DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
73

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
73

DMITRI LEPP

Solving simplification problems in the domain of exponents, monomials and polynomials in interactive learning environment T-algebra

TARTU UNIVERSITY PRESS

Faculty of Mathematics and Computer Science, University of Tartu, Estonia

Dissertation is accepted for the commencement of the degree of Doctor of Philosophy (PhD) on June 29, 2011, by the Council of the Institute of Computer Science, University of Tartu.

Supervisor:

Associate Professor Rein Prank (Cand. Sc.)
University of Tartu
Tartu, Estonia

Opponents:

Professor Nicolas Balacheff (PhD)
Research Director, Laboratoire d'Informatique de Grenoble
MeTAH research team
Grenoble, France

Associate Professor Jaak Henno (Cand. Sc.)
Tallinn University of Technology
Tallinn, Estonia

Commencement will take place on August 25, 2011 at 16:00 in Liivi 2–403.

ISSN 1024–4212
ISBN 978–9949–19–787–3 (trükis)
ISBN 978–9949–19–788–0 (PDF)

Autoriõigus: Dmitri Lepp, 2011

Tartu Ülikooli Kirjastus
www.tyk.ee
Tellimus nr. 476

CONTENTS

LIST OF ORIGINAL PUBLICATIONS

1. Lepp, D. (2003). Program for exercises on operations with polynomials. In *Proceedings of 6th International Conference on Technology in Mathematics Teaching*, pp. 365–369, Volos, Greece
2. Issakova, M. and Lepp, D. (2004). Rule dialogue in problem solving environment T-algebra. In *Proceedings TIME–2004: Montreal International Symposium on Technology and its Integration into Mathematics Education*, 16 p., Montreal, Canada.
3. Lepp, D. (2005). Extended Solution Step Dialogue in Problem Solving Environment T-algebra. In *Proceedings of the 7th International Conference on Technology in Mathematics Teaching* (ICTMT7), volume 1, pp. 267–274, Bristol, UK.
4. Lepp, D., Issakova, M. and Vaiksaar, V. (2005). Expression Editor Features That Simplify Student Work On Manipulating Expressions. In *Proceedings of the 7th International Conference on Technology in Mathematics Teaching* (ICTMT7), volume 1, pp. 259–266, Bristol, UK.
5. Issakova, M., Lepp, D. and Prank, R. (2005). Input Design in Interactive Learning Environment T-algebra. In *Proceedings ICALT–2005: The 5th IEEE International Conference on Advanced Learning Technologies*, pp. 489–491, Kaohsiung, Taiwan.
6. Lepp, D. (2006). Error Diagnosis in Problem Solving Environment Using Action-Object-Input Scheme. In *ITS 2006 Proceedings*, LNCS 4053, pp. 769–771, Springer-Verlag.
7. Lepp, D. (2006). Using Action-Object-Input Scheme for Error Diagnosis in Problem Solving Environment. In *Proceedings of the Student Track ITS 2006*, pp. 18–27, Jhongli, Taiwan.
8. Lepp, D. (2006). Design of polynomial transformation rules in problem solving environment T-algebra. In *Proceedings DES–TIME–2006: Dresden International Symposium on Technology and its Integration into Mathematics Education 2006*, 15p., Dresden, Germany.
9. Lepp, D. (2006). Error Diagnosis and Categorization in Problem Solving Environment Using Action-Object-Input Scheme. In *Proceedings of The 11th Asian Technology Conference in Mathematics* (ATCM 2006), pp. 215–224, Hong Kong, China.
10. Prank, R., Issakova, M., Lepp, D., Vaiksaar, V. and Tõnisson, E. (2006). Problem solving environment T-algebra. In *Proceedings of 7th International Conference Teaching Mathematics: Retrospective and Perspectives*, pp. 190–197, Tartu, Estonia.
11. Prank, R., Issakova, M., Lepp, D. and Vaiksaar, V. (2006). Designing Next-Generation Training and Testing Environment for Expression Manipulation. In *International Conference on Computational Science* (ICCS 2006), Part I, LNCS 3991, pp. 928–931, Springer-Verlag.
12. Issakova, M., Lepp, D. and Prank, R. (2006). T-algebra: Adding Input Stage To Rule-Based Interface For Expression Manipulation. *Inter-*

national Journal for Technology in Mathematics Education, 13(2): 89–96.

13. Lepp, D. (2007). Study of Student Mistakes in Solving Simplification Problems on Paper and Possibility of these Mistakes in the T-algebra Environment. In *Proceedings of the 8th International Conference on Technology in Mathematics Teaching* (ICTMT8), 6p., Hradec Králové, Czech Republic.
14. Prank, R., Issakova, M., Lepp, D., Tõnisson, E. and Vaiksaar, V. (2007). Integrating rule-based and input-based approaches for better error diagnosis in expression manipulation tasks. In *Symbolic Computation and Education*, pp. 174–191, World Scientific Publishing Co.
15. Prank, R., Issakova, M., Lepp, D., Tõnisson, E. and Vaiksaar, V. (2008). T-algebra – Intelligent Environment for Expression Manipulation Exercises. In *Topic Study Group 22: 11th International Congress on Mathematical Education*, pp. 1–7, Monterrey, Mexico.
16. Prank, R., Lepp, D. (2010). Tools for acquiring data about student work in interactive learning environment T-algebra. In *ITS 2010 Proceedings*, LNCS 6095, pp. 396–398.

1 INTRODUCTION

This thesis is based on the work that has been done for the T-algebra project in 2004–2008. The outcome of the project is a new interactive learning environment for step-by-step solving of expression manipulation problems in four different fields, particularly for solving simplification problems in the field of exponents, monomials and polynomials. Different general decisions and common parts of the system were designed during project seminars by the whole project team with help from math teachers involved in the project. Some other, field specific decisions were suggested by the responsible author. The thesis contains a general description of different aspects of the T-algebra environment based on articles published by the author and other team members.

The main contribution of the thesis is design, implementation, testing and evaluating of the environment with a novel step dialogue for proper learning and diagnosis of knowledge gaps in solving simplification problems in the field of exponents, monomials and polynomials.

1.1 Motivation

Expression manipulation is one of the central skills needed for solving tasks in practically all fields of mathematics. However, learning outcomes in this area are often not satisfying. One of the reasons for poor performance is repetition of incorrect solution attempts without getting feedback. In the paper-and-pencil training process, students make many mistakes but teachers are not able to discover and correct them in time. Thus, mistakes are repeated many times and can become habitual. The need to analyse information quickly implies that the training could be improved by using computerised training environments.

At school, the basic types of expression manipulation tasks are usually taught together with some solution algorithms. When a student solves such tasks, he should at each solution step:

1. choose a transformation rule corresponding to a certain operation in the algorithm (or some simplification or calculation rule known earlier),
2. select the operands (certain parts of expressions or equations) for this rule,
3. replace them with the result of the operation.

For proper learning of expression manipulation as well as for assessment and diagnosis of knowledge gaps, an environment should be available where all the necessary decisions and calculations at each solution step would be made by the student and the program would be able to understand the mistakes.

Existing software does not address the whole complex spectrum of potential problems. For example, some systems do not allow students to make all decisions during the steps, and also do not allow making certain typical errors. Some other environments do not provide error diagnosis to support the student.

In the next section we will see some examples of existing software that we have studied prior to starting our own project. We have formulated the features (see section 1.3) that the environment should have to be useful for students.

In 2004 we started a project for creating a new learning environment called T-algebra for four areas of school mathematics: calculation of the values of numerical expressions; operations with fractions; solving of linear equations, inequalities and systems of linear equation; operations with polynomials. Prior to that, in my Master's thesis (Lepp, 2003a, Lepp, 2003b), I tried to create a prototype of the system with a similar solution step dialogue, which was reused and improved for the T-algebra project. Our main goal is to design a solution dialogue that allows the program to understand the decisions made by students at all three stages of the step.

1.2 Related works

Prior to making any decisions we studied similar environments and other computerized ways to teach mathematics. Before writing this thesis we checked again if some new systems were created, but we have not found any new interesting system worth mentioning. We also studied results of other researches who have tried to evaluate interactive learning systems, for example, research of the TELMA project (Trgalova et al., 2009). In this section we present some results of our study. During the study we paid attention to the following features:

- correspondence of the environment to school curriculum;
- cognitively faithful solution generation possibility with explanation of steps;
- student activity in the environment and system activity;
- possibility for student to make decisions about solution path;
- possibility to make errors and system ability to diagnose them and react accordingly;
- what skills are learned while working with computer.

Alessi and Trollip (Alessi et al., 2001) presented a list of categories for math education software:

- drills,
- tutorials,
- games,
- simulations,
- hypermedia,
- tools and open-ended learning environments.

Handal (Handal et al., 2003) tried to estimate how relevant are those categories nowadays and provided some software examples for each category. He

concluded that categories of computer use in schools described by Alessi and Trollip are still a helpful framework for classifying web-based mathematics learning activities. In our study we concentrated on a slightly different grouping of software that we think is more suitable for the complex field of algebraic transformations:

- testing environments;
- tutorials and spreadsheets;
- computer algebra systems (CAS) and calculators;
- interactive problem solving environments.

1.2.1 Testing environments

Considering their method, testing environments are very close to drilling programs, as they present a certain set of questions and problems to be solved by students. In most cases, students can enter only the final answer to the problem. In case of complex problems, students have to do calculations on paper and enter the answer. The computer reaction is usually limited to correct/incorrect, but some testing environments provide a possibility to have a different reaction to certain student answers. General-purpose testing environments do not have any domain expert module, so there is no separate error diagnosis module or some solution generator built in. However, students can still learn by getting immediate response to their answers. One example of an online testing environment is the testing module of the Blackboard Learning System (Blackboard Learning System by Blackboard Inc). In this system, for example, the author of questions for the test can define the reactions of the system to some typical incorrect answers.

1.2.2 Tutorials

Nowadays it is possible to find quite many web-based or standalone tutorials for school mathematics and algebra on the Internet. Tutorials present information and also guide students through their learning processes. A tutorial usually contains an information part and some practical exercises similar to testing environments, where students usually only enter the final result. In comparison to testing environments, tutorials usually give feedback on the procedure to get the correct answer. According to Alessi and Trollip (Alessi et al., 1991), tutorials are effective for "presenting factual information, for learning rules and principles, or for learning problem-solving strategies", but not for learning to perform separate solution steps and make expression transformations.

MathAid (MathAid by MathAid) is an example of a typical tutorial system. It has a lot of structured material: for each topic, one explanation page and several practical problems where students have to enter the final result. It is possible to ask for help and then the system will generate a stepwise solution.

However, as expressions have a fixed structure, the solution is quite rigid, with only constants changing for different problems. MathCentre (MathCentre by Mathematics Education Centre) is another example of a tutorial system with different learning resources grouped by topics. Mathematics V10 (Mathematics V10 by EptSoft) is a further example of an interactive tutorial system. Under each topic, students can change certain parameters or generate random expressions to illustrate and explain the material. It has also a calculator with a good expression editor. However, it does not have a built-in testing component, so student learning in the system is quite passive. Tutorials usually include a domain expert module of some kind but in most cases it cannot be used for proper diagnosis of student errors.

A slightly different approach is used in applications, which are specifically designed for automatic generation of solutions, for example, Equation Wizard (Equation Wizard by ElasticLogic). In such a system, students can enter an expression (polynomial expression or linear equation) and ask the system to simplify or solve an equation. The system produces a step-by-step solution, making one simplification on each line and presenting a description of each step. The steps and transformation rules used are quite similar to those taught at school, so we can say that the solution generator in this system is cognitively faithful. In this system, students can enter only the initial expression, they cannot make any mistakes and, therefore, no error diagnosis is available. Student learning in the system is quite passive, although they can learn the solution algorithm and how to create individual solution steps, as those aspects are explained.

1.2.3 Computer algebra systems and applications

Another category that we have studied is computer algebra systems (CAS) and also algebra calculators. Many different CAS are used directly for educational needs, for example, Derive (Kutzler, 1996) or WIRIS (Xambo et al., 2002) and also algebra calculators like TI-92 (Kutzler, 2000). The use of CAS, such as Maple (Stephens et al., 1999), has some positive effects on student results, but the teaching methods using computer algebra systems are different from traditional teaching. It is not possible to use CAS for practicing problem solving (neither algorithms nor applications of single simplification rules), as those are not designed for such work. It is possible to get the final answer to a problem very easily but, until recently, systems did not generate or show the solution path. The commands or transformation rules implemented in CAS are usually too powerful for school education, making many simplifications in one step and sometimes producing results that are beyond the school curriculum (this can be configurable in some cases). Students cannot control the solution process by making decisions on the result, nor make any mistakes.

Although CAS themselves are not best suited for algebraic problem solving under student control, they can be used for this purpose in other systems.

Serving as a domain expert module of such systems, the core of a CAS can be used for application of some simplification rules, checking student results, providing feedback, etc. Error diagnosis and feedback to students is usually limited to indications whether an answer is correct or incorrect, except for some instances when certain typical misconceptions are checked separately. One example of such integration of CAS into a task oriented tutorial system is described by Postel (Postel, 1999). In the described system, students are able to solve equations or simplify algebraic expressions step-by-step under the control of the system. The Environment and Tutor modules of the system are custom-made while the Expert module uses MuPAD. The Expert module is used not only for checking students' decisions – like whether a given mathematical operation is correct and leads towards the solution or whether the result of a solution step is mathematically correct and compatible with the corresponding operation – but also for computing the result of a solution step and suggesting an appropriate mathematical operation as the next solution step. Its design is similar to rule-based problem solving environments considered below, but we mention it here as an example of CAS integration. Some other examples of interactive CAS and possibility to generate step-by-step solutions are presented by Maleševic (Maleševic, 2009), for example, when using the "Student" package in the Maple integration, the problem solution generated by the system is more detailed and contains more smaller steps. CAS are used in other systems for computer aided assessment (evaluate student answers and provide feedback), for example, STACK (Sangwin, 2007) uses computer algebra system Maxima, AiM (Klai et al., 2000) and Wallis (Mavrikis et al., 2003) use Maple. In these systems students have to enter only an answer to a question, an algebraic expression, and the system makes checks using CAS. Although it might be possible to build very complicated response processing trees (Sangwin, 2007), the system still reacts according to certain predefined responses. The system is not able to generate a solution or diagnose the exact position of a mistake in an answer. Help provided by the system is restricted to the same response processing tree that has to be defined for each problem, which can be time consuming. The student learns from immediate feedback but still has to do calculations on paper without any explicit checking.

1.2.4 Interactive problem solving environments

The final group of applications we have studied includes interactive problem solving environments, which belong to open-ended learning environments according to the categorisation of Alessi and Trollip (Alessi et al., 2001). In these environments problems are solved step-by-step under the control of the environment and according to a certain schema for making solution steps. We have identified the following schemas of problem solving in these environments:

- only rule/command based environments, where students select commands and subexpressions and the environment applies the rule automatically;
- only input based environments, where each step is freely entered by the student;
- combined rule-input based environments, where student makes transformations using some rules supported by inputting some parts of the result.

1.2.4.1 Rule-based environments

In *Rule-based* environments, when a student makes a solution step, he usually selects a transformation rule and, in some cases, a part of the expression to apply the selected rule to. The transformation itself is made automatically by the computer. Generally, in such environments, students can learn and practice the solution algorithm, but the learning of performing algorithm steps (details of operations) is passive, because the computer performs more work than the user. However, the student has some freedom in choosing a solution path and has a possibility to make some transformations before others.

The possibility for students to make mistakes is limited in rule-based environments – the only possibilities are selection of incorrect operands and, in some cases, selection of inapplicable rules. In some systems it is sufficient to select the whole expression for application of the rule and the system automatically identifies suitable operands. In other systems, after selecting a subexpression, the system provides a list of possible operations on the selected expression.

Although, for applying some rules, students need to input some additional information (for example, coefficient to multiply both sides of equation, etc.), these systems are not categorised in the group of combined rule-input systems, because the result is usually calculated automatically using this entered additional information.

Some examples of rule-based environments worth mentioning are:

- EXPRESSIONS (Thompson and Thompson, 1987);
- ALGEBRALAND (Brown, 1985);
- Mathpert (Beeson, 1990);
- MathXpert (Beeson, 2002);
- Aplusix (Nicaud et al., 1999);
- L'Algebrista (Cerulli and Mariotti, 2002);
- AlgeBrain (Alpert et al., 1999);
- Education Program for Gifted Youth (EPGY) (Ravaglia et al., 1998);
- Cognitive Tutor Algebra 1 (Cognitive Tutor by Carnegie Learning Inc.);
- Ms. Lindquist (Heffernan and Koedinger, 2000).

The largest and most popular of them, for example, Cognitive Tutor, MathXpert, Ms. Lindquist, Aplusix, are also evaluated in the TELMA project

research (Trgalova et al., 2009). A brief description of the listed systems and their features is given below.

EXPRESSIONS is one of the oldest, classic rule-based environments. It has two different representations of expressions: the usual (sentential) form and an operator-based tree. In order to perform a step, students should select a rule (button) and then a subexpression (by clicking on a node in a tree). The program changes the expression and the tree accordingly. In their study, Thompson detected typical errors that students made in the system: trying to apply an inappropriate rule to current expression and trying to apply a correct operation to wrong subexpression (in a tree).

The first version of the Aplusix environment enabled to practice factorizing polynomials by choosing an action, selecting an expression and, in some cases, entering additional information. The current version of Aplusix utilizes an input-only interface with possibility to select some simplification commands.

Mathpert and its evolution to MathXpert both follow the principle that students should not have a possibility to make mistakes. When solving problems students should select a subexpression, the system then displays the list of suitable rules and, after one of them is selected, the system applies it automatically. Therefore, students can err neither in selecting the rule nor in applying it. If the student is stuck and unable to proceed, MathXpert can offer different kinds of help, for example, generate the next step automatically.

The L'Algebrista system also follows the "no mistakes" principle and does not allow selecting mathematically incorrect subexpressions, for example, if a student wants to select a+3 in the expression 2*a+3, the system will automatically extend the selection to the whole expression.

There are other rule-based systems worth mentioning. In the AlgeBrain web-based intelligent tutoring system for solving equations, students should select operands (a term can be selected by clicking on its primary operator) and an operation. Like the systems mentioned above, the system does not allow selecting syntactically incorrect parts. The system proposes hints and animated feedback. A similar scheme is used in the Education Program for Gifted Youth (EPGY). In this system, in order to make solution steps, students usually have to enter some additional information. EPGY uses the kernel of Maple as an expert module. Cognitive Tutor Algebra 1 includes simplification exercises on polynomials and exponents (chapters 9 and 10) where students have to choose an operation for making a solution step. For some operations, the system asks to enter some additional information or select parts of expression to apply the rule to.

Ms. Lindquist is an algebra word problems tutoring software. The algebra model concerns symbolization, i.e., the task of writing an algebraic expression given a real-world problem context, which is considered as a major determinant of problem difficulty. Although students enter expressions similarly to input-only environments, the dialogue guides them to the solution step by step, making certain steps / operations at each step.

1.2.4.2 Input-based environments

In *Input-based* environments, when a student makes a solution step, he usually enters the result of transformation on a new line. This is more similar to paper-and-pencil solutions where students simply write transformed expressions line by line. In such systems the expression is usually copied to a new line and modified. Different techniques can be used for modification, for example, in Aplusix users can move parts of expressions with the mouse (for example, move terms to other side of equation, etc.). In other systems, simple keyboard input is used for entering result.

