Mechanical Explanation in "Systems that Explain Themselves"

Walther Neuper
wneuper@ist.tugraz.at

University of Technology, Graz, Austria

1 Introduction

Any kind of "systems that explain themselves" allow to dissolve magic by enabling students to investigate systems' internals; given sufficient endurance such investigation would enable students to know what they do and would avoid using mathematical software for any kind of incantation by getting some solution from some magic source.

Further questions concerning education and learning of mathematics are bypassed for the sake of brevity; here only features of technology are described, which emerge from a new software generation [1] built upon technology from (Computer) Theorem Proving (abbreviated "TP"). The extended abstract uses a prototype [KN18] for discussing general features of the new generation. The prototype is built upon the TP Isabelle [NPW08].

The extended abstract gives lists of features, which are partly realised in the prototype and can be implemented by more or less direct re-use of Isabelle's TP technology. The list items are short; a planned research paper will describe them in more detail.

2 Mechanical Explanation and Language Layers

The word "explain" indicates involvement of language in human-computer interaction. The prototype under consideration employs four different language layers in order to cover all of engineering mathematics. Each of the layers provides principal benefits for users as well as added-value by implementation in interactive software. Two of the layers can provide "next step guidance", i.e. the system is able to suggest a next step in case the student gets stuck in problem solving.

2.1 Term Language

The language of mathematical formulas (propositions, predicates, algebraic terms) is the basis of the other language layers. The prototype adopts Isabelle's simple typed lambda calculus [Bar85]. Benefits and added-value come into bearing in the other languages, in particular by "next step guidance".

2.1.1 Principal Benefits

Principal benefits of simple typed lambda calculus are:

1. uniformity over all domains of mathematics
2. type system which efficiently excludes ambiguities
3. clear description of functions and respective rules (for instance, the chain rule for differentiation, which states a specific property of *functions*)

In: O. Hasan, W. Neuper, Z. Kovcs, W. Schreiner (eds.): Proceedings of the Workshop *CME-EI: Computer Mathematics in Education - Enlightenment or Incantation*, Hagenberg, Austria, 17-Aug-2018, published at http://ceur-ws.org

[1] See for instance `https://www.uc.pt/en/congressos/thedu`.

2.1.2 Added Value of Implementation

Simple typed lambda calculus gains added-value by implementation in Isabelle's front-end [Wen14] as follows:

1. each element of a formula (operator or variable or constant) reveals it's type by mouse pointer
2. each element of a formula is connected with it's definition by mouse click; close to definitions there usually are also relevant theorems
3. feedback to input of formulas is given: imprecise input is commented (for instance, ambiguous parse trees), incorrect input is indicated by spotting the error
4. the structure of a formula, i.e. respective sub-terms can be made visible (for instance by coloured boxes [Mah18, p. 33])
5. the internal representation of lambda calculus can be presented to students according to their detailed requests.

Fig.1 gives an idea about features already present in Isabelle2017.

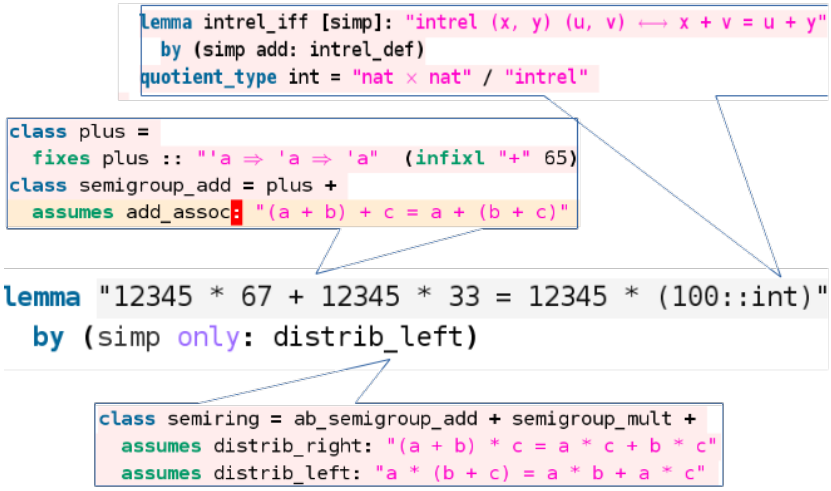

Figure 1: Isabelle shows underlying knowledge transparently.

