions (1) and (2) are transformed by introducing the Skolem constants a_1, a_2 instead of the existential variables. Now the prover must produce the witness $a_1 * a_2$ needed for the existential variable a in the current goal (3). We use the well known technique of *metavariables*, that is we replace the existential variable by a new symbol, which is a name for the term which we need to find. This will be found later, when the prover generates a certain simplified formula [6, 1] which we will call *formula (A)*:

$$\forall_{a_1,a_2}\exists_{a_0}\forall_{e_0}(e_0 > 0 \Rightarrow \exists_{e_1,e_2}(e_1 > 0 \wedge e_2 > 0 \wedge \forall_{x_1,x_2}(\mathrm{Abs}[x_1 - a_1] < e_1 \wedge \mathrm{Abs}[x_2 - a_2] < e_2 \Rightarrow \mathrm{Abs}[x_1 * x_2 - a_0] < e_0)))$$

The value of the metavariable (standing for a_0) can be found using Quantifier Elimination (QE) by Cylindrical Algebraic Decomposition (CAD), as described in [1] – which works for the case of the *sum* of convergent sequences. However in the case of *product* the the corresponding QE problem cannot be solved in short time by CAD (e. g. in *Mathematica*), and even if solved, it generates a very complicated expression which cannot be used for finding a_0.

In the proof above we used another, much simpler technique: *reasoning about terms behaviour in zero*. It is clear that the formula (A) expresses the behaviour of the polynomials under Abs in any (small) vicinity of zero. Since polynomials are continuous, this will also be their behaviour *in zero*. One can in fact prove that formula (A) is equivalent to the formula:

$$\forall_{a_1,a_2}\exists_{a_0}\forall_{x_1,x_2}(\mathrm{Abs}[x_1 - a_1] = 0 \wedge \mathrm{Abs}[x_2 - a_2] = 0 \Rightarrow \mathrm{Abs}[x_1 * x_2 - a_0] = 0)$$

In our special case it is immediately clear that a_0 equals $a_1 * a_2$, but we implemented a more general method: the two LHS equations are solved for x_1, x_2, then the values are replaced in the RHS equation, which is then solved for a_0.

Later in the proof one needs to find a witness for M, this can be handled by quantifier elimination [1].

Instantiation Terms for Universal Assumptions. The most challenging part is the automatic generation of the instantiation term needed for e_1, e_2, which is performed by a heuristic combination of solving, substitution, and simplifying, as well as rewriting of Abs expressions.

Note the goal has under the Abs the expression (call it E_0) corresponding to $x_1 * x_2 - a_1 * a_2$. Let us also name the Abs arguments of the assumptions as E_1 and E_2, respectively.

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: $a_1 * y_2 + a_2 * y_1 + y_1 * y_2$.

Now the prover uses *rewriting of the Abs expressions*. Namely, we apply certain rewrite rules to expressions of the form $\mathrm{Abs}[E]$ and their combination, as well as to the metavariable e, and $\mathrm{Abs}[y_1]$, $\mathrm{Abs}[y_2]$ are transformed into 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), in our case t is: $\mathrm{Abs}[a_1] + \mathrm{Abs}[a_2] + 1$.

Then we know that we need a value for e which fullfils $t*e \leq e_0$, thus we can set e to e_0/t. The rewrite rules come from the elementary properties of Abs (e. g. $\text{Abs}[u+v] \to (\text{Abs}[u] + \text{Abs}[v])$) and from the principle of *bounding the ϵ-bounds*: Since we are interested in the behaviour of the expressions in the immediated vicinity of zero, the bounds (e, e_0, e_1, e_2) can be bound from above by any positive value. In the case of product (presented here), we also use the heuristic rule: $e * e \to e$, that is we bound e to 1. This is why the final expression of e is a minimum, namely: $\text{Min}[1, e_0/(Abs[a_1] + \text{Abs}[a_2] + 1)]$.

In order to make it work for more complex expressions, namely rational functions, we use a second set of rules which decrease the term – in order to obtain a bound for U/V, increase U and decrease V. Using this we obtain automatically appropriate bounds for the case of inverse of a sequence and for the case of fraction of two sequences.

Full detail of the techniques and of the examples are presented in [4].

4 Conclusion and Further Work

The full automation of proofs in elementary analysis constitutes a very intersting application for the combination of logic and algebraic techniques, which is essentially equivalent to SMT (combining satisfiability checking and symbolic computation). Our experiment 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.

References

[1] Erika Abraham and Tudor Jebelean. Adapting Cylindrical Algebraic Decomposition for Proof Specific Tasks. In G. Kusper, editor, *ICAI 2017: 10th International Conference on Applied Informatics*, 2017. In print.

[2] Bruno Buchberger, Tudor Jebelean, Temur Kutsia, Alexander Maletzky, and Wolfgang Windsteiger. Theorema 2.0: Computer-Assisted Natural-Style Mathematics. *JFR*, 9(1):149–185, 2016.

[3] G. E. Collins. Quantier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183. Springer, 1975.

[4] Tudor Jebelean. Experiments with Automatic Proofs in Elementary Analysis. Technical Report 18-06, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, April 2018.

[5] Tudor Jebelean, Bruno Buchberger, Temur Kutsia, Nikolaj Popov, Wolfgang Schreiner, and Wolfgang Windsteiger. Automated Reasoning. In B. Buchberger et al., editor, *Hagenberg Research*, pages 63–101. Springer, 2009.

[6] R. Vajda, T. Jebelean, and B. Buchberger. Combining Logical and Algebraic Techniques for Natural Style Proving in Elementary Analysis. *Mathematics and Computers in Simulation*, 79(8):2310–2316, April 2009.