When solving problems students have the possibility to make many transformations on the same line. Input-based interface also enables students to make all the mistakes that are possible on paper. An exact error diagnosis is quite complicated for the system – similarly to paper solutions, there is no information on the student's intentions (what transformation rule and to what objects is the student trying to apply). Most input-based environments diagnose equivalence of two expressions, which is quite trivial in most cases of school algebra. The domain expert module of input-based environments usually does not provide a precise diagnosis of errors made.

When working in input-based environments, students practice both making separate steps and making decisions about the solution path. The solution algorithm strategy is usually not supported by the system – there are no rules for the system to give any hints. Furthermore, input-based environments usually do not generate any step-by-step solutions for students.

Some examples of input-based environments worth mentioning are:

- BUGGY/DEBUGGY (Brown and Burton, 1978; Burton, 1982);
- Algebra tutor (Anderson et al., 1990);
- Aplusix (Nicaud et al., 2004);
- Math-Teacher (Math-Teacher by MATH-KAL);
- Treefrog (Strickland and Al-Jumeily, 1999).

The first prototypes of input-based systems that tried to model student behaviour were created as early as in the seventies of the last century. BUGGY was the first diagnostic system based on "the Buggy model", where student's errors are diagnosed as a typical "bug" – a discrete modification to the correct skills. The system tried to identify a bug that could explain the student's answers. DEBUGGY, a development of BUGGY, is a much more sophisticated diagnostic system, that takes into account both the fact that more than one bug can cause the student's errors and the fact that sometimes there is no hypothesis explaining the student's behaviour completely, so that the system has to find a model that best explains it. Both systems allowed students to solve in-place subtraction problems.

Algebra tutor, an early version of Cognitive tutor Algebra 1, also required only entering of the result and the program tried to figure out, what step was performed, and to give appropriate feedback. However, authors of cognitive

algebra tutors found that, “The problem was that the students’ error might well have occurred at some intermediate step that the students were no longer fixated upon. It was very difficult to communicate to the student what the problem was.” (Anderson et al., 1990, p. 42).

Another problem that arises is the possibility for the theory of a domain to be completely specified (as it is possible, for example, to list a complete set of rules for arithmetic or high school algebra) – it is difficult, if not impossible, to enumerate all the misconceptions and other errors that may possibly be encountered in students’ work, even when one only considers the errors that students generally tend to make. That is to say, it is generally impossible to have a complete bug library. And even if it were possible to have, at the start, a bug library that contained at least the most common errors of a group of students, the results of (Payne et al., 1990) suggest that different groups or populations of students (e.g., students from different schools) may need different bug libraries.

Aplusix is an ILE for teaching and learning secondary school arithmetic and algebra. It lets students solve exercises and provides two fundamental feedbacks: it verifies the correctness of calculations and of the end of exercises. The current version of Aplusix diagnoses only non-equivalence of a new expression to the previous one. The authors of Aplusix are developing the program further. They are building a library of correct and incorrect rules, which can describe how one expression was transformed by the student to the next expression, and adding student modelling using conceptions (the models will be provided only for teachers, not students) (Nicaud et al., 2006; Nicaud et al., 2005). They are also planning to provide good feedback for the student from the calculated conceptions. When solving problems in Aplusix, the program copies the content of the previous line (expression, equation or system of equations) to the next line and the student should edit it to get the result of the step by typing or using drag and drop technique. The system is giving the student feedback about correctness of the step.

Math-Teacher is another example of a system with an input-only interface. It covers problems from different fields of mathematics, starting from arithmetical problems like calculation of value of numerical expression. It also contains problems from algebra like simplifying polynomial expressions, linear and quadratic equation solving, etc. However, it also contains material and typical problems from calculus and geometry. The expressions in Math-Teacher should be entered in a linear, Maple-like form. In most cases, there is a possibility to enter the final solution to a problem, although the system advises the user to reach the solution step-by-step. The program provides three basic types of feedback: correct, incorrect or syntax error. Some help (like hint on the last answer line) or information about the lesson is provided. Finally, Treefrog is yet another example of an input-based system. Similarly to Math-Teacher, students enter the next line of expression and receive feedback whether it is correct or not. Students can solve problems step-by-step or enter the final result at once. In case of incorrect input, it provides a hint about which operation should be

performed first, based on the priority of operations and, in case of equations, a solution algorithm.

1.2.4.3 Combined rule-input based environments

The third group of systems uses direct input while also enabling selection of transformation rules and objects (*combined rule-input*). The intermediate version of Cognitive Tutor: Algebra (Anderson et al., 1995) was a system where the student could decompose a result of calculation into substeps recursively until primitive steps were reached. At each substep, the student had to choose the operation that should be performed on the expression, enter the arguments to proceed to this operation, and enter the result. The tutor embedded boxes on top of boxes to indicate the levels of embedded goals. However, after evaluation the authors found that the tutor did not give positive results and "… the major reason for the lack of effect was that there was a large difference between the tutor interface and the interface used in class (i.e., paper and pencil). It was just not obvious how to map the boxed representation of algorithmic decompositions to the linear line-by-line transformations…" (Anderson et al., 1995, p. 183).

In my Master's thesis (Lepp, 2003b, Lepp, 2003a) I also tried to implement a system with a combined step making mode (input result of application of transformation rules) as a prototype for T-algebra. The interface was quite similar to paper and pencil working mode but the input stage of transformation rules required entering different parts of expressions and the rule argument selection was too dependent on the chosen rule.

The supervisor of the author's research had experience with step-by-step problem solving environments from early nineties. A package for exercises in Mathematical Logic was developed in 1988–91 at the University of Tartu (Prank, 1991). One of the programs was an interactive environment for stepwise solution of formula manipulation exercises in Propositional Logic (expression of formulas through $\{\&, \neg\}$, $\{\vee, \neg\}$ or $\{\supset, \neg\}$ and finding normal forms). The first version of this program had an input-only interface. At every step the student had to type a new formula on the next line (with some copy-paste possibilities). The program checked the syntax, equivalence to the previous line and whether the target form of the expression was reached. Prank saw that the greatest problems were posed by errors caused by misunderstanding the order of operations. The system was generally unable to diagnose them without explicit information about the object of conversion. In the second version (Prank and Viira, 1991) the step dialog was built using an Object-Action scheme. The student had to mark a subformula and then convert it to the result of the step. The strings before and after the marked subformula were copied automatically. For the second substep, the program had two working modes: input and selection of a conversion rule from the menu. As a result, the program was able to verify separately the selection of operand and the performed conversion. This addition of a marking phase gave a level of feedback that was sufficient for that

group of users (second-year students) and it was decided that there is no further need to make it more precise.

1.2.5 Task-oriented tutorial systems

Holland (Holland, 1994) gives another overview of some intelligent tutoring systems from mid 90-s and proposes to use subclasses of tutorial systems for teaching mathematics. He defined the so-called subclass of task-oriented tutorial systems (TTS). According to Holland, two essential educational goals of a TTS are:

- The students know which operators are required or permissible for solving the task (e.g., transformation rules for transforming terms or equations). What is to be exercised here is the skill to apply the operators in the context of a problem solution consisting of several steps.
- The students should know and be able to apply heuristic methods to solve problems.

When formulating goals for the T-algebra project, we tried to fulfil both these educational goals: student has to be able to practice both applying separate transformation rules and choosing correct rules (finding solution path) to solve the problem.

1.2.6 Conclusion

In previous sections we have reviewed different learning tools and problem solving environments. We think that students can get most effect from problem learning environments as they are using "learning-by-doing" technique; therefore, we focused mostly on them in our research. However, we have pointed to some problems in different solution step making approaches.

Some systems do not allow students to make all decisions when making steps (for example, rule-based systems do not allow entering the result of application of a rule or to select objects of a rule), and also do not allow making certain typical errors. Learning from own mistakes is the most effective way and feedback of the system is extremely important for that. However, as we have seen, most environments do not provide an exact error diagnosis to support the student.

From this point of view, the systems with a combined rule-input based solution step dialogue are most suitable for students, although there are very few examples of systems utilizing this approach. In such a dialogue the students make all decisions about the solution step, they can make the same errors that are possible on paper and, for the system, this dialogue creates a possibility to give most accurate diagnosis based on student decisions. Although Anderson faced problems in Cognitive Tutor: Algebra (Anderson et al., 1995, p. 183), those were caused by the difference between the user interface and the usual paper-and-pencil approach. My supervisor also had a positive experience of using this combined rule-input dialogue in an environment for solving problems

in Mathematical logic. Therefore, we have chosen the rule-input based solution step dialogue for our project.

1.3 Problem statement

The goal of the T-algebra project was to design, implement and test a problem solving environment of a new kind for proper learning as well as for assessment and diagnosis of gaps in the knowledge and skills. The project covers different problems from four fields of school mathematics and algebra (grades 4–8):

- calculation of the values of numerical expressions;
- operations with fractions;
- solving of linear equations, inequalities and linear equation systems;
- operations with exponents, monomials and polynomials.

When designing the functionality of T-algebra, we focused on certain features that we decided the system should have:

- enable students to solve problems step-by-step and line-by-line in a manner similar to solving problems on paper;
- allow the student to make all the necessary decisions and calculations at each solution step and explicitly provide this information to the system;
- leave an opportunity for the student to make the same mistakes as on paper;
- give the possibility to learn both the algorithms and their steps in detail;
- contain such dialogue that allows the program to understand all decisions made by students (collect direct information about chosen operation, selected operands, entered result);
- contain such domain expert module, which would be able to not only give an answer, but to show a solution path using the designed interface;
- be intelligent enough to check the knowledge and skills of the student, understand mistakes, offer feedback and advice.

These features resulted in decisions about solution step dialogue and user interface for making solution steps, applications of domain expert module, error diagnosis, etc., which are described in Chapter 2 of the thesis. The parts that were in my responsibility in the project are listed in the next section.

1.4 Contribution of the thesis

As mentioned, this thesis is based on the work that has been done for the T-algebra project. The thesis presents general decisions and solutions of the problem solving environment as well as particular decisions and implementation details of rules and problem types for the fields I was responsible for: exponents, monomials and polynomials.

In T-algebra we used the combination of rule-based and input-based approach for making solution steps. We called this the "action-object-input scheme". The decisions and the program implemented in my Master's thesis (Lepp, 2003b, Lepp, 2003a) were used as a prototype for this scheme. It is hard to identify the particular contributions of each team member to designing and implementing the general ideas of the action-object-input scheme. Chapter 2 of this thesis presents different aspects of the system, even if it is not the main contribution of the author. In some cases, materials are based on articles presented by other team members (for example, 3 input modes for the input stage of a step) but the description of a feature is included in the thesis to facilitate a better understanding of the system and other contributions of the author.

The main contribution of the author of the thesis can be divided into three large parts:

- design decisions and implementation of certain general features of T-algebra;
- study, design decisions and implementation of problem types and rules for a specific domain – domain of exponents, monomials and polynomials;
- experimenting efforts in evaluating the general features, like solution dialogue of T-algebra as well as domain specific decisions, problem types, transformation rules, etc.

When designing and developing T-algebra, some features were designed and implemented mainly by the author of the thesis:

- participation in design and implementation (project seminars with school teachers and authors of school textbooks) of action-object-input solution step dialogue (presented in section 2.5);
- implementation of expression parsing and rendering in expression editor;
- design and implementation of expression editor features to support the solution step dialogue (presented in section 2.7);
- design and implementation of extension to the action-object-input dialogue (presented in section 2.6);
- design and implementation of general principle of error diagnosis and categorization (presented in sections 2.8.6 and 2.9);
- internal design and implementation of basic classes of rule and problem type and their usage in general solution algorithm, error diagnosis, etc. (presented in sections 3.2.2 and 3.3.1).

The author's domain specific (domain of exponents, monomials and polynomials) contribution is:

- study of problems solved at school in the chosen domain and design of problem types for T-algebra (presented in section 3.1);

- study of school textbooks and student solutions in order to extract transformation rules needed in this domain (both domain specific and learned before), design of transformation rules in T-algebra, discussion of the design with school teachers and publication of decisions (presented in section 3.1);
- investigation of typical errors for the selected domain, based on an experiment with students (presented in section 4.1) and related works in order to design error diagnosis for designed transformation rules (presented in section 2.8.6);
- implementation of identified problem types, including error diagnosis, conditions for starting and ending expressions, solution algorithm, etc. (presented as different subsections in section 3.3);
- implementation of domain specific transformation rules, including error diagnosis and domain expert for application of implemented rules (presented as different subsections in section 3.2).

The author of the thesis participated in numerous experiments and trials with students and teachers (results are presented in Chapter 4):

- experimental validation of created dialogues with students and teachers;
- evaluation of the environment in the chosen domain of exponents, monomials and polynomials, trials with real students;
- investigation of student solutions and their mistakes when solving problems in T-algebra (particularly problems of the chosen domain) and comparison to the results of the experiment for collecting mistakes from paper solutions.

1.5 Structure of the thesis

In addition to this introduction, the thesis contains four chapters, a conclusion and appendices. General description of T-algebra is based on the papers by the author presented in the List of original publications and also on papers presented by other T-algebra team members. The thesis also contains a large chapter on the author's domain specific contribution.

Chapter 2 (T-algebra interactive learning environment) thoroughly describes the design decisions for T-algebra and its main components. This chapter concentrates on the parts of the system that I was responsible for (either design, implementation or both). The first part of this chapter describes T-algebra as a task-oriented system and presents its components. The second part introduces the problem solving environment T-algebra in general. The third part of Chapter 2 gives a brief description of the problem composing program and its features (including random expression generation). The fourth part gives an overview of expressions supported by the system. The fifth part presents the design of solution step dialogue as an extension to the dialogue tried in the Master's thesis. The next part lists extensions to the dialogue designed for the needs of

certain rules. The seventh part of Chapter 2 provides details about the implemented expression editor for supporting the dialogue. The eighth part of this chapter gives an overview of different applications of the domain expert of T-algebra. The last part describes different error categories and statistics collected by T-algebra.

Chapter 3 (Problems, rules and algorithms in the domain of exponents, monomials and polynomials in school textbooks and in T-algebra) describes the domain of exponents, monomials and polynomials. First, exploration of mathematics school textbooks is presented (definitions, algorithms, problem types). The second part of this chapter describes the rule engine of T-algebra, gives information about common checks for the rules, different usages of the rules and also some rule implementation details. This part also provides a detailed description of the domain specific rules, designed for solving simplification problems (exponents, monomials and polynomials) in T-algebra. The last part similarly describes different aspects of problem types in T-algebra: general information, usage of problem types, common checks, and some implementation details. This part also presents composed problem types and their solving algorithms in the chosen domain.

Chapter 4 (Conducted experiments) describes four different experiments conducted to validate the user interface, to evaluate the created interactive learning environment (the domain of exponents, monomials and polynomials), to investigate mistakes made by students during problem solving in T-algebra, and to compare them with mistakes made on paper.

The thesis also contains four Appendices. Appendix A lists the problems used for practice exercises in one of the experiments on paper. Appendix B contains the list of problems from the problem file that was used in another experiment with students. Appendix C provides the list of all error categories diagnosed in T-algebra. Appendix D presents, in Backus-Naur Form, a full description of expressions supported in T-algebra.

2 INTERACTIVE LEARNING ENVIRONMENT T-ALGEBRA

One contribution of this thesis is to design and implement an interactive learning environment for solving polynomial simplification problems. This goal was realized as a part of T-algebra interactive learning environment, which enables step-by-step solving of algebra problems in four areas of school mathematics:

- calculation of the values of numerical expressions;
- operations with fractions;
- solving of linear equations, inequalities and linear equation systems (my contribution to this environment);
- simplification of polynomials.

T-algebra was developed from 2004 by the Master's and Doctoral students of the Institute of Computer Science at the University of Tartu (Dmitri Lepp – simplification of polynomials; Marina Issakova – solving of linear equations, inequalities and linear equation systems; Vahur Vaiksaar – operations with fractions and calculation of the values of numerical expressions) and under the supervision of their instructors (Rein Prank – project manager, Eno Tõnisson). Rein Prank also implemented the random expression generator for different types of tasks.

While designing the most important part of the system – solution dialogue and transformation rules – help of consultants was used. Great contributions were made by mathematics teachers Mart and Maire Oja and the authors of textbooks for schools Tiit Lepmann and Anu Palu. This version is developed as a project financed by the 'Tiger Leap' computerization programme for Estonian schools. In the end of 2008, the first release of T-algebra was completed and made available for all Estonian schools.

In this section I describe different general aspects of design and implementation of T-algebra. I mostly describe those parts of the system I was responsible for but, to give full overview of the system, I shortly describe also those parts of the system that were designed and implemented by other team members. My main contributions in the general part of the system are expression text representation and expression editor, extension of action-object-input scheme (the basis of this scheme was tried in my Master's thesis (Lepp, 2003b, Lepp, 2003a)), design decisions and implementation of base classes for rule and problem type (see sections 3.2.2 and 3.3.1), rules and problem types for the field of simplification of polynomials (described in sections 3.2 and 3.3).

Design decisions and implementation details are published by me and other T-algebra team members (Issakova and Lepp, 2004; Lepp, 2005; Lepp et al., 2005; Issakova et al., 2005; Lepp, 2006a; Lepp, 2006b; Lepp, 2006c; Lepp, 2006d; Issakova et al., 2006; Issakova, 2006; Prank et al., 2006a; Prank et al., 2006b; Prank et al., 2007) and some of those articles were used as a basis for the material in this section.

2.1 T-algebra – a classic task-oriented system

According to Postel (Postel, 1999) and Holland (Holland, 1994) a typical *task-oriented tutorial system* (TTS) consists of four main parts:

- *Expert Module*: The system must be an expert on the subject in question. The system must be able to answer student questions, to solve tasks put to the student, and to analyse student answers for bugs and misconceptions.
- *Environment Module*: The system must know how to present the subject matter in an appropriate way, and must allow the student to enter his/her problems in an appropriate way.
- *Tutor Module*: The system must have knowledge about the curriculum and offer the student a repertoire of tutorial strategies in order to be able to intervene tutorially in an optimal way at any point.
- *Student Module*: The system must have an idea of each student's knowledge and skills and be able to adapt its own hypothetical student model dynamically to the student's learning progress.

Postel formulated some key features for each component, which are also valid for T-algebra. In this section we try to describe all those essential components of T-algebra.

The first component, the *Expert Module*, should (Postel, 1999):

- find a solution for each problem of the problem class. The solution is appropriate to the knowledge state of the student;
- be able to check a student solution for correctness and quality. It is able to classify errors as they occur;
- be "transparent", that means, it uses only knowledge and methods the student is supposed to learn and use.

In T-algebra, the expert module is able to generate a solution for all problems using exactly the same set of transformation rules that is available for the student. A specific set of transformation rules is defined for each problem type, based on the curriculum and Estonian schoolbooks. See details in sections 2.8, 3.1, 3.2 and 3.3.

The second component, the *Environment Module* for the dialogue between student and tutor should (Postel, 1999):

- minimize the number of actions (keystrokes, mouse clicks, etc.) which are necessary for the communication with the system;
- represent the problems in that way that the representation reflects the structure of the problems;
- give as much information as possible about the problem-solving process;
- detect possible mistakes done by the student during the communication (e.g., input of syntactical correct mathematical expressions).

We tried to make the user interface and solution step dialogue in T-algebra similar to working on paper. The estimated amount of input required from the student is similar to pure input environments and is much smaller than when solving problems on paper (Prank et al., 2006a). See some details about the environmental module further in sections 2.5, 2.6 and 2.7.