2.2 Specification Language

The prototype is designed for interactively solving engineering problems, see in [KN18, Fig.3] the left window for an example problem in structural engineering. Such a problem is formalised by a specification using the term language from §2.1, which makes input items and output items explicit and where input is constrained by so-called "pre-conditions" and where the characteristics of the problem is given by a "post-condition" relating input and output. For instance, [KN18, p. 100], gives an example.

2.2.1 Principal Benefits

Engineers are educated in specifying their problems more or less formally, at high-school specification occurs during translation of word problems or figures into calculations, usually not in a systematic way. So making specification explicit might be beneficial even for younger students.

1. specification is a prerequisite for mechanical problem solving — it is the cost for obtaining mechanised explanations (and solutions)
2. the pre-condition explicitly determines which input values make the problem solvable
3. the post-condition makes explicit, how a result solves a problem: in case the problem is solved, the post-condition evaluates to true (or not) if instantiated with the result
4. specifications are collected in trees, where certain branches contain certain types of problems
5. problems can be decomposed into sub-problems with a specification of their own

2.2.2 Added Value of Implementation

In case of high-school students the most important added value might be, that the dialog module [KN18, p. 97ff] might skip specification at all and immediately start stepwise calculating towards a solution. However, in

engineering studies it is crucial to learn how to specify problems.

1. specifications as black boxes simplify problem solving by breaking down complex problems to larger sub-problems (represented by specifications)
2. specifications can be interactively arranged until all their inputs are covered by outputs of other specifications, see a respective slide movie [2].
3. specifications can be easily searched in the respective tree, inserted and replaced in solutions
4. successfully specified problems can be solved by a key stroke automatically constructing the solution
5. trees of specifications allow automated refinement (for instance, determining an appropriate specification for solving a certain equation)

Fig.2 shows a (forward) proof by construction in a format close to calculations as done in engineering mathematics; this is implemented in the ISAC prototype.

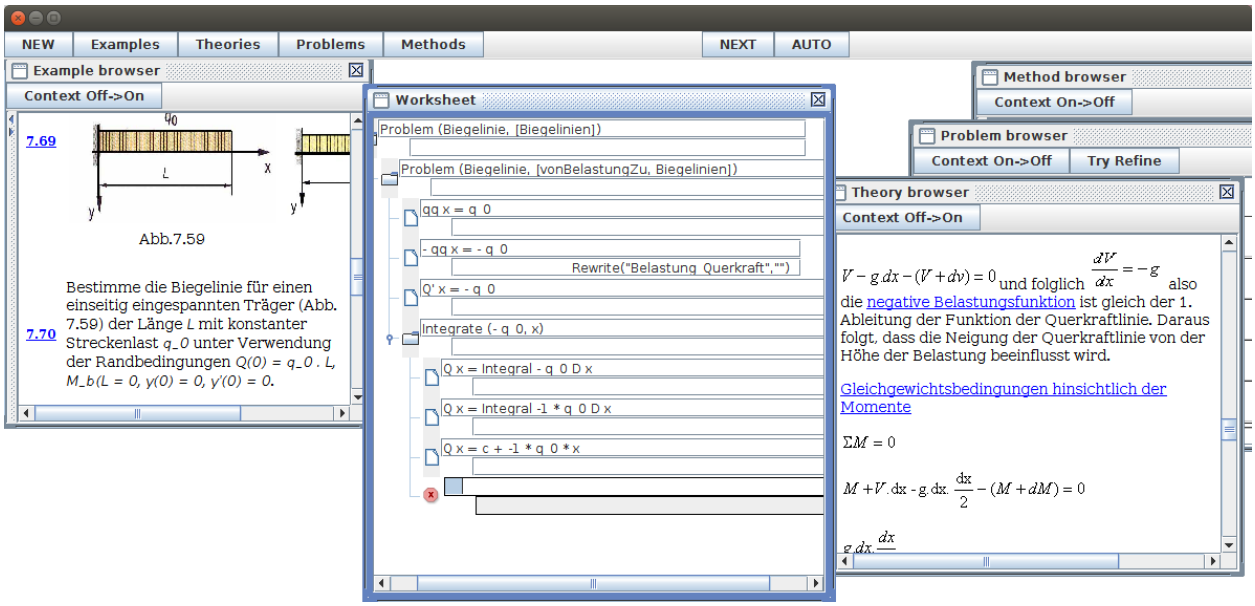

Figure 2: ISAC's Worksheet close to hand-written calculations.