The third component for tutorial aspects, the *Tutor Module*, should (Postel, 1999):

- monitor each step the student makes toward a solution. For this, the tutor makes use of the expert module;
- offer help at any stage in the problem-solving process to the student in form of hierarchically graded help. Help begins with general heuristic hints and ends with prescribing the very step toward a solution the expert would have chosen in this situation.

Different ways of using the expert module in T-algebra were described by Issakova (Issakova, 2006). One of the usages is providing feedback and different kind of help in every situation depending on the current situation/ expression and the stage of the step. For details see sections 2.8.5 and 2.8.6.

The fourth component, the *Student module*, keeps statistics about the student. T-algebra monitors and logs all errors and help usages, tries to classify errors to some predefined classes and monitors overall progress of the student. For details see sections 2.8.6 and 2.9.

2.2 Brief introduction to problem solving environment T-algebra

The T-algebra package consists of two programs, one is for students and the other is for teachers. The student's program is meant for students for solving problems but also for teachers for checking solutions, reviewing student errors and collecting statistics from student solutions. Students can also use these features to revise their own solutions, check the errors made, etc. The teacher's program is meant for composing problem files.

The main problem solving window of the T-algebra student's program is shown in Figure 2.1. The window has been divided into two logical parts; any of those can be hidden. The left-hand part contains a field displaying a list of problems in opened file. The list includes expressions and formulations of problems. In addition, completed problems (green background) and the active problem (surrounding red box) are marked on the list.

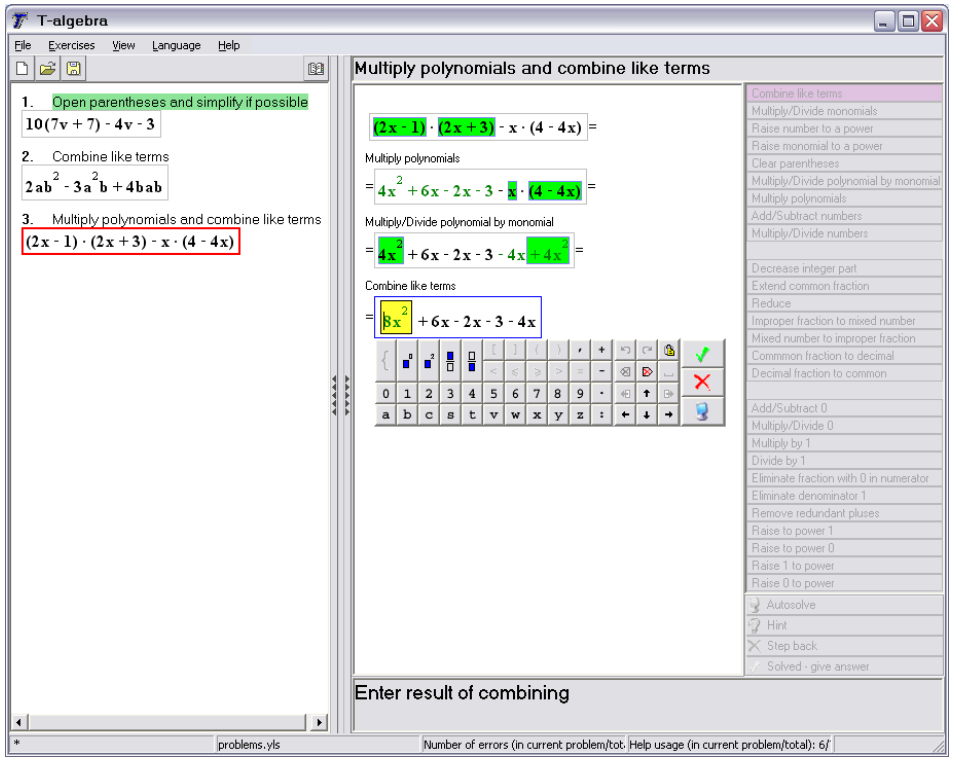


Figure 2.1. Main problem solving window of T-algebra student's program

The right-hand part of the window is meant for problem solving. It contains a list of available transformation rules, buttons for confirming the answer, asking for hints, etc. At the top of the right side, there are instructions for the whole problem, and at the bottom, instructions for the current stage of solution step. The window also contains solution steps (expressions) already performed by the student together with an expression editor and a virtual keyboard for entering the result of operation.

The sample window shows the resolution process for the simplification problem $(2x-1)\cdot(2x+3)-x\cdot(4-4x)$. The solution is not yet complete, but some steps have already been taken. At the first step, polynomial multiplication was performed. The resulting polynomial was entered in the result. At the second step, polynomial and monomial were multiplied the same way. For the last completed step, the operation *combine like terms* was picked and two like terms were selected. The selection has been confirmed and, as the next step, the user would have to enter the result of the combining in the yellow box on the next line.

2.3 Brief description of problem composing

In addition to a problem solving environment, the T-algebra package also includes a program that is meant for teacher usage – program for composing problem files. Each problem file contains a set of problems, for example, for one lesson or for home practice, etc.

When composing a problem, the teacher has to define the following attributes:

- problem field, problem type;
- text of the problem (for example *Simplify*, or *Multiply polynomials and simplify*);
- decide whether initial expression is entered or generated;
- type of expression (if an expression is generated);
- initial expression for the problem, values of parameters if any (if an expression is entered);
- what input mode is used (free, structured, partial);
- what kind of hint / help is enabled or disabled.

The last two can be the same for all problems in the file if they are set in general properties of the problem file. Figure 2.2 shows a screenshot of the teacher's program during composition of a problem (initial expression is entered).

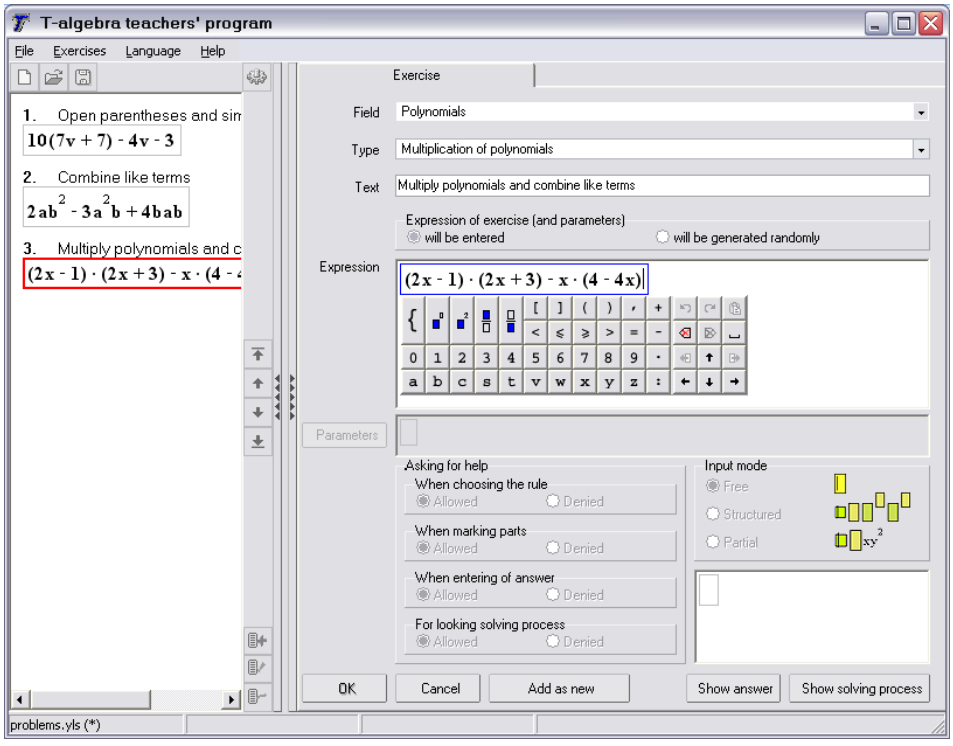


Figure 2.2. Problem composing in T-algebra teacher's program

Before a problem is added to the list, T-algebra performs the following checks:

- all necessary fields are filled: problem type, text;
- in case an expression is entered, T-algebra checks whether the initial expression is not empty, is a syntactically correct expression and whether the expression is suitable for the problem type (the problem is solvable by means of T-algebra rules designed for this problem type);
- if any additional parameters (for example, values of variables) had to be entered, T-algebra checks correctness and suitability of those;
- if an expression is generated, T-algebra checks whether the type of expression is selected.

While composing the problem, the teacher can check the automatic answer (small box and button in the right bottom corner of the window) or generate the solution (by means of the same rules that students may use) and decide whether the initial expression of the problem is suitable for the students (for example, whether all coefficients in the answer and in each line of the solution path are integers, etc.).

2.3.1 Random expression generation

We have implemented a random task generation in T-algebra. This creates a possibility to prepare many different problem files (for example, in many variants) easily.

For each problem type in T-algebra, we have implemented several types of random expressions that are generated – most of them are similar types to those used in schoolbooks. For example, in case of problem type *combine like terms* the random expression types are the following (the list includes a short description and a sample expression):

- "2 variables. 2 groups of like terms. Like terms have identical form", sample expression $-5b^2-2b^2t+10b^2-3b^2t$;
- "2 variables. 3–4 groups of like terms. Like terms have identical form", sample expression $-4ys^3+8y^3s+2ys^2-y^3s-7y^3s-7ys^3$;
- "2 variables. 2 groups of like terms. Like terms can have different form", sample expression $-7bc^3+9c^2cb-3c-3c^3b$;
- "3 variables. 3–4 groups of like terms. Like terms can have different form", ex. $-7tb^4w^3+5w^3b^4t+6bw^3t^4b^3-8wbw^2b^3t-5w^5t^3b^2-3bw^3tb^3$.

From the description of expression types it is clear that either 2 or 3 different variables are used in expressions, 2 or 3–4 groups of terms are used (some could be also single terms without pairs to combine with) but at least one pair of like terms should be present. The last option (same or different form of like terms) defines whether all terms of one group have identical variable parts (order of variables, powers) or different (for example order of variables can be different $-7bc^3-3c^3b$ or the same variable can exist multiple times, not in a normal

form $-7bc^3+9c^2cb$). T-algebra allows combining terms in different form but it is more difficult for the student to identify such like terms.

There are two ways how to use random expression generation in T-algebra. The first possible way to use random expressions is to save the random expression type in the problem file. Each time the problem file is opened in the T-algebra student's program, new expressions are generated for solving. Using this option enables to give students the same problem file and be quite certain that all students get different problems to solve (for example, to prevent students from copying each other's solutions during a test).

Another way is to generate a set of sample problems and choose one for solving or edit it further. In this case the teacher can easily create a problem file containing different problems without entering expressions himself. The selected sample expressions are saved to a file and every time the problem file is opened the same expressions are solved.

Figure 2.3 shows random expression selection in the problem composition window. On the left, an expression type can be selected for the given problem type. The right panel displays generated examples; it is possible to ask T-algebra to generate another set of examples. There are buttons for deciding whether one of the examples will be used as the initial expression or whether an expression will be generated when the problem file is opened in the student's program.

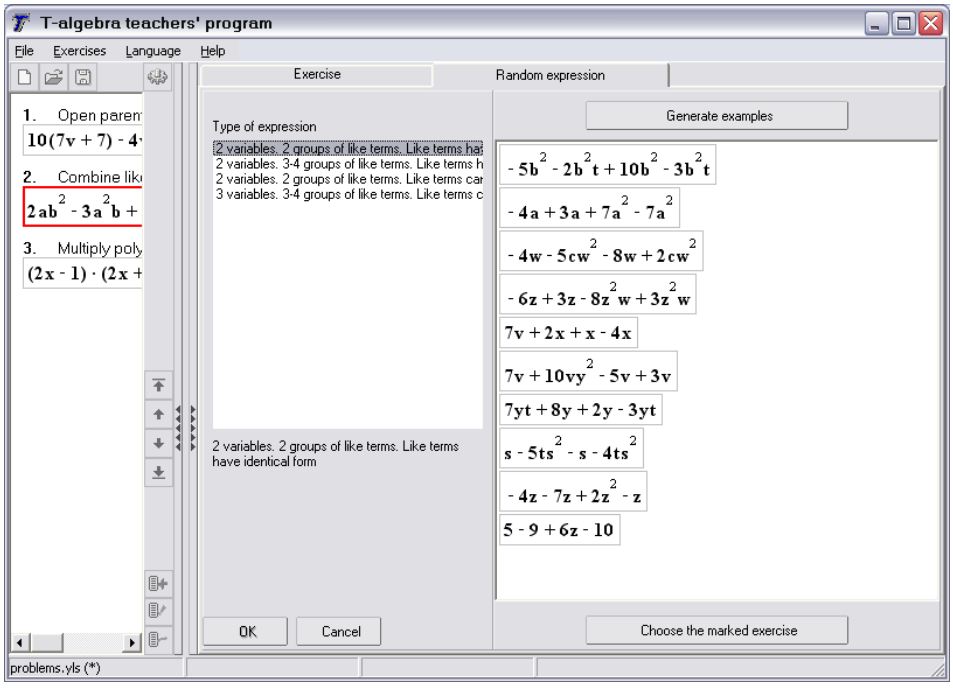


Figure 2.3. Random expression generation in T-algebra teacher's program

2.4 Expressions in T-algebra

The main objects that T-algebra is working with are algebraic expressions. In this section we describe, which expressions are allowed in the program, i.e., which expressions are treated as correct.

Problems solvable in T-algebra belong to four fields of mathematics: calculation of the values of numerical expressions; operations with fractions; solution of linear equations, inequalities and linear equation systems; polynomial simplification and factorisation. Thus the expressions from T-algebra also belong to these fields.

We define expressions in T-algebra the following way. Elementary expressions (or basis) are integers, decimals and variables. Algebraic expressions are composed by recursively applying different operations (unary – and +, binary +, -, *, /, exponentiation, fraction). Created algebraic expressions can be connected using equality and inequality signs (equations grouping into systems). A list of possible operations is presented in Table 2.1 (page 48). A full Backus-Naur Form description of expressions in T-algebra is presented in Appendix D. Here are some examples of correct expressions:

- $\frac{1}{2}-2\frac{2}{3}$;
- $x^2(1+x)^2$;
- $2x-2\le 3-x$.

The following expressions are treated by the program as incorrect:

- $(1+x)\frac{1}{2}-3\frac{0,2}{3}$, because decimal is not allowed in mixed numbers;
- $a2b+2bc3av-(x-1)^{-4}$, because a multiplication sign is required in monomial multiplication, and constants are not permitted between variables in monomial multiplication.

The editor does not put any quantitative constraints on expressions. The editor also supports several-storied fractions and exponentiations, etc. However, almost all problem types define some constraints on expressions (described in section 3.3.5). For example, when solving linear equations, all expressions have to be linear equations – they have to contain the equality sign and exactly one variable. In almost all problem types, the expression cannot contain fractions with variables in denominator.

There is one expression form, which created some problems for us. When defining the order of operations, textbooks declare that operations with the same priority are to be applied from left to right. For instance, expression $8a^4(a+b)^3:2a$ would mean, that we divide the first part $8a^4(a+b)^3$ by two and then multiply by a. The multiplication sign is often omitted in expressions. So expressions $2a(a+b)$ and $2a\cdot(a+b)$ have the same meaning. A problem arises when we have both division and multiplication in an expression. Some

Estonian schoolbooks mean by $8a^4(a+b)^3 : 2a$ that the first part of the expression $8a^4(a+b)^3$ is divided by monomial $2a$. The Council of Mathematics at the Ministry of Education of Estonia asked us to support the same form in T-algebra. Therefore, T-algebra considers multiplication without sign as a higher priority operation than ordinary multiplication or division (as if part of the expression were put into parentheses). For example, in expression $8a^4(a+b)^3 : 2a(a+b)a \cdot (a+b)$ the first part $8a^4(a+b)^3$ is divided by $2a(a+b)a$ and multiplied by $(a+b)$. However, we ourselves are not satisfied with this solution.

2.5 General Action-Object-Input dialogue scheme in T-algebra

T-algebra enables step-by-step problem solving. There are two basic possibilities for taking a step in interactive programs (discussed in section 1.2.4): free input of the step result (for example, in the Aplusix system (Nicaud et al., 2004)) or conversion by some rules or commands (for example, in the MathXpert system (Beeson, 2002)). T-algebra combines both these options – conversion by rules is supported by the input of some parts of the result. Each solution step in T-algebra consists of three stages:

1. selecting a transformation rule (action),
2. marking the parts of expression (object),
3. entering the result of the application of the selected rule (input).

Hereinafter we will refer to this scheme as the Action-Object-Input scheme after its three stages (Issakova et al., 2004). This type of scheme was first used in the program Polynom developed by me as part of my Master's thesis (Lepp, 2003a, Lepp, 2003b). The following simple example shows, how a student would complete the stages of dialogue and how these stages correspond to the solution algorithm taught at school. Let the problem be the following: simplify the expression $5x^2 - 6x + 4x - 2x^2$. When solving the problem on paper, the student would at first examine the expression and then decide to combine like terms. Then he would underline the like terms he wants to combine and enter the resulting expression after the equality sign. The program follows principally the same scheme of actions. The corresponding solution step consists of the following three stages:

1. Selecting a transformation rule: the student selects from the rule list the rule of combining like terms – the program allows selecting any rule without checking, whether it is possible to apply such transformation at this stage.
2. Marking parts of expression: the student marks all the monomials similar to x^2, using the mouse and selection buttons – the program checks, whether the selected parts of the expression are actually like

terms, and it also checks, whether these terms can be combined (i.e., whether they belong to the same polynomial).

3. Entering the result of the application of the selected rule: the program copies unchanged parts of the expression onto the next line and asks the student to enter the resulting monomial or its parts. Depending on the input mode, different boxes are shown (see details in section 2.5.2). The program checks, whether the entered parts are correct and the whole expression is equivalent to the expression displayed in the previous line, and then displays the resulting expression in the next line of the solution.

This list should provide an idea of the connection between the actions of the student and the program, what and when is checked by the program. If an error message is displayed at any checking stage during solving the problems, the student must first correct the error himself or let the program correct the error in order to proceed to the next stage. For each stage of the solution step, the program gives specific instructions ('Choose the rule to apply next', 'Select terms to combine', etc.). The student can cancel the step at any moment. It is also possible at any stage of the step to ask the program for help and let the program complete certain stages automatically (see details in section 2.8).

2.5.1 Action-Object and Object-Action in general and possibility in T-algebra

Interactive programs in which the user processes some kind of objects (text, image, table, etc.) step by step usually allow the user to apply different menu-selectable operations. The user can apply operations in a different order. If the operations are applied to the objects with the same structure, it is normal to use the so-called Object-Action scheme in which the user first selects objects and then chooses an operation to apply to these objects. Such scheme is used, for example, in text editors for changing the font of a paragraph, copying text, etc. Most computer algebra systems also use this scheme.

However, when the arguments of different operations have very diverse structures (monomials, polynomial, parentheses, etc.), it might be difficult to apply the Object-Action scheme. It is not clear before the operation is selected, what information needs to be entered to specify the object (whether the object is a monomial or a polynomial, an expression in brackets, an exponential expression, etc.). In this case, an Action-Object scheme is preferable, i.e. the user first selects an operation and then marks the objects to which the operation will be applied. Similarly, it is not possible to use the Object-Action scheme in dynamic geometry programs, for instance, where the order of objects is important (for example, centre of circle and point on a circle, etc.).

The Action-Object scheme is more suitable for working with the resolution algorithms used at school. The algorithm tells the student, what step should be taken next (what operation). So the student thinks of the next operation and then

tries to find objects and decide whether this operation is possible at this moment.

In the beginning we also chose the Action-Object scheme for T-algebra. As each rule can only work with certain types of objects (monomials, polynomials, expressions in brackets, variables, fractions, etc.), it should be clear beforehand, which objects ought to be marked. After the transformation rule (action) is selected, T-algebra is able to provide rule specific instruction on what objects should be selected. In addition, after selection of the rule, T-algebra is able to help the student and select suitable objects for the operation.

However, after the first trials we also enabled the Object-Action scheme, as we saw that students tried to select objects before the rule. As the user interface for selection of objects is common for all rules, we were able to do that easily. Students are now able to select objects (parts of expression) and then decide what to do with them. The only downside of this is that T-algebra does not provide any special instruction about objects to be selected and also cannot help the student in selecting those before the action is selected.

In T-algebra, the Action-Object scheme was upgraded with a third component – entering the parts of the resulting expression (Input). This gives the student the possibility to participate in the solution process. It also enables the program to check the knowledge and skills of the student.

2.5.2 Stages of the Action-Object-Input scheme

In this section we describe the details of design, implementation and user interface of individual stages of the dialogue.

2.5.2.1 Selection of operation

At the first stage of a solution step, the student has to select an operation – click on one transformation rule on the list on the right of the solution window (Figure 2.1). The set of rules displayed in the menu depends on the problem type. Rules designed for the domain of powers, monomials and polynomials are thoroughly described in section 3.2. It is possible to change the rule before the selection of objects is confirmed without recording any error, even if a non-applicable rule is selected, as this is not confirmed by the student yet.

T-algebra does not check the selection of rule separately – it is checked together with objects. However, it diagnoses separately whether the rule is applicable to any objects in the current expression – if not, a separate error message is shown to the user after selection of object is confirmed.

2.5.2.2 Selection of objects

In addition to selection of the operation, T-algebra requires selection of operands. Unlike many other programs, T-algebra requires precise marking of operands for diagnostic purposes. For example, for the operation *Combine like terms*, the student should mark only those terms that will be actually combined.

A special multiple selection mode of the expression editor is implemented for selection of objects (see details in section 2.7.4.1) – in this mode, the expression editor of T-algebra enables to mark more than one piece of the expression.

After selection of objects is confirmed, T-algebra checks if the rule is applicable to the selected objects (see details in the description of rules in section 3.2).

2.5.2.3 Input of the result

At the third stage (Input) of each step, the student should enter certain parts of the expression that result from the previously selected operation. The program generates an expression on the next line, based on the selected rule and marked parts, and leaves blank certain important parts of the new expression. When solving problems at school with paper and pencil, the students always have to write an expression of the same length after the equality sign. Consequently, they try to reduce their workload by making several transformations at once. The program makes the work easier for the students by copying the parts of the expression that remain unchanged so that the students would have to enter only the parts that were modified. As a result, only one transformation can be made in each step. This makes it easier for the program to check the solutions and gives a better overview of the student solution to the teacher.

The parts of the expression that the student has to enter are highlighted with yellow boxes. The form and the number of user-definable parts depend on the selected rule, marked parts and the mode (see details and examples in section 2.5.3). While entering the results, the program protects other parts of the expression from modification – the expression can be modified only in highlighted locations (see details about constrained input of the expression editor in section 2.7.4.2). This makes it easier for the program to check the solution and, in addition to checking equivalence between the new expression and the previous one, it also enables checking the correctness of separately entered parts, thus improving the overall responsiveness of the program to errors.

2.5.3 Three input modes

As mentioned, the Action-Object-Input scheme was first tested in my Master's thesis (Lepp, 2003b, Lepp, 2003a), which serves as a prototype of T-algebra. The input forms in that program were designed separately for each conversion rule, trying thereby to minimize the input so that only critical information for the particular operation would have to be entered. The form and number of parts that could be entered became too varied for different rules and the user interface of the program became too confusing. In T-algebra we try to design, uniformly for all rules, three fairly standard input modes, which we have named free input, structured input and input of some components. Free input mode is easily comprehensible (it is similar to working on paper) and it can be designed for each rule. Structured and component input modes are more specific. The

program helps the user in a certain way, whether by indicating the structure of the result or even by filling out a part of the result. The input mode is selected by the teacher during problem composition. It turned out that there are some rules for which it is impossible to apply all three input modes.

At the third stage (Input) of each step, the student should enter certain parts of the expression that result from the previously selected operation. The program generates an expression on the next line, based on the selected rule and marked parts, and leaves blank certain important parts of the new expression. Depending on the input mode used, the number and types of empty boxes can be different. The following subsections provide details about each input mode (Issakova et al., 2005).

2.5.3.1 First input mode: free input

In the free input mode, the program generates one input box (or two boxes in case of some rules with fractions and equations) inside the expression on the next line instead of marked parts (Figure 2.4). The input box is in the same position as the first marked part in the expression on the previous line. The student should enter in the box one expression, replacing the whole marked part from the previous line. In the free input mode, the student should also create such non-linear elements of the sub-expression as fraction lines and exponents if needed. Even though the name of the mode is free input, input is still restricted to some extent. The editor gives the student freedom for entering but, after the input, the program checks not only syntactical correctness of the expression and equivalence to the previous expression, as for example in APLUSIX (Nicaud et al., 2004), but in some cases also the correctness of applying the rule.

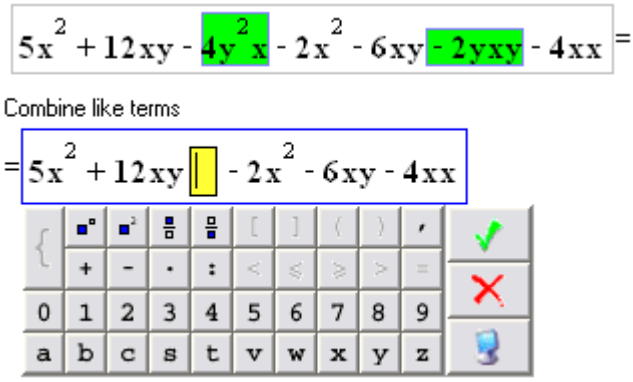


Figure 2.4. Free input

For example, in Figure 2.4, the rule *Combine like terms* was selected and two like terms were marked. After the input is confirmed, the program first checks whether the entered part is a syntactically correct expression, then whether the entered part is a monomial (because the result of combining like terms should be a monomial) and then whether the entered expression is equivalent to marked parts. Finally, the program checks equivalence of the complete new line to the previous line (sometimes the student should type brackets around the entered sub-expression). In some rules where result has a more complex structure, only equivalence and some operation priorities are checked.

The free input mode is the most general input mode. This mode can be designed for all rules.

2.5.3.2 Second input mode: structured input

In some cases, we do not want students to work on creating the structure of the expression by replacing the operands. In the structured input mode, our program uses the information about the actual rule and operands, and predicts itself the structure of the required input using different input boxes for signs, coefficients, variables, exponents, etc. (Figure 2.5).

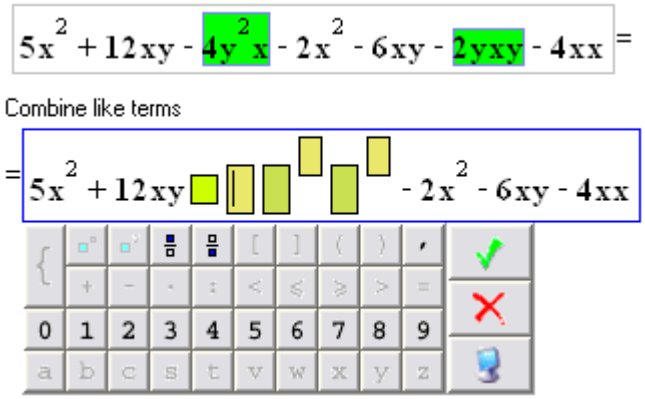


Figure 2.5. Structured input

The size and position of the boxes should make the user understand at once what he should enter. In this mode, input in the boxes is restricted. If the cursor is in an input box, the buttons with unavailable symbols on the virtual keyboard are inactive and the corresponding keys on the regular keyboard do not work. For example, the sign input box enables the user to enter only addition, subtraction, multiplication or division signs. The coefficient input box (active on Figure 2.5) accepts common fractions, numbers and comma (for decimal fractions). The variable input box accepts only letters, etc.

For example, in Figure 2.5, where the rule *Combine like terms* was selected and two like terms were marked, the program offers a pattern of monomial with six boxes on the next line. The first box is sign input box, the next is coefficient input box, followed by boxes for input of variables with exponents.

Generally, the number of boxes for the variables of one monomial offered by our program is the same as the number of variables in the marked parts. Variables can be entered in an arbitrary order inside one monomial. However, the program requests the user to standardize the result to some extent, because the number of offered boxes is limited. For example, although the form of one monomial is yxy in Figure 2.5, the program offers only two boxes for entering variables on the next line, i.e., the user must standardize the form yxy and change it to xy^2 or y^2x.

While the program offers a pattern of the result and permits certain entries in every box, it is possible to leave some boxes empty. For example, if the power of a variable is 1, the exponent can be left empty. If the power of a variable is 0, both the exponent and variable boxes may be left empty. If the coefficient box is left empty, the program interprets this as 1. If all boxes in one monomial are left empty, the program presumes that this term is combined or reduced, depending on the selected rule, etc.

If the user has finished entering then the program checks, whether the new expression is equivalent to the previous one and whether the entered parts are equivalent to the parts calculated by the computer. If they are equivalent, no further checking is required. If the expressions are not equivalent, it is possible to check the correctness of each entered part to produce a more specific diagnosis.

The structured input mode is rule-specific (each rule requires a unique input pattern of the resulting expression) and it turned out that this mode is useless for some rules. For example, it would be pointless to offer a pattern for the result if the applied rule was *Remove parentheses*, because only signs change.

2.5.3.3 Third input mode: partial input

The third mode is a simplified form of the second mode, where the program fills some boxes by itself. For example, Figure 2.6 shows the same example as Figure 2.5, but using input of some components.

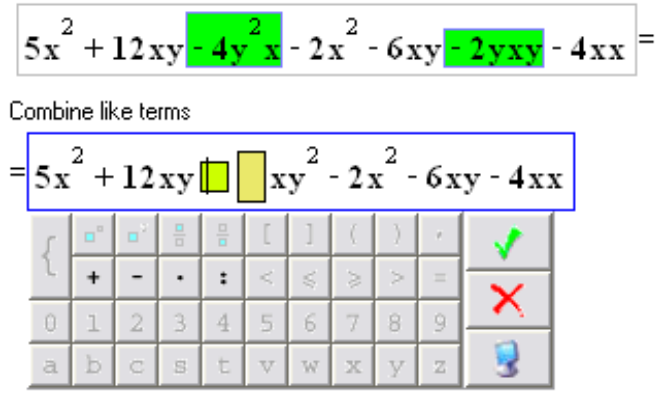


Figure 2.6. Partial input

The program itself writes the variables with exponents. The user should only enter the sign and coefficient of the monomial. The program simplifies the work of the user also by doing the standardization of the variables of monomials, i.e., converting the monomial into normal form.

After the input the program checks correctness of the expression and equivalence to the previous one as in other modes.

2.6 Extended Action-Object-Input dialogue scheme

When designing the rules for T-algebra, we found that it is difficult to realize some rules using pure Action-Object-Input dialogue (section 2.5). In order to decide, which features we need to add to the dialogue, we studied students' works on paper – how and which steps do they make while solving problems on paper. We also reviewed school textbooks to find all the rules used for making solution steps and algorithms for solving the problems. Virtually in every topic we found some rules where adequate realization required modification of the dialogue.

The easiest cases were rules that did not require selection of the object. For example, in the solution of one equation or inequality, the rule *Multiply the sides* is always applied to the whole line (however, selection is required in the case of a system of equations).

We found two different reasons for inserting additional sub-steps in the rule dialogs. The first reason is that, for some rules, the form of writing the solutions as suggested by textbooks contains more than one 'input'. For example, Estonian textbooks suggest writing addition of fractions with different denominators as follows:

$$\frac{1}{6}\Big|_{\underline{4}} + \frac{3}{8}\Big|_{\underline{3}} = \frac{4+9}{24}$$

Here the students first calculate the common denominator of the fractions being added and write it to the resulting fraction after the equality sign. Then the students find so-called extenders (the factors for multiplying the numerator and the denominator of the fractions) and write them to each addend. After that they find the members of the numerator of the result.

The second reason is that, in some rules, the result of application is not uniquely defined by operands but depends on some additional decision of the student (the choice of a common denominator in the case of addition of fractions, the values of syntactic variables of multiplication/factoring formulas). Even if this information is included in the final input, it could be very difficult to guess if the input is inconsistent. In some cases, this information is needed already for checking the intermediate input (extenders) or for building the structure for structured and partial input.

The third reason for introducing some novelty was that we did not want to predict the number of monomials in the result in some operations of the structured mode, such as multiplication of polynomials and multiplication/ factoring formulas.

For solving these problems, the dialogue of some rules was modified by adding certain special features (Lepp, 2005; Issakova et al., 2006). We extended the input stage of the dialogue by adding three new features. Each rule may use one or several of these features at once, depending on the mode running. The added features are the following:

- input of rule-specific additional information,
- input of intermediate result,
- adding terms to the result.

The following paragraphs present these added features in more detail, describing what the program does, what the student has to enter and what the program checks. We will also give some usage examples for each new feature.

2.6.1 Skipping some stages of the initial dialogue

When designing the rules, we found it unnecessary to realize all the rules using all three stages of the dialogue. Therefore, we added the possibility of skipping the selection of objects or input of the result in some rules.

Some rules are applicable to the whole expression (multiply both sides of equation, compare the fractions, etc.), which is why it would be pointless to ask the user to select the whole expression. Another set of rules is designed specifically for single step problems and these rules are not used elsewhere (define order of operations, find reciprocal value, compare the fractions, calculate approximate value of the fraction). These rules are also applicable to the whole expression; therefore, the stage of object selection is skipped. After the rule has been selected, the program checks whether the rule is applicable and then applies the rule, letting the student to fill in the boxes in the result.

The final input stage can also be skipped in some rules. For example, using the rule for reducing the fraction, the student enters the result of the application of the rule in the intermediate results stage (in a different form, reduction results are entered to every part of the fraction) and the program rewrites it automatically.

2.6.2 Input of the rule-specific additional information

Application of some rules requires certain specific information, which does not belong to the resulting expression or is difficult to extract from the result in case of an error. For example, when adding fractions with different denominators, the common denominator value should be entered. Even though it is a part of the resulting expression, it should be entered before the input editor for the resulting expression is created, because this information is needed while checking the correctness of extenders. As we still want to check the student's skills and identify the cause of errors, this specific information has to be entered separately. When creating the expression on the next line, the program uses the selected rule, objects and information entered.

For each such rule that needs additional information, a separate input window was created containing rule-specific input boxes. For example, when adding fractions, the student has to input only one number – common denominator of the fractions. However, this window can also have a more complex structure, for example, in the case of the rule Factor out common factor, the program prompts the user to enter the common monomial to factor. Similar input was used in the Mathpert system (Beeson, 1998).

This added window is the first new feature that can be followed by other options or the usual input of the resulting expression. When the objects of the rule have been selected, the program checks whether the rule is applicable to them and after that displays this input window to the user. After the student has made the input in this window, the program checks whether the entered information is correct. If no errors were diagnosed, the student may proceed to the next stage. Our example shows the additional input window (Figure 2.7) for the rule: addition of fractions with different denominators. In this window the program asks the user to enter a common denominator for all selected fractions. Figure 2.7 shows the initial expression in which the objects (fractions to be added) have been marked, and an additional window for entering the common denominator. The information entered in this window is used not only for additional knowledge testing but also for construction of the resulting expression – the resulting fraction already has the denominator filled in.

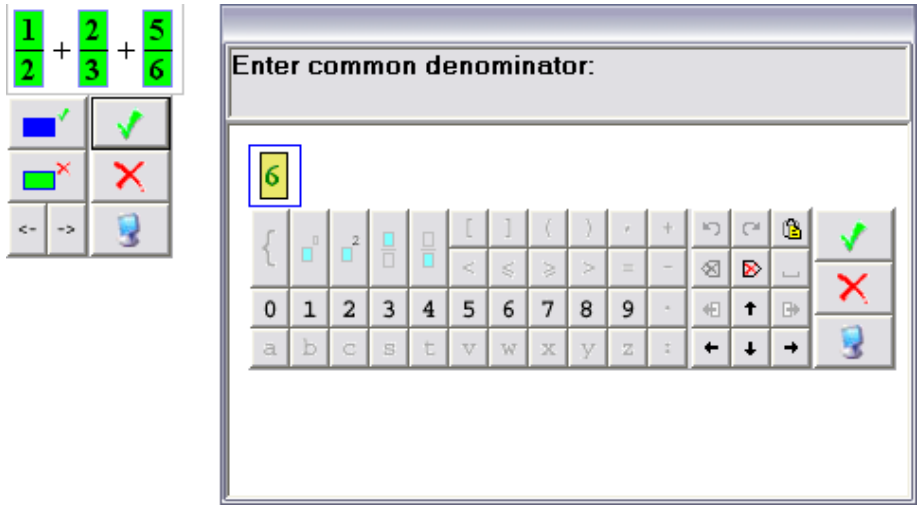


Figure 2.7. Input of the rule-specific additional information in the case of addition of fractions with different denominators

In the field of polynomials, this additional input window is used, for example, when asking the student to enter the common factor in the rule *Factor out common factor* (see details in section 3.2.18).

2.6.3 Input of intermediate result

Looking through the solutions on paper, we found that some rules are applied using two input stages: first some intermediate result is found (for example, extenders for each term are found when multiplying both sides of equation) and then the final result is written. We tried to follow the same pattern to extend the dialogue that is used when working with pen and paper: at first, the common denominator is entered in an additional window, then the extenders of the fractions are entered and, after that, the members of the final result are entered. As we wanted to keep the initial expression unchanged with the objects selected in it, after entering the common denominator, the program copies the expression to a new line and provides boxes for entering extenders. The same constraints are used here as in structured or partial input – the boxes allow entering only limited expressions (only numbers, etc.). Figure 2.8 displays the extenders input boxes required by the rule of addition of fractions.

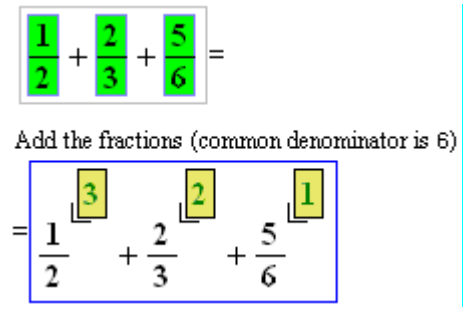


Figure 2.8. Input of intermediate result when adding fractions with different denominators

After the intermediate result has been entered, the program checks the correctness of entered parts. In case of an error, the student is given an appropriate message and the program lets the student to correct the result before proceeding. If no error is diagnosed then the program constructs the result of applying the rule based on all the information entered and lets the student to enter some parts of the result, depending on the solution mode running.

2.6.4 Adding terms to the result

Most rules that are used for making transformations to algebraic expressions actually shorten the initial expression: the number of terms decreases, two or more terms are joined somehow, etc. However, rules dealing with polynomial multiplication lead to a growth of expressions and the structure of added terms differs from the structure of the terms that caused this growth. In the free input mode, the student has to build the structure of the result himself. In the structured input mode, described above, we would give the student too many hints on how the result should be found, e.g., the number of terms in the result, etc. We have found a better solution.