2.2.3 "Next Step Guidance"

A next step is determined by the prototype using a so-called formalisation hidden behind the textual problem description (see [KN18, p. 100]). The post-condition can be simplified by stripping off quantifiers, such that engineers are only concerned with relevant parts (see the field `07 Relate` in [KN18, p. 100])

1. input items, output items and parts of the post-condition are marked (by colours) as "correct", "incorrect", "missing" or "superfluous", which suggests respective action to the student
2. arrangement of sub-problems while constructing a solution can be read from the respective program (see §2.4 below) and suggested as well.

2.3 Proof Language

Isabelle's proof language Isar [Wen07] is more powerful than required by what we call engineering mathematics; the prototype shows that forward proof and equational reasoning is sufficient. The proof language builds upon the term language §2.1.

2.3.1 Principal Benefits

The benefits of equational forward reasoning is that it coincides with traditional formats for (symbolic) calculations, see example in [KN18, Fig.1]; so explanations benefit from the following features:

1. students get calculations in a format they are familiar with

[2] `http://www.ist.tugraz.at/projects/isac/publ/movie-sub-problems.pdf`

2. the format collects all steps of calculation towards a solution in a consistent framework (a tree + formal metalogic [Bac10])
3. each step is justified by application of appropriate theorems
4. specific steps are equivalent to functions in Computer Algebra: simplification, differentiation, equation solving, etc
5. these steps, like all others, can be decomposed into elementary steps of application of a single theorem (so Computer Algebra becomes "transparent")

2.3.2 Added Value of Implementation

Explanations arise from flexible interaction as partly shown already in the prototype:

1. quick change from survey to detail by collapsing and expanding the branches of the calculation tree
2. justification for any step can be inspected on demand
3. inspection can go down to application of single theorems
4. application of a theorem can be investigated according to §2.1.2 Pt.4
5. certain steps found inappropriate some steps later can be redone while trying alternative ways to a solution
6. alternatives can be tried in parallel windows on the screen
7. how a stepwise constructed result solves a problem can be displayed according to §2.2.1 Pt.3

2.3.3 "Next Step Guidance"

To know a next step in a proof is out of scope and out of interest for interactive theorem proving. However, engineering mathematics is usually taught together with algorithms solving certain engineering problems. A program language for describing such algorithms is introduced below in §2.4 together with Lucas-Interpretation. Lucas-Interpretation allows the system to suggest a next step while retaining users' freedom to decide on their own next steps is described on [Neu12, Neu16].

1. in case the student gets stuck, the system can propose a next step (at least after backtracing to an appropriate step)
2. a next step can be presented by (partially) giving the resulting formula according to §2.1.2 Pt.4
3. a next step can be presented by giving an applicable theorem (partially), or a list of theorems for selection, etc

2.4 Programming Language

The program language extends the term language §2.1 (programs are Isabelle terms) by so-called tactics, which are handled as break-points by Lucas-Interpretation and pass control to users (or more precisely, to the dialogue module).

2.4.1 Principal Benefits

The programs describe algorithms required to solve the engineering problems, this is how the prototype provides "next step guidance" during stepwise construction of solutions according to §2.3.3. However, this language is primarily relevant for authors extending mathematics knowledge for use in the prototype. Students need not see this language layer, but it could explain the state of a solution by presenting the program code together with the current break point, or it could explain the algorithm from an abstract point of view.

2.4.2 Added Value of Implementation

Since only recently Isabelle's function package has been adopted instead of parsing from strings, there is a recent account on added value [Neu18] repeated here in short:

1. syntax errors are indicated accurately at the right location
2. type annotations for the functions arguments shift into the initial signature
3. less type annotations are required within the code
4. syntax highlighting indicates how identifiers are interpreted (as constants, as free variable, etc)
5. free variables on the right-hand-side of equalities are rejected.