The members of the resulting sum have the same general structure. Instead of drawing the boxes for all terms, we can draw a box for the first term and give the possibility to add more terms dynamically by adding or removing monomial structures. When checking the result, the program checks whether an appropriate number of terms was added and it also checks each term separately.

Figure 2.9 shows an example of adding terms to the structure of the result in the rule of multiplying two polynomials. The result of application of this rule is also a polynomial that the student has to construct of monomials. At first, one monomial structure is given (Figure 2.9 on the left). Then the user can extend the structure by pressing the appropriate button on the virtual keyboard and the program adds one more monomial (Figure 2.9 on the right shows the added monomial, input boxes are filled with parts of the result). This mode requires exact application of this rule only; combining similar terms is not allowed.

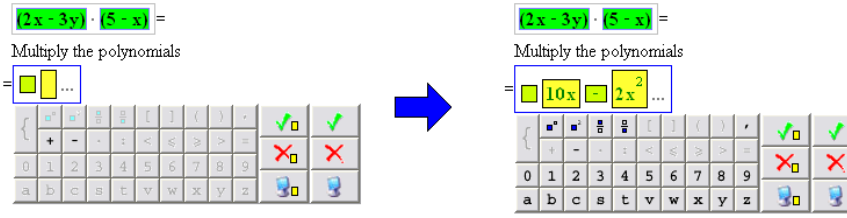


Figure 2.9. Adding terms to the result in the rule of multiplying two polynomials

2.6.5 Extended dialogue scheme

The following is a summary of the new extended scheme, including all the checks performed by the program. If an error is found then the program does not permit proceeding to the next stage of the solution step.

1. The student selects a transformation rule from the menu – the program allows selecting any rule without checking, whether it is possible to apply such transformation at this stage.
2. The student selects a set of objects in the expression (may be not used in case of some rules) – the program checks, whether the rule is applicable to the selected objects.
3. The student enters rule-specific additional information in a separate window (may be not used in case of some rules) – the program checks the correctness of entered information.
4. The student enters an intermediate result of application of the rule (may be not used in case of some rules) – the program checks the correctness of entered information and the correctness of rule application.
5. Before the final result is entered, the student can add structures of terms to the result to achieve a fitting number of terms (may be not used in case of some rules) – the program does not check, whether the number of terms was exact. The important question here is, whether the answer will fit in. Terms can also be added in the next stage if the result does not fit.
6. The student inputs some part of the resulting expression, depending on the problem solving mode (may be not used in case of some rules, the program rewrites the intermediate result automatically) – the program checks, whether the rule is applied correctly.

The described scheme or some subset of these stages should be suitable for any rule in T-algebra.

2.7 Expression editor features to support Action-Object-Input solution step scheme

There are different ways to present and enter expressions in different computer algebra programs and learning environments. For example, older versions of Maple (Maple by Maplesoft) use command line input (linear input for 2D expressions) but present expressions naturally in a 2D form. Some other systems like MathCAD (MathCAD by PTC) provide WYSIWYG 2D editors: both input and representation are 2D. Different ITS also utilize both mentioned ways for inputting expressions, some of them even use expression representation as a tree and allow selecting nodes for different operations, for example, EXPRESSIONS (Thompson and Thompson, 1987).

We designed our own WYSIWYG 2D expression editor for T-algebra (Lepp et al., 2005) to support, in particular, the second and third stages of the step (selection of objects to apply selected rule to, and input of essential parts of the result). We wanted to achieve the following goals with the editor:

- The student has to perform as few operations as possible in one solution step (compared to pen and pencil solutions, where he has to write all unchanged parts of the expression at each step – T-algebra copies unchanged parts and protects those from modifying).
- During solving the student has to have a possibility to make mistakes (in selecting the rule and in applying it) and the program should be able to help the student to correct errors.

The last goal is also the reason why we had to design an editor that does not guarantee correct structure of the expression. The designed 2D editor is easier and clearer to use for school students than command line input. In addition to classical operations, the created editor allows some rule dialogue specific operations – rule objects selection, constrained input of the result. These operations enable to simplify student work when making the solution step, force the student to enter parts of the result that are essential in applying the selected rule, and identify the correct objects to apply the rule to.

2.7.1 Different representations of expressions

In the T-algebra expression editor, there are three different representations of expressions: natural 2D representation, linear string (1D) representation and tree representation (object representation). Different expression representations are discussed by Nicaud and Bouhineau (Nicaud et al., 2008). The natural 2D representation is used to display expressions to the user in the editor; most people are used to seeing algebraic expressions in this form. This 2D representation is used in all textbooks. The other two are used internally by the program. In the expression input mode, the program stores them as strings (1D representation). String representation is also used when saving problems and solutions to a file. When manipulating expressions, applying rules, selecting

objects, etc., a tree-like object structure is used. Let us review both these inner representations.

2.7.1.1 Inner string representation

If we check the expressions that are supported by T-algebra (see section 2.4) then we can see that most of allowed operations and sub-expressions have linear screen representations with some exceptions (fraction, exponent, etc.). When storing expressions as strings we define codes for storing such non-linear sub-expressions, as for example a^2 is being stored as string „a^{2}". The program keeps the linear parts of the expression unchanged ($a+bc$ is stored as „a+bc"). When the user modifies the expression, the inner string representation is changed accordingly. All changes are immediately reflected in 2D representation. A question can arise at this point: Why was XML or XML-based MathML, OpenMath or even TeX syntax not used? The main reason is that we wanted to store linear sub-expressions almost unchanged (small changes, note that Estonia uses comma as decimal separator but we still used point; schools use colon as division sign but we still used slash as division mark) and keep the structural sub-expressions using shortest possible codes, so that when modifying an expression, all editor keystrokes go directly to the inner string. Some supported elements are presented in Table 2.1 along with their code in the inner string representation.

Table 2.1. Supported elements with their codes

Expression	String	Expression	String	Expression	String
$2{,}2+x^2-y$	2.2+x^{2}-y	$1\frac{2}{3}$	1{2/3}	$(-1)+[-b]$	(-1)+[-b]
$2x\cdot y:2z$	2x*y/2z	$2x-3=x+2$	2x-3=x+2	$\begin{cases} x-y=3 \\ 2x+y=5 \end{cases}$	x-y=3& 2x+y=5

2.7.1.2 Inner tree structure

When we have the correct expression then the program can parse the inner string representation and build up a tree representation. The program does some further operations with this tree – it checks the possibility to apply the rule, correctness of the selection of rule objects, and rule application. After that, the resulting tree is transformed back to string representation to enable the student to input the resulting parts.

The tree stores operations of the same priority and arguments of these operations on the same level. Atomic expressions can be found at tree leaves – variables, numbers and operation signs. The inner vertexes of the tree include several kinds of objects that define the priority of the next level operations:

1. System of equations, which has equations as its children.

2. Equation and inequality – connecting mark and both sides are the children.
3. Sum-difference objects – children on the next level are sub-expressions connected by plus and minus signs.
4. Product-quotient objects – has multiplication (also with omitted multiplication sign) or division members as children.
5. Power object has 2 children: argument and exponent.
6. Bracket object includes the bracket kind and sub-expression as its only child.
7. Fraction objects contain 2 children: numerator and denominator.
8. Mixed number also contains 2 children: integer part and fraction part.

Such grouping of operations by priorities simplifies program checks and application of rules. For example, if we need to check that monomials, selected by a student for combining, are suitable we check that they have the same parent node in the tree (members of the same sum), we check that the parent node is a sum-difference object (that gives that monomials are not members of product) and we check separately that variable parts are equal. Figure 2.10 shows tree representations for the expression $5a+b-3a+14b:2a$. Here the sum-difference object is marked as "+-" and the product-quotient object is marked as "*/".

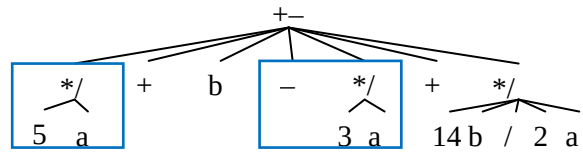


Figure 2.10. A tree representation of expression

When the student selects $5a$ and $3a$ as objects then the program selects both roots of these sub-trees and also the preceding sign before the monomial $3a$ regardless of whether the sign was selected or not in 2D representation (Figure 2.10). When applying the combining rule to selected objects, the selected objects are removed from the tree, and new monomial $2a$ is created and inserted into the tree to the position of the first selected object. The tree that the program gets after applying the combine rule is shown in Figure 2.11. From the resulting tree, a new string representation is formed, where some places are left for student input "??". For example, the resulting expression in the partial input mode would be "??a+b+14b/2a".

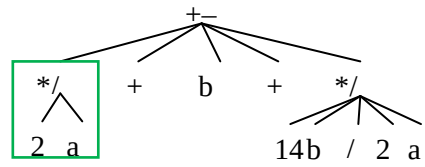


Figure 2.11. Tree after rule application

2.7.2 Expression parser

As we use two different representations, we needed a tool to transform those from one to another. It is quite trivial to transform object representation to string representation. Each object "knows" how to present itself as a string. In case of tree leaves it is trivial (single number, sign or variable). In case of inner vertexes of a tree it calls this transformation method to all its children and combines the result to one string. To get string representation for the whole expression we simply call this "to string" method from the root of expression object.

A bit more complex transformation is from string to objects. We have implemented a special expression parser for this case. First of all, object representation is always correct, so objects always define the correct string representation. However, the string representation can be syntactically incorrect (for example, a missing bracket, two operators in a row, etc.). Therefore, the parser has a possibility to detect those errors. This feature is used in different rules to detect student errors in entering the resulting expression. In many cases (especially in the free input mode) students are able to enter an arbitrary expression in input boxes, but this expression should be of course mathematically correct. For checking correctness in rules, we first try to parse those entered parts and get an error if the entered sub-expressions are syntactically incorrect.

Another similar parser is used in the expression editor itself for expression rendering. The rendering parser has many differences, for example, it should be able to parse and then to render also incorrect expressions that may appear while entering (for example, a missing bracket, empty denominator in fraction, etc.).

2.7.3 Expression editor

Let us take a closer look at the 2D expression editor of T-algebra. By 2D editor we mean a system, which displays expressions in natural representation and enables the user to modify them. This means that the user can move the cursor inside the expression and make changes to the expression. The string that is actually being modified when the user modifies the expression is the inner string representation and the program displays a new expression to the user. We

wish to consider the classical actions of an editor: place the cursor, move the cursor, select part of an expression, input or delete at the cursor position or over a selection, copy or cut a selection, and paste at the cursor position or over a selection.

There are three modes in which the editor can interact. The first, free input, is meant for composing problems. In this mode, no constraints apply to the entered expression. The second mode, selection of objects, is meant for selecting the objects for applying the rule. The expression in this mode is read only. The third mode, constrained input, is meant for input of the essential parts of the result at each solution step. The student can modify the expression only in specific places defined by the structure of the result.

If we used the terminology from the APLUSIX articles (Nicaud et al., 2004) and (Nicaud et al., 2008), we could say that T-algebra editor also uses the so-called *text&box* input mode. In *text&box* mode, expressions are seen as strings of characters or boxes – expression is a string in general with some boxes in it. These boxes contain sub-expressions (strings) and could contain other boxes (fraction, exponent, etc.) as well. A similar mode is also used in some other editors, for example, MathType.

2.7.3.1 Expression correctness in editor

The editor should be able to deal with both correct and incorrect (incomplete) expressions. It is clear that different expressions, including incorrect ones, can be entered in the entering mode. The editor displays the expression based on the inner string representation. The editor identifies the codes for different structures (box operators) such as fractions, exponents, etc., and displays them correctly.

After the input phase, the program checks whether the expression is correct and satisfies all the constraints. This is done by parsing the expression into tree. The question arises, why we have chosen the *text&box* mode for our editor? The goal of T-algebra is teaching different aspects of problem solving. The student should have a possibility to make mistakes in entering results as well as in selecting the rule and its objects, which would be impossible in structured mode (as in APLUSIX, for example) where the editor guarantees correctness of the structure.

2.7.3.2 Classic operations in the editor

Classic operations were listed in section 2.7.3; let us take a closer look at them.

In *text&box* mode, there are two possibilities of entering the expression. The first one is to add a symbol to the cursor position and the second is to add a structure for entering a nonlinear object to the cursor position. The buttons on the virtual keyboard or keyboard shortcuts can be used to enter these so-called box-operators. There are also some keyboard buttons duplicated on the virtual keyboard – brackets, arithmetical operations, numbers, etc. The virtual keyboard and the editor are shown in Figure 2.12.

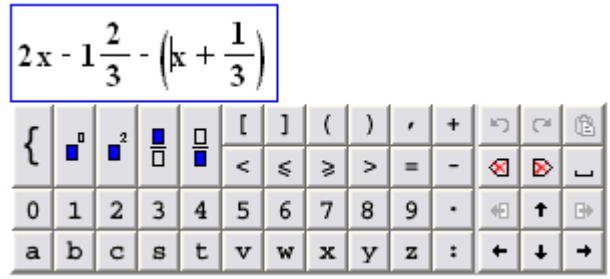


Figure 2.12. Expression editor, virtual keyboard

Unlike other system editors, the T-algebra editor does not show empty boxes in structures for entering fractions, exponents, etc.

In *text&box* mode, an arbitrary string or box can be selected. Selection is done similarly to all text editors. Either the keyboard or mouse can be used. In case of box-operators, such as a fraction, the user can select a part or the whole denominator, numerator or the whole fraction. If the user starts selecting from the denominator and drags the mouse to the numerator then the whole fraction is selected. If selection is started outside the fraction box and ends inside then the program adds the whole fraction to the selection. Such selection can be used for copying expressions, etc.

The next classic operation is input over a selection. Usually in *text&box* mode, when a part of expression is selected and the user executes input, the selected part is replaced by the entered one. In T-algebra, input over a selection proceeds the usual way, but we have added some features to simplify user work.

- When the user presses the exponentiation button on the virtual keyboard, the selected part is put into brackets and the corresponding exponent (two or an empty box) is added. The cursor goes to the exponent entry box.
- When the user selects one of the fraction buttons, the fraction structure is created in place of the selection and the selected part goes to denominator or numerator, depending on the selected fraction button.

There are different kinds of deletion: to the left of the cursor, to the right of the cursor, and deleting the selection. When a part of expression is selected and any delete action is executed (backspace, delete, buttons on virtual keyboard), only the selected part is deleted. There are two possibilities of deleting when nothing is selected. If there is a linear symbol at the cursor, then this symbol is deleted. If there is a box-structure, the program selects that structure and deletes it after the user executes the delete operation (presses button on the keyboard or the virtual keyboard) for the second time.

The final classic operations in the editor are copy, cut and paste. These work in the usual way. Copy and cut move the selected part of the expression (actually, part of the inner string representation) to the clipboard and cut also executes the delete operation. As we wanted to use the Windows clipboard to

leave the student the possibility to copy parts from other expressions, we had to modify the paste operation, so it checks whether symbols being pasted are of correct syntax and all box-operators are complete. The paste command is disabled in case of unsuitable subexpressions on the clipboard.

2.7.4 Advanced features of the editor

As was mentioned above, the T-algebra expression editor offers some additional features, which are used by the program for problem solving.

2.7.4.1 Selection of arguments (multiple select)

Every solution step in T-algebra is application of the selected rule to the expression. After the rule is selected, the user has to select objects to apply the rule to. The editor in the selection mode is shown in Figure 2.13.

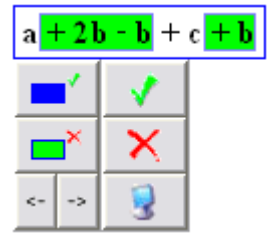


Figure 2.13. Editor in selection mode

As the selected scheme for the rule dialogue is Action-Object-Input scheme, the user has to select objects of the rule at the second stage. Selection of the rule objects is slightly different from ordinary selection of a part of expression. Firstly, selected objects can be located far from one another in the expression. Secondly, the student has to have the possibility to select and deselect objects, confirm his selection and let the program check it. The corresponding buttons are added to editor in this mode:  adds selected part to the objects of the rule;  removes selected object from selection. The student can move between selected objects using the following buttons  .

Such *multiple-selection* is rare in CAS (the Mathpert system is probably the only example), but is widely used in other programs: word processors (MS Word, etc.), table processors (MS Excel, etc.), Windows Explorer, etc. Different selection modes are used in the listed systems, but we preferred a slightly different selection mode for T-algebra. There are several reasons for that. In the listed programs, there are no restrictions on selections. However, if we are dealing with expressions and select objects for application of a rule, it is clear

that the selected objects should be correct sub-expressions. So the program should check for that when adding to the selection. The other reason is that we need two different types of selection in one editor (selecting objects for applying the rule and selecting a part of expression for copying to the resulting expression), so we used two different colours for these selections.

For user convenience, objects located close to each other can be selected as one object. For example, in expression $a+2b-b+c+b$, when user selects the Combine similar terms rule and selects objects containing the variable b, he can select $2b$ and $-b$ as one object and $+b$ as another. During selection of rule arguments, the expression is presented as a tree and all selected objects are selected in the tree. When the user adds a selection to the rule arguments, the program actually selects all sub-trees covered by the selection. Thus, if the user selects $+2b-b$ then the program marks the following sub-trees $+, 2b, -, b$. Let us see the example in Figure 2.14 (the tree and objects correspond to the expression and selections in Figure 2.13) – user-selected parts are shown in blue boxes and the selected sub trees are shown in smaller lilac boxes. If the user would select all three objects as different parts, the selected sub-trees would be the same.

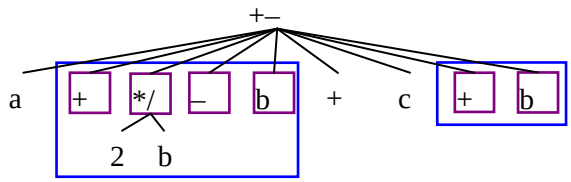


Figure 2.14. Tree showing selected objects

Another simplification made for user convenience is that the user does not have to select the operation sign to the left of every object. When applying the rule, the program considers that these signs were selected as well. In the same example, when the user selects $+b$ without selecting the plus sign, the program still selects it automatically in the inner tree.

When selecting objects, T-algebra checks that selection is syntactically correct, for example, it is not possible to select only part of a number as an object.

2.7.4.2 Constrained input

After the rule arguments have been selected and the selection has been confirmed, the program first checks whether selected parts are suitable for application of the selected rule and then applies the rule. The program leaves some essential parts of the resulting expression blank to be entered by the student. The parts that are left for the student to enter depend on the selected

rule and input mode. Most rules in the free input mode require the student to input the whole part of the result in one piece with no constraints on the sub-expression to be entered. In other input modes, a structure of the result is built and the student is given the opportunity to fill the gaps (structured input mode). Figure 2.15 shows the editor in this mode. All input is disabled outside the input boxes, protecting the unchanged parts of the expression.

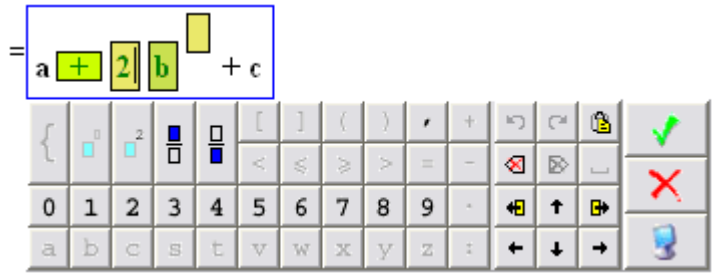


Figure 2.15. Editor in constrained input mode

In the input mode for parts of the resulting expression, the program offers one or more boxes for entering the parts. Other parts of the expression are copied from the previous step and protected by the program from modifying. There are constraints on the kinds of expression parts that can be entered in different boxes. The colours, sizes and locations of the input boxes may vary depending on these constraints. Different boxes allow entering only variables, numbers, operator signs, equality and inequality signs or arbitrary expressions. For example, in Figure 2.15, the cursor is in the coefficient box and the virtual keyboard shows what possible symbols can be entered in that box.

2.8 Applications of domain expert module

When designing and implementing T-algebra, we defined a set of issues or key attributes we wanted to achieve. T-algebra has to

- allow the student to make all the necessary decisions and calculations at each solution step;
- contain such dialogue that allows the program to understand all decisions made by students (chosen operation, selected operands, entered result);
- enable to solve problems step-by-step and line-by-line as on paper;
- give the possibility to exercise both the algorithms and their steps in detail;
- leave an opportunity for the student to make the same mistakes as on paper;
- automatically calculate answer to a problem;

- generate and show a solution path to a problem using the same set of rules that is used at school in paper solutions;
- check students' solutions and diagnose errors;
- offer feedback to the student;
- provide advice on request how to proceed with solution.

Out of this list, the first five options are available by design of solution step dialogue, expression editor and defined set of rules. The last five options are available as a result of the applications of the domain expert module of T-algebra. This section presents these options and other main applications of the domain expert module in detail.

2.8.1 Checking equivalence of two expressions

Checking equivalence of two expressions is the most common application of the domain expert module in all intelligent problem solving environments dealing with expressions. If we would not need other applications we could even consider using a computer algebra system engine for this purpose (Ravaglia et al., 1998; Sangwin, 2005). However, as we had implemented a domain expert for other needs, we used our own checking engine for this, as it was easy to do for supported expressions.

Checking of equivalence is the most used feature of the domain expert module in T-algebra. This is used for different purposes in the implementation of almost all rules and problem types, mostly for comparing student input with the automatic result of T-algebra in the free input mode.

The checking algorithm itself is quite simple and straightforward. It uses the same transformation rules and simplification / solution algorithm as all problem types (described in section 3.3). A fixed set of transformation rules (23 rules) is used in this algorithm. For checking equivalence of expressions A and B, T-algebra creates another expression $A-(B)$ and tries to simplify it. If it simplifies to 0 then expressions are counted as equivalent.

The composed checking algorithm might not be an optimal one – not the fastest and it may be using resources, but it was not a big problem. In addition, this algorithm may not work for every possible algebraic expression. However, it is suitable for the expressions appearing in the limited set of problem types that we have designed for T-algebra as, for example, T-algebra does not work with trigonometry and absolute value.

This algorithm can be used for checking equivalence of linear expressions. A different approach is used, if needed, for more sophisticated cases, for example, checking of equivalence of equations or fractions with variables.

In case of equations, for example, the left and right parts are compared separately (this is good enough for checks required in the implementation of the rules, but not for general equivalence of equations). Systems of equations can be handled the same way; different equations are checked separately.

In case of fractional expressions with numbers only, it is easy to compare values of the expressions. When variables are introduced then there are only some rules that support operations with fractions with variables and special separate problem types defined for practicing those, for example, the rule *Raise quotient to a power*. In the free input mode, the student enters the resulting fraction himself. As by design we usually do not allow other simplifications, as the rule name says, we could check equivalence of student input to the result of T-algebra by checking separately the numerator and denominator part of the fraction using the same algorithm (otherwise if, for example, the student could reduce the fraction at the same time, we could not use the same algorithm).

When talking about equivalence here and later in the context of T-algebra, we mean equivalence as it is taught and used in school. For example, in school expressions $\left(x^{-1}\right)^{-1}$ and x are considered to be equivalent. However, according to the definition of equivalence of algebraic expressions, the values of two equivalent expressions should be same for every set of variables. In this case the value of the first expression is undefined if $x = 0$ and the value of the second expression is 0 so those are not equivalent by definition.

Another variation of equivalence checking is to see whether expressions are opposite to each other (for expressions A and B it means that $A+(B)$ will simplify to 0). This is used, for example, in the rule $(a+b)*(a-b)$ to check selection of objects – whether the selected polynomials contain the same monomial with different signs. These monomials can be in a different form, therefore, some simplification is needed when checking it, for example, $\frac{1}{2}xyx$ and $-0{,}5x^2y$.

2.8.2 Checking the initial expression of a problem

The function of checking the initial expression of a problem is used in two cases: when entering new problems in the teacher's program and when generating a random expression for a problem (we need to be sure that the generated problem satisfies exactly the same conditions as the entered one) in the student's program. A problem type should be selected when composing problems – some checks that are performed are specific to problem type.

This checking process itself contains several smaller checks:

- syntactical correctness of expression – this is checked using expression parser (2.7.2), incorrect expressions generate error on parsing;
- expression form, defined by the problem type, for example, equation, polynomial, fraction, etc.;
- problem type specific checks (described in subsections of section 3.3), for example, for the problem type *Combine like terms*, the initial expression should contain at least one pair of like terms;

- problem should be solvable by the rules and algorithm defined for this problem type (this is checked by the solution algorithm of the problem type).

2.8.3 Calculation of result

T-algebra is able to apply the solution algorithm and calculate the answer to a problem (without showing / generating a solution path). This is used in the teacher's program. While entering new expression for a problem, the teacher can check whether the problem is solvable and see the answer by pressing either the *Show answer* or *Show solution process* button. The teacher can thus decide whether the problem is suitable for students (for example, the solution of an equation is integer / decimal, the resulting polynomial has only integer coefficients, etc.). An example of the problem composing window with the result of a problem is shown in Figure 2.16. You can see both the initial expression of the problem and the calculated result in the bottom right part of the window.

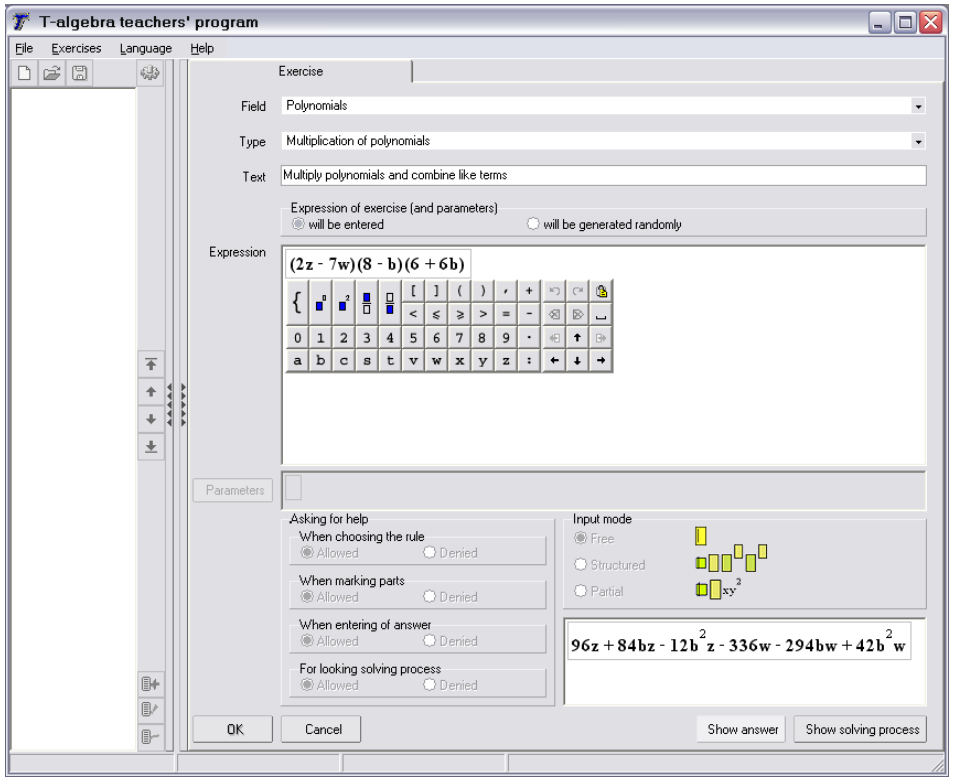


Figure 2.16. Example of automatic calculation of result during problem composing

2.8.4 Automatic solution generation

In addition to automatic calculation of the result of a problem, T-algebra is able to generate step-by-step solutions to problems. The solution is generated using the same transformation rules as may be used by students (from the set of transformation rules for the selected problem type) and the algorithms described in schoolbooks are followed as closely as possible (defined by the solution algorithm of the problem type).

Automatic solution generation is used in two cases in T-algebra. While solving problems in the student's program, the user can ask the system to solve a problem until the end. The teacher can disable this possibility in the problem file. If this feature is enabled then the student can press the *Autosolve* button at any time when solving a problem. See an example of an automatically generated solution in Figure 2.17. T-algebra is able to generate an automatic solution starting from the expression in the last row of the solution path, meaning that the student can make some steps himself and then ask T-algebra to complete the solution. Each usage of this automatic solution in the student's program is logged and saved to the solution file (see details in section 2.9).

Another way to use this feature is in the teacher's program when composing problems. It is possible to ask the program to generate the solution for the problem being entered (for the current expression). The teacher can estimate the solution path and decide whether the entered problem expression is suitable for his students (it is similar to automatic calculation of the result, but here it is also possible to assess all the intermediate steps).

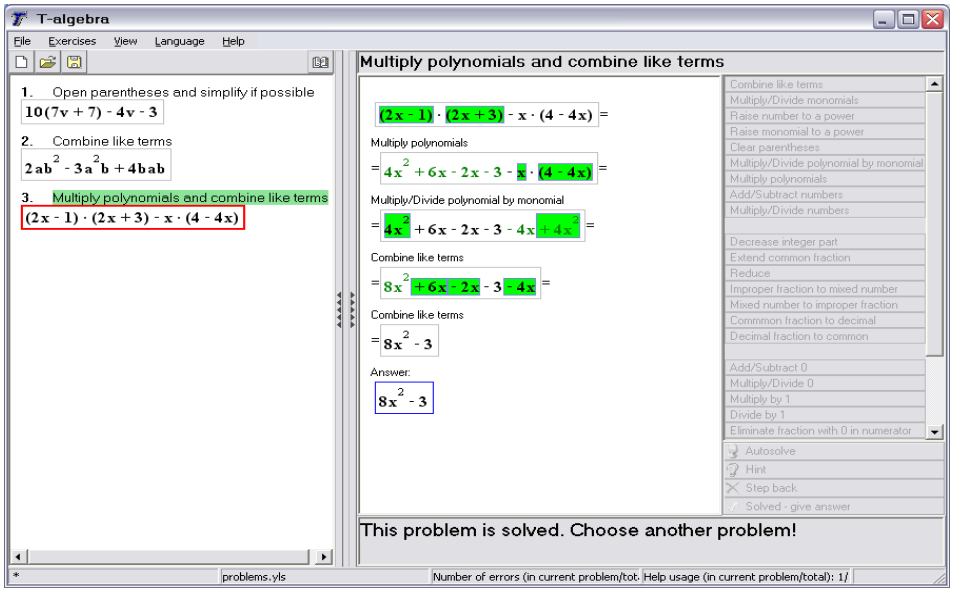


Figure 2.17. Example of automatically generated solution in the student's program

2.8.5 Advice on request

In addition to automatic solution generation, T-algebra offers local kinds of hints or help to students. Students are able to do the following:

- ask which rule to perform next,
- ask the program to select suitable objects for the chosen rule,
- ask T-algebra to add a correct number of input boxes in the structure extending mode (see details in section 2.6.4),
- ask the program to fill in input boxes in different input stages of the step (including input in a separate window).

These features can also be individually disabled in the teacher's program when composing a problem file. Let us review those possibilities one by one.

For selection of a rule, the student is able to ask program which rule should be applied by pushing the *Hint* button. The expert module will choose a suitable rule according to the solution algorithm for the problem type and the current expression and program will display an appropriate message to the user (Figure 2.18). The same rule would be used if automatic solution generation was selected. If the problem is actually solved, T-algebra will display a different message saying that the student has to give answer to a problem.

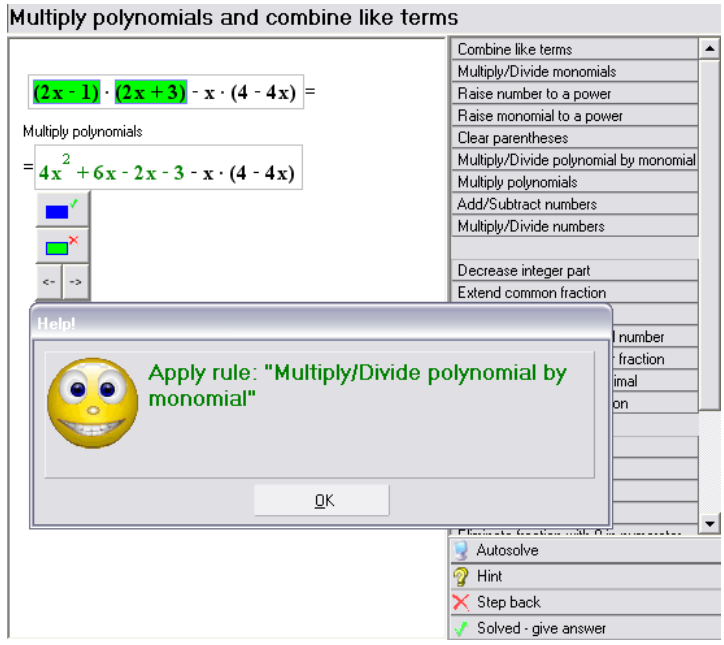


Figure 2.18. Example of a hint for choosing a rule

After the rule is selected, some more buttons become visible in the expression editor, including the hint button (Figure 2.19, left side). The student is able to ask the program to select suitable objects for this rule by pressing this button. As a result, T-algebra selects objects for the active rule or shows the student a message if this rule cannot be applied to the current expression. Objects suggested in this hint are dependent on the rule and the problem type. If the student himself has selected some objects in the expression (either right or wrong) those are ignored (unselected). Figure 2.19 shows an example of using this hint: the initial expression is on the left and the expression with selected objects for the rule *combine like terms* is on the right.

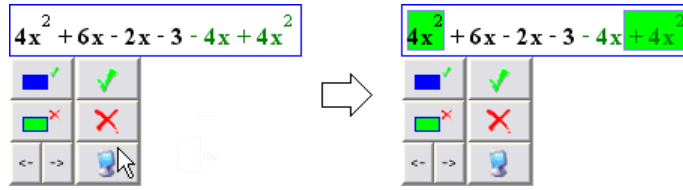


Figure 2.19. Example of a hint for choosing objects

During the input stage (also during the input of additional info in a separate window) it is possible to ask the program to fill in input boxes with correct values by pressing the hint button in the expression editor. T-algebra calculates the correct result of application of the rule to the selected objects and writes it in the input boxes. If the student has already entered some results in some boxes (either right or wrong), those are overwritten by the parts of the result calculated automatically. Figure 2.20 shows an example of using this hint: the initial expression with all input boxes empty is on the left and the expression with input boxes filled with correct parts of the result is on the right.

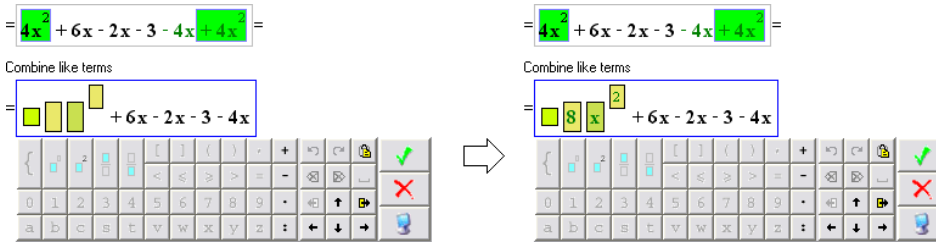


Figure 2.20. Example of a hint for input stage

In case of extending the structure in the input stage, it is possible to ask the program to add a correct number of input boxes by pressing the hint button in the expression editor. T-algebra does not change the input of the student (except if it reduces the number of boxes to a correct one). Figure 2.21 shows an example of using this hint: the initial expression (with one set of boxes) is on the left and the expression with correct number of input boxes is on the right.

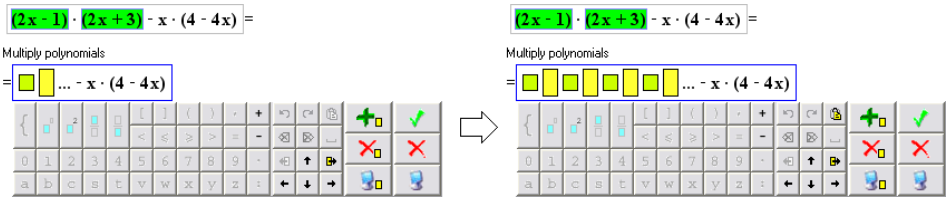


Figure 2.21. Example of a hint for extending the structure

We can compare the hint features of T-algebra with the features of another rule-based system, Mathpert (Beeson, 1998). It has three different hint features: autofinish, autostep and hint. Autofinish is similar to automatic solution generation in T-algebra, hint is similar to rule selection hint in T-algebra. Autostep automatically makes a next step in the solution; in T-algebra we have focused on different stages of a step (rule objects and result), therefore, we have separate hints for each stage.

2.8.6 Student error diagnosis

When solving on paper, a student can make a mistake and continue solving the problem. This leads to a wrong answer and all the steps that the student made after the error might not be checked by the teacher. So the student does not get any feedback on whether these steps were correct or not. When designing the T-algebra solution dialogue, we tried to avoid such situations. Therefore, the main principle of problem solving in T-algebra is that as soon as an error is found the student cannot proceed to further stages of the step or to the next steps before the error is corrected.

As each solution step in T-algebra consists of three stages and the student has to confirm his selection made during the stage (confirm the rule and selected objects, confirm the entered resulting expression or its parts in partial input mode), the program can check the correctness of each stage of the step. Different checks are performed depending on the type of the problem, input mode and the rule selected – we will review all of them in the following parts, grouped by the stages of a step.

2.8.6.1 Error diagnosis after selecting a transformation rule

At the beginning of each solution step, the student has to select the transformation rule he is going to apply. In T-algebra we tried to implement all the different rules that are taught at school and used when solving similar problems on paper. The rules were given the same names as in textbooks.

As we did not want to restrict the students' solutions and wanted give them freedom in choosing the solution algorithm, the program does not check separately whether the selected rule is suitable for applying to the current expression. If the rule is not applicable to the current expression, no error message is displayed until the student confirms his selection of objects – he is given a chance to realize that no suitable objects can be found and correct the choice of the rule. After objects are selected and confirmed, T-algebra first checks if the rule is applicable to at least some objects in the expression. If not, an error message is displayed. If the selected rule is applicable to some objects of the expression, T-algebra proceeds with further checks.

The only check performed immediately upon selecting the rule is whether the current expression is already in the form of the answer to the problem (in case of simplification problems, a monomial or a polynomial with no similar terms). In this case the student has either difficulties recognizing the right form of the answer to the problem or he does not know any algorithms for solving the given problem.