3 Conclusions

The above lists are not simple enumerations for kinds of explanations, which justify the claim, that "systems that explain themselves" can be built by exploiting the power of TP technology. Rather, creative combination of the various features listed above might justify this claim — together with "next step guidance".

For example, if a student tries to solve an equation by use of the theorem $a \cdot c = b \equiv a = \frac{b}{c}$, this theorem might be not applicable — for instance, in case the "$\cdot$" is an operation in a ring. This might occur during stepwise equation solving according to §2.3 while the relevant explanation comes from inspecting the types according to §2.1.2 Pt.1. And then the student might review the specification according to §2.2.1 Pt.2 and find out, that he or she chose an inappropriate problem, which motivates to lookup the collection of equation solvers according to Pt.4 in §2.2.1 and to select a more appropriate algorithm.

How students are able for such creative combinations, this will remain an open question until TP-based math software will be available for field tests. A major challenge will be to design the dialogue module capable to restrict "next step guidance" such, that students are not deduced to just hit buttons, but are invited to accomplish as most as possible on their own while asking the system for explanations in all the varieties listed above.

References

[Bac10] R.-J. Back. Structured derivations: a unified proof style for teaching mathematics. *Formal Aspects of Computing*, 22(5):629–661, 2010.

[Bar85] H. P. Barendregt. *The Lambda Calculus*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, New York, Oxford, second printing edition, 1985.

[KN18] Alan Krempler and Walther Neuper. Prototyping "systems that explain themselves" for education. In Pedro Quaresma and Walther Neuper, editors, Proceedings 6th International Workshop on *Theorem proving components for Educational software,* Gothenburg, Sweden, 6 Aug 2017, volume 267 of *Electronic Proceedings in Theoretical Computer Science*, pages 89–107. Open Publishing Association, 2018. `https://arxiv.org/pdf/1803.01470v1.pdf`.

[Mah18] Marco Mahringer. Formula editors for tp-based systems. state of the art and prototype implementation in ISAC. Master's thesis, University of Applied Sciences, Hagenberg, Austria, 2018. `http://www.ist.tugraz.at/projects/isac/publ/mmahringer-master.pdf`.

[Neu12] Walther Neuper. Automated generation of user guidance by combining computation and deduction. In Pedro Quaresma and Ralph-Johan Back, editors, *Electronic Proceedings in Theoretical Computer Science*, volume 79, pages 82–101. Open Publishing Association, 2012. `http://eptcs.web.cse.unsw.edu.au/paper.cgi?THedu11.5`.

[Neu16] Walther Neuper. Lucas-interpretation from users' perspective. In *Joint Proceedings of the FM4M, MathUI, and ThEdu Workshops, Doctoral Program, and Work in Progress at the Conference on Intelligent Computer Mathematics*, pages 83–89, Bialystok, Poland, July 25-29 2016. `http://cicm-conference.org/2016/ceur-ws/CICM2016-WIP.pdf`.

[Neu18] Walther Neuper. Lucas interpretation from programmers' perspective. 2018. submitted to ThEdu'18 `http://www.ist.tugraz.at/projects/isac/publ/lucin-prog-view.pdf`.

[NPW08] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL, a proof assistant for higher-order logic*. Springer Verlag, 2008.

[Wen07] Makarius Wenzel. Isabelle/Isar — a generic framework for human-readable proof documents. In R. Matuszewski and A. Zalewska, editors, *From Insight to Proof — Festschrift in Honour of Andrzej Trybulec*, volume 10 of *Studies in Logic, Grammar, and Rhetoric*, pages 277–298. University of Bialystok, 2007. `http://mizar.org/trybulec65/19.pdf`.

[Wen14] Makarius Wenzel. System description: Isabelle/jEdit in 2014. In *Proceedings Eleventh Workshop on User Interfaces for Theorem Provers, UITP 2014, Vienna, Austria, 17th July 2014.*, pages 84–94, 2014. `http://dx.doi.org/10.4204/EPTCS.167.10`.