2.8.6.2 Error diagnosis after selecting the operands

In T-algebra we let the student make all the decisions on how to solve the problem. After selection of the transformation rule, the student has to select the objects to which the rule is applied. Many errors arise at this stage. The approach in T-algebra is different from the one in the Mathpert system (Beeson, 1998), where the student selects parts of the expression and the program offers rules that are suitable for the selected parts. The approach used in Mathpert does not allow students to make mistakes at all, while T-algebra lets students to make many mistakes, warns about them and requires them to be corrected.

When selecting separate objects, the program checks whether the selected parts are syntactically correct sub-expressions, and displays an error message if they are not (half of a number is selected, a variable is selected without its power, one of the brackets is not selected, etc.). After confirming the selected objects, some extra checks are performed.

The first significant issue that the program is able to diagnose is whether the student knows and considers the priorities of the operations: for example, tries to add two numbers and one number is a member of a multiplication term.

The second issue that the program is able to check is whether the objects are of the correct structure and their number is correct – for example, the rule *Combine like terms* requires selection of at least two similar monomials. After the rule is selected, the program tells the user what kinds of objects are required by the selected rule – so the program checks whether the student knows and recognizes suitable objects in the expression.

If an error is found then a corresponding error message is displayed to the student. If possible, the program also indicates which object was causing the error (Figure 2.22). The student has to correct all errors before proceeding to the next stage. In this way he does not have to make unnecessary steps after the error, as it would be on paper. In addition, in case of unsuitable objects, the program would be unable to offer a structure for the expression on the next line.

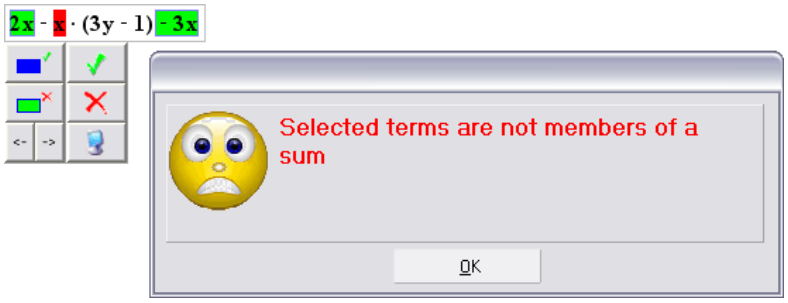


Figure 2.22. Error in selecting objects for the rule Combine like terms

2.8.6.3 Error diagnosis after entering a resulting expression

When applying the rules, the program copies the unchanged parts of the expression to a new line and protects them from modification. It only lets the user to enter the result of applying the rule – the user has to fill in yellow boxes with the result or with essential parts of the result. The number and types of the boxes depend on the rule and input mode (in addition to objects), and so the checks, possible errors and diagnoses that the program is able to perform depend on the rule and input mode.

When the user confirms his input, the program has full information on the selected transformation rule, the objects as well as the result of the application of the rule offered by the student. This gives the program a possibility for better diagnosis of the exact error and its cause. In most cases, the issue whether the entered expression is equivalent to the previous one is not the only aspect that can be clarified. Having full information on the rule and objects, the program is able to apply the rule itself and compare the student's result with the correct one.

There are some common checks that are always performed, irrespective of which particular rule is applied. One of those checks whether the entered parts and the whole resulting expression is syntactically or mathematically incorrect, for example, missing parentheses or two multiplication signs next to each other, etc. Another common check for all rules is whether all necessary yellow boxes are filled in (some boxes may be left empty intentionally, for example, monomial coefficient 1, etc.). In case of an error, a message is displayed to the user and the yellow box with the error is indicated if possible (Figure 2.23).

Figure 2.23. Error in input of the result – empty yellow box

All further checks depend on the input mode and the rule. In the free input mode, a check of the structure of the result is performed in most cases (for example, when combining like terms, the result of the application of the rule should be a single monomial). Even if the entered expression and the correct one are equivalent, an error is still displayed to the user if the structure of the answer differs from the correct one (Figure 2.24). This is because we want the student to apply exactly the same rule that he chose and not to simplify something else (for example, multiply polynomials and immediately combine like terms). The other issue that is checked is the priority of operators – whether the student knows it and adds brackets if needed (for example, in the case of multiplication of polynomials if the product is a member of another multiplication). After the structure of the result is checked, the content is checked in exactly the same way as it is checked in the structured input mode. In some very sophisticated cases, only equivalence between the entered and correct expression is checked (for example, raising a multi-level fraction to a power, etc.).

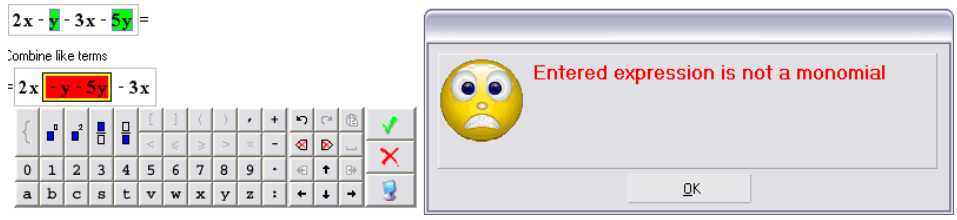


Figure 2.24. Error in input of result – incorrect structure of result in the free input mode

The free input mode is the closest to the way problems are solved in the Aplusix program. The latter only checks equivalence of the entered expression to the previous one. Lately, the authors of Aplusix have done some research in automatic diagnosis of incorrect rule. However, T-algebra checks not only equivalence of the whole expressions, but also equivalence of the exact result of the application of the selected rule to the correct result, priority of operators

etc., even in the most sophisticated cases. T-algebra diagnoses student's knowledge of applying exactly the same rule that he chose.

In the structured input and partial input modes, the resulting expression already has the correct structure, because the student is prevented from entering unsuitable parts into corresponding boxes (some structure checks are performed anyway, just in case). In these input modes the program checks the essential parts of the resulting expression separately in order to find the exact error and diagnose its cause. When the result is a single monomial, the operation sign, coefficient, variables and their powers are checked separately. When the result is a polynomial then the set of the monomials in the student's result is compared to the set of monomials in the correct result. If a difference is found, the exact error is diagnosed if possible. For example, when multiplying two monomials/ polynomials, the students often forget to consider both operation signs (pluses or minuses) or simply do not know the rules and calculate the resulting coefficient and variables with powers correctly but make an error calculating the operator sign (Figure 2.25).

In comparison with other similar systems, such full information diagnosis has many advantages – the student can be shown the exact place of error and the exact error type can also be diagnosed in most cases.

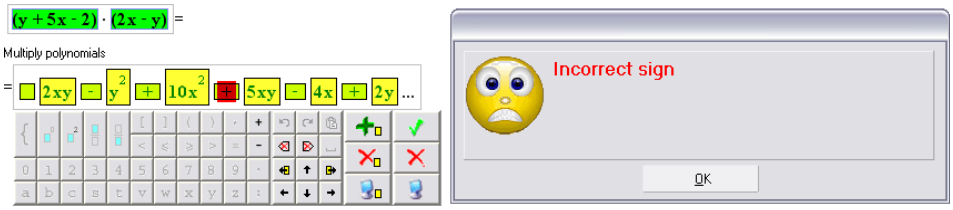


Figure 2.25. Error in input of result – wrong operation sign entered

2.8.6.4 Other errors in solution

We have reviewed different possibilities for making errors in the three stages of the solution step. However, there are other possibilities for making errors in T-algebra, which system is able to diagnose. Those cases are the following:

- input of additional rule information in a separate window – diagnosed similarly to final input as details are entered in yellow boxes in the editor;
- input of intermediate information for a solution step – diagnosed similarly to final input as details are entered in yellow boxes in the editor;
- input for extending the structure – errors in adding less or more than the correct number of boxes – diagnosed similarly to final input as details are entered in yellow boxes in the editor;

- error when giving answer to a problem (unfinished solution) – diagnosed when the student presses the button for giving an answer (if there are rules that need to be applied then an error message is shown);
- error when giving answer to a problem (choice of answer) – diagnosed in a separate window when choosing an answer (for example, for equation it is possible that any number is a solution or the equation has no solution).

2.9 Error categorization and student statistics

T-algebra calculates different statistics during the solving process. For that, every error that is made by the student or every hint usage is recorded. When recording the error or help usage, T-algebra records, for subsequent reviewing, many different attributes describing the current situation at the time when the error occurred / help was used. The student can possibly learn from his mistakes and the teacher can use this information on mistakes for assessment or information on help usages to identify the hardest parts of the material that might need revising. This statistics (error and help usage situations) is also saved to the solution file (.lah). Calculated statistics can be viewed from the *View* menu in the student's program.

2.9.1 Error categories and attributes

While the student is solving problems, T-algebra checks his steps and tries to diagnose errors (see section 2.8.6). Each time when an error is diagnosed, T-algebra records the error situation and tries to categorize the error.

We have designed 20 different categories and divided all diagnosed error types between them. The categories include, for example, selection of objects of wrong form, selection of incompatible objects, errors in the form of entered subexpression, calculation errors, errors in calculating the sign of entered subexpression, etc. For a full list of categories, see Appendix C. We have separated mathematical errors (which are probably caused by a mathematical misconception or mistake) and other errors, most likely caused by the use of T-algebra / computer for solving, which would probably not happen on paper. Those categories are quite common and therefore suitable for all rules and problem types that we have implemented. We did not want to confuse the teacher by introducing hundreds of categories and typical misconceptions that are specific only to a certain rule. We conducted a study (see section 4.4) where we tried to divide errors from a certain field into more detailed categories but found that the current implementation is quite useful as well.

For every error situation and every error check that we have implemented, there is a separate error message (stored in the language file) that is shown to the user if a check results in recognition of an error and a category for this error situation (for example, the results of the check for coefficient of monomial are,

in most cases, classified as calculation errors). For some most common checks, where it was hard to choose a category, there is a separate category of un-classified errors.

There are eight different situations where errors are diagnosed, with corresponding sets of attributes stored. Because of this, there are also minor differences in how error situations are displayed in the review window, but most differences are shown in one larger area. The following list catalogues those different situations where errors are diagnosed.

1. Selection errors in the editor – diagnosed by the editor and expression parser;
2. Rule selection errors – diagnosed by the problem type specific algorithm or general T-algebra solution engine;
3. Object selection errors – diagnosed by the selected rule error diagnosis and expression parser;
4. Additional information input errors – diagnosed by the selected rule error diagnosis (syntax errors are diagnosed using the expression parser);
5. Intermediate (or final in case of one input) input errors – diagnosed by the selected rule error diagnosis (syntax errors are diagnosed using the expression parser);
6. Final (second) input errors – diagnosed by the selected rule error diagnosis (syntax errors are diagnosed using the expression parser);
7. Result errors – diagnosed by the problem type solution engine;
8. Result selection errors (if used) – diagnosed by the problem type solution engine in a separate answer window.

For each error situation, different attributes are stored that fully describe the error situation. Some attributes are common for all situations:

- error category,
- error time,
- error code,
- message shown to student,
- problem number,
- problem type,
- selected rule,
- input mode.

Other specific attributes are added for different error situations, for example, *Selection errors in the editor* have the following additional attributes:

- expression in editor,
- objects already selected in editor,
- erroneous selection in editor.

Another example, *Input errors (in case of single input line for the rule)*, is more complex:

- expression in editor before applying the rule,
- selected objects for applying rule,
- additional info, if any (for example, common denominator),
- expression in editor (with boxes),
- types of boxes (number, sign, etc.),
- data entered by student in boxes,
- index of erroneous input box.

2.9.2 Help usage categories and attributes

When the student is solving problems, he is able to ask for help in different situations (see section 2.8.5). Each time when the student asks for help, T-algebra records the help usage situation. We grouped help usage into 7 categories depending on the place where help was asked.

1. Autosolve – automatic problem solving until an answer is given;
2. Rule selection hint – student asks which rule to apply next according to the solution algorithm;
3. Object selection help – T-algebra automatically selects suitable objects for the selected rule;
4. Help for additional information input – T-algebra automatically calculates additional information for the rule, for example, a common denominator;
5. Input help (first input – intermediate result in case of two lines and final input in case of one line input per rule) – T-algebra automatically fills in all the boxes in the result;
6. Input help (second input, final input for two lines per rule) – T-algebra automatically fills in all the boxes in the final input;
7. Help for extending the structure – T-algebra automatically adds the required number of boxes in the structured input mode in case of rules that require extending of the structure.

Similarly to error situations, there are numerous attributes that describe each help usage situation. Common attributes are:

- help usage category,
- help usage time,
- problem number,
- problem type,
- selected rule,
- input mode,
- message shown to student;
- last expression;

- objects selected in last expression (if any);
- solution step number (especially important in case of autosolve).

For example, in case of *Object selection help*, there is only one extra attribute in addition to the common ones. In this case, the user-selected objects (before asking for help) are stored under common attributes and the correct ones (selected by T-algebra) under this extra attribute:

- correct objects in expression (selected by T-algebra).

2.9.3 User interface for reviewing

As mentioned above, the user interface for reviewing can be used by both students and teachers. Students can, at any time, use the error list as a reference source and possibly learn from their mistakes. Teachers can open student solutions and use the information on mistakes for assessment or information on help usages to identify the hardest parts of material that might need revising. This user interface is accessible in T-algebra through the *View* menu.

First, it is possible to view *Error counters* and *Error list*. The error list displays a list of single error situations that can be reviewed one by one. In the error counters view, errors are grouped into categories and total numbers (per problem and total) are shown. It is possible to select errors of one category, errors in one problem, or even errors of one category in a certain problem for reviewing. This selection works as a filter for the error list view – in such case the error list displays only a limited set of error situations. The screenshot (Figure 2.26) shows two forms: the bottom one shows error counters grouped by category and the top one shows the list of error situations. The window displays the data that is stored for one error situation:

- which problem was being solved,
- initial expression of the problem,
- what rule was selected,
- input mode for the problem,
- the expression to be transformed at the moment of making the error,
- which objects were selected,
- error message shown to the student and red box indicating the error (if any) shown to the student.

In some cases there are more (or less) attributes to display, depending on the error category.

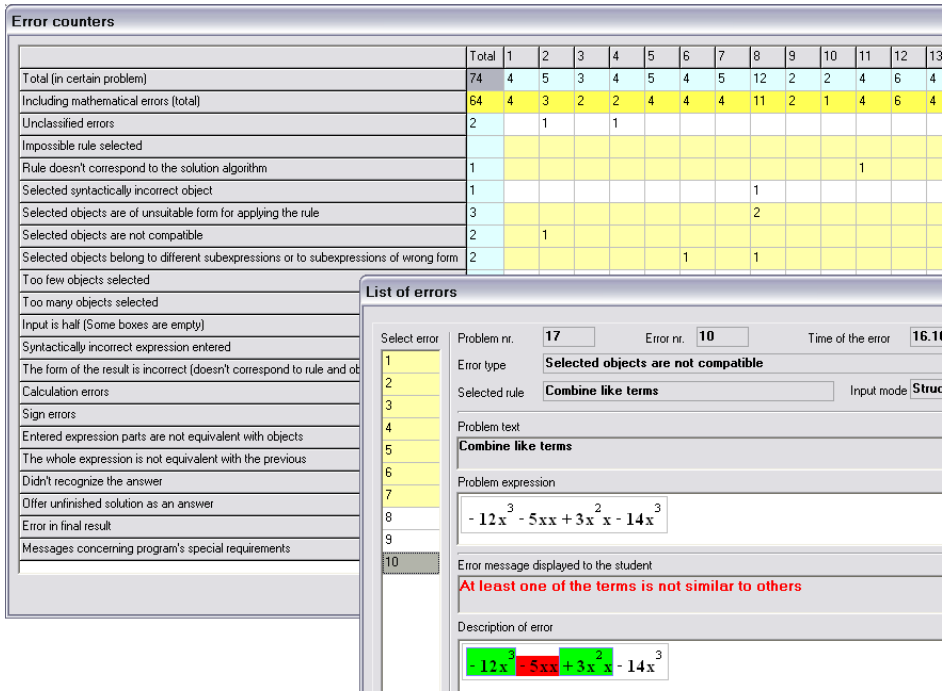


Figure 2.26. Error counters and description of a particular mistake

The other two items in the *View* menu are *Counters of help usage* and *List of help usage*. Those act similarly to the error counters and the list or errors – the first window shows counters of help usages grouped by categories and the other shows a list of help usages where each situation can be checked individually (Figure 2.27). The window displays the data that is stored for one help usage situation:

- which problem was being solved,
- initial expression of the problem,
- what rule was selected,
- input mode for the problem,
- the expression to be transformed at the moment of making the error,
- which objects were selected,
- what input boxes were offered to the student and what the student entered in those boxes before asking for help,
- the expression with boxes filled with correct input (generated by the help feature).

In some cases there are more (or less) attributes to show, depending on the help usage category.

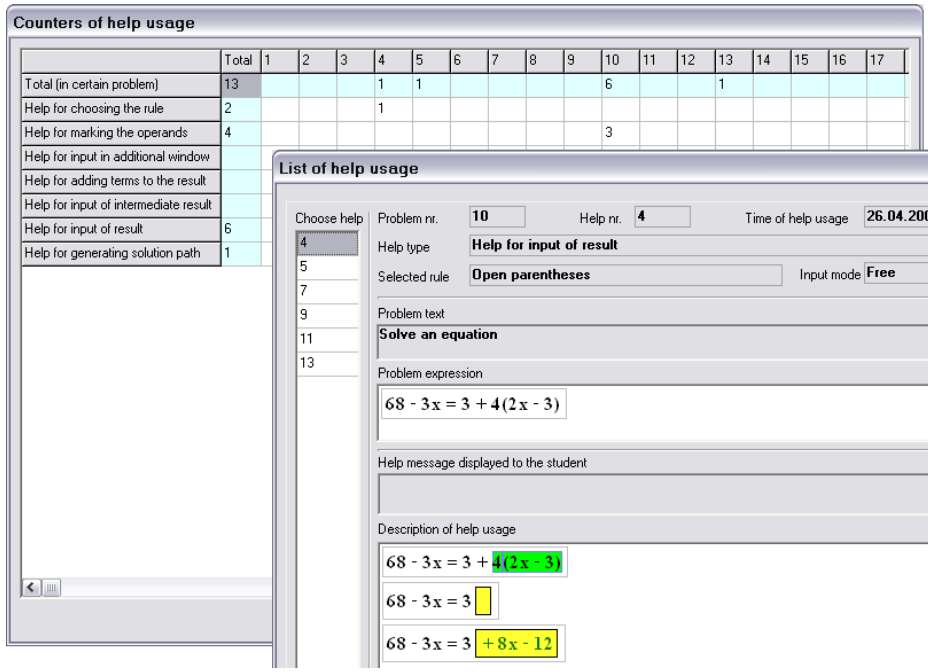


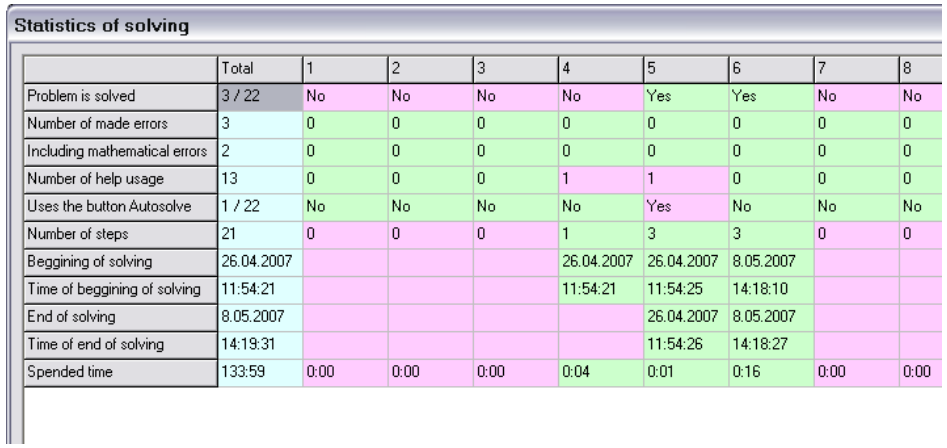
Figure 2.27. Counters of help usage and description of particular help

The last item in the *View* menu is *Statistics of solving*. This table enables to review general statistics of solving, like how many problems are solved and how many errors were made. This view combines both errors and help usages, but also shows other calculated attributes. An example is shown in the screenshot below (Figure 2.28). Fields on the pink background may require some attention from the teacher or student (for example, errors or help usages not zero for some problem, problem not completely solved, etc.). The following data is shown for each problem:

- problem is solved – indicating that the student has completely solved the problem and given an answer to the problem (is also true if the student used autosolve help);
- number of errors made – total number of errors per problem, same as in the error counters;
- including mathematical errors – total number of so-called mathematical errors per problem, same as in the error counters; this number is included in the total number of errors;
- number of help usages – total number of help usage situations per problem;
- uses of the autosolve button – true if the student has used the autosolve button while solving (at any step); this is included in the previous counter

but shown separately as it is quite important – it is counted as only one help usage, but it is possible that the student did not complete any steps by himself;

- number of steps – number of steps performed, even if the problem is not completely solved;
- date and time of beginning and end of solving the problem; if not completely solved then end is the latest time when any actions were performed on this problem;
- time spent on the problem.



Statistics of solving

	Total	1	2	3	4	5	6	7	8
Problem is solved	3 / 22	No	No	No	No	Yes	Yes	No	No
Number of made errors	3	0	0	0	0	0	0	0	0
Including mathematical errors	2	0	0	0	0	0	0	0	0
Number of help usage	13	0	0	0	1	1	0	0	0
Uses the button Autosolve	1 / 22	No	No	No	No	Yes	No	No	No
Number of steps	21	0	0	0	1	3	3	0	0
Beggining of solving	26.04.2007				26.04.2007	26.04.2007	8.05.2007		
Time of beggining of solving	11:54:21				11:54:21	11:54:25	14:18:10		
End of solving	8.05.2007					26.04.2007	8.05.2007		
Time of end of solving	14:19:31					11:54:26	14:18:27		
Spended time	133:59	0:00	0:00	0:00	0:04	0:01	0:16	0:00	0:00

Figure 2.28. Student statistics

3 PROBLEMS, RULES AND ALGORITHMS IN THE DOMAIN OF EXPONENTS, MONOMIALS AND POLYNOMIALS IN SCHOOL TEXTBOOKS AND IN T-ALGEBRA

I have chosen the domain of exponents, monomials and polynomials for exploration (related to my Master's thesis (Lepp, 2003b, Lepp, 2003a)) and implementation in T-algebra. First of all, school textbooks in mathematics were explored, then the rules were programmed, and finally problem types were composed. In this section I describe problems and algorithms in the domain of exponents, monomials and polynomials in school textbooks and in T-algebra on the basis of published articles (Lepp, 2006a).

3.1 Problems, definitions, rules and algorithms in schoolbooks

Prior to designing T-algebra, transformation rules and problem types, I studied them in school textbooks. For that, I examined different best-known and most used textbooks in Estonia. I also studied English school textbooks, but the Estonian ones were used as a basis, especially if any differences in rules were found. I paid more attention to the following issues: definitions, rules on how expressions are changed and also typical tasks in each topic. In the subsections of this section, I present the three different fields I was responsible for in T-algebra: exponents, monomials and polynomials.

Almost all typical tasks found in school textbooks are implemented in T-algebra as different problem types. At the end of each subsection, we mention typical tasks that were not implemented in the existing version of T-algebra, mostly word problems, most trivial reordering problems (transformations to normal form), some factorisation problems and simplification problems containing division by polynomials.

3.1.1 Exponents

"Exponents" is a topic in mathematics that is presented very differently in different textbooks. According to some textbooks, this topic is studied in the 7th grade (Tõnso, 2002), while other series of textbooks divide it between the 7th and 8th grades (Nurk et al., 2006 and Lepik et al., 2000; Pais, 1998 and Pais, 2001), and others introduce it entirely in the 8th grade (Veelmaa, 2004). In some textbooks this topic is presented as a separate chapter (for example, Pais, 1998); in others it is included under the topic "Monomials" (for example, Veelmaa, 2004).

In all textbooks the topic begins with a description of the number exponent. First, the square of number is described: "The square of number *a* (or the second power of number *a*) is product of this number *a* with *a*, and is written as $a^2 = a \cdot a$". Then the cube of number is introduced: "The product $a \cdot a \cdot a$ is called the cube of *a* (or the third power f number *a*) and is written as $a^3 = a \cdot a \cdot a$". After that the general definition of the power of number is given: "The *n*th power of number *a* is product of *n* factors of *a* or a^n means $a \cdot a \cdot a \ldots a$ (*n* times). Number *a* can be any number (positive, negative or zero). Number *n* should be a natural number higher than 1". After this definition, the definition of base and exponent is presented: "In the expression a^n, *a* is known as the base and *n* as the exponent (whereas a^n is the *n*th power of *a*)". In some textbooks (Nurk et al., 2006; Zuckerman, 1976) the exponentiation is described as the operation of determining a power of a number. In Estonian textbooks the following remark is also presented: "Applying exponent to a negative number the number should always be in parentheses". In addition, some books (McKeague, 1979) say that there are expressions in exponential form (like 3^4) and in expanded form (like $3 \cdot 3 \cdot 3 \cdot 3$).

After the exponent and exponentiation are described, the following typical exercise tasks are presented for students: "Find the value (raise to a power)"; "Convert expression to expanded form (or write power as product) and calculate"; "Convert expression to exponential form (or write product as power)".

After practice of writing and calculating the power, the properties of exponents are presented. The properties are presented in a different order in Estonian and English textbooks. Here we follow the order that can be found in Estonian textbooks.

Some books (Veelmaa, 2004; Barnett et al., 1990) introduce the properties with examples: "$9^3 \cdot 9^2 = (9 \cdot 9 \cdot 9) \cdot (9 \cdot 9) = 9 \cdot 9 \cdot 9 \cdot 9 \cdot 9 = 9^5$". After examples, or right in the beginning in some textbooks, the first property is given. In Estonian textbooks, the property is formulated in the following way: "To multiply two powers of the same variable (number), write down the base and add the exponents, or $a^m \cdot a^n = a^{m+n}$". In English textbooks, a more formal definition is given: "If *a* is a real number, and *r* and *s* are positive integers, then $a^r \cdot a^s = a^{r+s}$. The product of two expressions with the same base is equivalent to the base raised to the sum of the exponents from the original two expressions". In English textbooks the justification for property is also presented.

After introducing the property, typical problems like "Simplify by rewriting the expression so that the variable occurs only once"; "Find the product of powers"; "Simplify the expression using the property of exponents";

"Calculate", are practiced in some textbooks. In other textbooks the same typical tasks can be found after description of several properties.

The second property in Estonian textbooks is division of powers. Again, some books give an example for introduction: "$2^5 : 2^3 = \frac{2^5}{2^3} = \frac{2 \cdot 2 \cdot 2 \cdot 2 \cdot 2}{2 \cdot 2 \cdot 2} = 2 \cdot 2 = 2^2$". The definition of the property itself is "To divide two powers of the same variable (number), write down the base and subtract the exponents, or $a^m : a^n = a^{m-n}$". In some textbooks, a different style for writing division is used: $\frac{a^m}{a^n} = a^{m-n}$. In English textbooks this property is given as the last property (the fifth or the sixth) and the formulation is: "If *a* is any real number, and *r* and *s* are any two integers, then $\frac{a^r}{a^s} = a^{r-s}$". With the help of the second property, the next remark is introduced in some textbooks: "From one side $2^5 : 2^5 = \frac{2^5}{2^5} = \frac{32}{32} = 1$. From other side $2^5 : 2^5 = 2^{5-5} = 2^0$. From this follows that $2^0 = 1$". This example is generalized as follows: "Every number in power zero is one, or $a^0 = 1$". In English textbooks (Barnett et al., 1990) it is noticed that "0^0 is not defined". In some books (Veelmaa, 2004) one more remark is introduced: "From one side $a^5 : a^4 = a^{5-4} = a^1$. From other side $a^5 : a^4 = \frac{a \cdot a \cdot a \cdot a \cdot a}{a \cdot a \cdot a \cdot a} = a$". From this follows: "Every number in power 1 is this number itself, or $a^1 = a$".

After learning this property and those remarks, students practice the following problems: "Simplify the expression using the property of exponents", "Find the quotient of powers", "Calculate".

The next property involves negative-integer exponent: "If $a \neq 0$, then $a^{-n} = \frac{1}{a^n}$ and $a^n = \frac{1}{a^{-n}}$". English textbooks give a more formal definition of negative-integer exponents: "If *a* is any nonzero real number and *r* is a positive integer, then $a^{-r} = \frac{1}{a^r}$. It follows, using equality property, that $a^r = \frac{1}{a^{-r}}$". From this property it is derived that negative exponents indicate reciprocals $a^{-1} = \frac{1}{a}$.

In order to clarify these definitions, the following problems are solved: "Calculate"; "Write as negative exponent"; "Simplify"; "Simplify and then calculate"; "Simplify, leaving answers with negative exponents"; "Simplify, leaving answers with positive exponents".

The next property again is presented with examples like "$(5^3)^2 = 5^3 \cdot 5^3 = 5^6$ Notice: $3 \cdot 2 = 6$". The result is generalized as: "To raise a power to another power, write down the base and multiply the exponents, or $(a^m)^n = a^{m \cdot n}$". English textbooks formulate this property as follows: "If *a* is a real number and *r* and *s* are positive integers, then $(a^r)^s = a^{r \cdot s}$. An expression with an exponent, raised to another power, is the same as the base from the original expression raised to the product of the powers." English textbooks also provide proof for this property.

We found the following typical exercise tasks for this property in the textbooks: "Calculate"; "Raise to a power"; "Simplify".

The next property of exponents arises when we have a product of two or more numbers raised to an integer power. For example: "$(3x)^4 = (3x)(3x)(3x)(3x) = (3 \cdot 3 \cdot 3 \cdot 3)(x \cdot x \cdot x \cdot x) = 3^4 \cdot x^4$". This leads to the property: "To raise a product to a power, raise every factor to a power and multiply the results, or $(ab)^n = a^n \cdot b^n$" or (English textbooks): "If *a* and *b* are any two real numbers, and *r* is a positive integer, then $(ab)^r = a^r \cdot b^r$". The English textbooks provide a justification as well.

In order to clarify this property, the following problems are solved: "Raise to a power"; "Simplify and then calculate"; "Calculate"; "Multiply and then raise to a power"; "Raise to a power and then multiply"; "Simplify".

The last property in Estonian textbooks is formulated for fractions: "To raise a fraction to a power, raise numerator and denominator to a power and divide them, or $(\frac{a}{b})^n = \frac{a^n}{b^n}$". English textbooks do not mention fractions, but talk about division as follows: "The last property was stated for products. Since division is defined in terms of multiplication, we can expect a similar property involving quotients. "If *a* and *b* are any two real numbers with $b \neq 0$, and *r* is a positive integer, then $(\frac{a}{b})^r = \frac{a^r}{b^r}$", and also present proof.

Typical tasks for practicing this property include: "Raise to a power"; "Simplify and then calculate"; "Calculate"; "Divide and then raise to a power"; "Raise to a power and then divide"; "Simplify".

After all properties are described, students are given problems requiring application of several properties, like "Calculate" and "Simplify".

Almost all typical textbook problems from the field of exponents were implemented in T-algebra as different problem types. Probably the only one that was left out is conversion of power to a product, which is quite trivial. For all other transformation rules presented here, specific problems were created, and also a separate problem type was added for calculation of the values of expressions.

3.1.2 Monomials

In English textbooks the topic "Monomials" is presented together with the topic "Polynomials". In Estonian textbooks monomials are described separately or with the exponents in the 8th grade. That is why the order of presentation of definitions is different in Estonian and English textbooks. We follow here the Estonian textbooks. In all textbooks that we have studied (except Barnett et al., 1990) this topic begins with the definition of monomial: "Monomial is the product of a constant and one or more variables raised to a whole-number exponent. Single number is also a monomial." In some textbooks (Lepik et al., 2000; Zuckerman, 1976) the definition of variable is given beforehand: "A variable is a symbol that represents any one of a given collection of numbers". In Barnett et al., 1990, monomial is described as one-term polynomial. In Estonian textbooks the definition of monomial in normal form (or monomial in standard form or simplified monomial) is as follows: "The monomial is in normal form if it begins with a numerical factor (with sign of term), followed by variables with exponents in alphabetical order". A definition of coefficient of monomial is also presented: "The coefficient of monomial is the single occurrence of a numerical factor when the monomial is in the normal form". It is remarked that coefficient *1* is not written and a minus sign before monomial means the coefficient *-1*. Finally, the definition of like monomials is presented: "Monomials, which are the same or differ only by coefficient, are called like monomials". The process of combining is revised (the process of combining like terms was introduced in the 7th grade under the topic "Linear equation") as follows: "Like monomials are combined by adding their coefficients".

There are not too many practice exercises for these definitions, but we did find a few: "Transform monomials to normal form"; "Find like monomials"; "Calculate the value of monomial"; "Combine like terms".

After such introduction almost all textbooks explain the multiplication of monomials. The following technique for multiplication is given: "To multiply monomials, rearrange the factors:

1. group all coefficients at the beginning;
2. group powers of the same variable together.

Multiply the coefficients. Multiply powers of the same variable (add the exponents)".

In order to clarify this technique, problems like "Multiply" or "Simplify" are solved.

English textbooks do not describe anything else about monomials. Estonian textbooks, in addition to the aspects mentioned, also present raising monomials to a power and division of monomials.

Raising monomials to a power is explained with reference to multiplication of monomials and examples: "As raising to a power can be replaced with the multiplication, then $(2xy^2)^3 = 2xy^2 \cdot 2xy^2 \cdot 2xy^2 = 2^3 \cdot x^3 \cdot (y^2)^3 = 8x^3y^6$". This concludes to: "To raise a monomial to a power, raise its every factor to a

power". After these examples, the typical problems "Raise to a power" and "Simplify" are practiced.

For division of monomials, the Estonian textbooks mostly use the symbol ":", for example $24x^2y^3 : 6xy$. Actually, this division should be presented like $24x^2y^3 : (6xy)$. Only one textbook (Veelmaa, 2004) mentions that parentheses should be written but there is an agreement that they are not written for the sake of simplicity. Therefore, division $24x^2y^3 : 6xy$ means $\frac{24x^2y^3}{6xy}$.

The technique for division of monomials is described as follows: "To divide monomials:

1. find the quotient of coefficients,
2. find the quotient of variables (subtract the exponents),
3. multiply the results."

It is also mentioned that, for division of complicated monomials, it is reasonable to write down division as fraction and then to reduce the fraction. Likewise, it is better to leave variables with positive exponent in the result (in the denominator if needed).

After this subtopic the students solve problems: "Divide", "Reduce" and "Simplify".

At the end of this topic, students practice simplification problems that involve multiplication, raising to a power and division of monomials.

Most of the typical textbook tasks from the field of monomials were implemented in T-algebra. The ones that were left out (find like terms, transform to normal form) were based on definitions, as no expression transformation takes place in some cases or transformations are trivial. In addition, multiplication of monomials is very similar to transformation to normal form, except T-algebra does not have special rules for reordering variables. Division of monomials in the form of a fraction is not supported (only the division sign is used) and reduction of fractions with monomials is not implemented (rules for operations with fractions only work with numbers). For all other transformation rules presented here, specific problems were created, and also a separate problem type was added for calculation of the values of expressions.

3.1.3 Polynomials

The topic "Polynomials" begins with the definition of polynomial: "Polynomial is defined to be a sum of monomials". Monomials used for sum are named terms of polynomial. The definition of like monomials and combining of like monomials (presented earlier, see 3.1.2) is repeated once more. This is needed for the definition of polynomial in normal form: "To transform polynomial into normal form, combine like monomials, order the monomials decreasingly according to the sum of exponents of variables in monomial and finally transform monomials into normal form". Then the definitions of binomial and

trinomial are given: “If a polynomial consists of two unlike terms, it is said to be a binomial. If it has three unlike terms, it is called a trinomial”. The English textbooks also present the definition of the degree of polynomial: “The degree of a term in a polynomial is the sum of the powers of the variables in the term. The degree of polynomial is the degree of its term with the highest degree”.

There are not many problems for practicing these definitions, but we found the following problems: “Combine like terms”; “Evaluate a polynomial for specific values of variables”; “Simplify”.

The next subtopic is addition and subtraction of polynomials: “To add polynomials, write one polynomial after the other with the same marks of terms and combine like terms if needed. To subtract polynomials, write one polynomial after the other with the opposite marks of terms and combine like terms if needed“. The English textbooks also propose to add polynomials vertically: “Rearrange the terms so that like terms are in the same column and add their coefficients”. The rule of clearing parentheses is also described for addition and subtraction of polynomials: “If there is a positive sign directly preceding the parentheses surrounding a polynomial, we may just remove the parentheses. If there is a negative sign directly preceding the parentheses surrounding a polynomial, we may remove the parentheses and preceding negative sign by changing the sing of each term within the parentheses”.

After this subtopic the students solve problems: “Add”; “Subtract”, “Simplify”; “Evaluate a polynomial for specific values of variables”.

Following the addition and subtraction of polynomials, the multiplication of polynomials by monomial is described: “To multiply a polynomial by a monomial, multiply the monomial with every term of the polynomial and add the results”. The English textbooks do not present this technique separately; they just mention that it is possible to use distributive law for multiplication of polynomials by monomial.

The tasks for practicing are “Multiply”; “Combine and then multiply”; “Simplify”; “Evaluate a polynomial for specific values of variables”.

The next rule in the Estonian textbooks is division of polynomials by monomial: “To divide a polynomial by a monomial, divide every term of the polynomial by the monomial and add the results”. The English books again describe the division differently: “To divide a polynomial by a monomial, use the definition of division (replace division with multiplication) and apply the distributive property”.

We found the following problems after that theme in the textbooks: “Divide”; “Simplify”; “Calculate”.

After multiplication/division of polynomial by monomial, the reverse operation (to multiplication), i.e., factoring out common factors, is explained in the Estonian textbooks. First, the relationship between multiplication and factoring is described and the definition of factoring is given: “Transformation of a polynomial to a product is called factoring of the polynomial”. The following technique for factoring polynomial is presented: “To factor a polynomial

1. find the common factor of all terms of the polynomial (a monomial that divides (is a factor of) each term of the polynomial);
2. write the common factor before (or after) the parentheses;
3. write into the parentheses the polynomial that remains after the given polynomial is divided by the common factor."

English textbooks do not describe factoring at full length; they only refer to the distributive property: "View the distributive property from right to left and rewrite a sum as a product".

In order to clarify this operation, the following problems are solved: "Factor out the given factor"; "Factor out the greatest common factor"; "Calculate as simply as possible"; "Evaluate a polynomial for specific values of variables".

The next operation to study is multiplication of binomials. Multiplication of binomials is explained in the Estonian textbooks with the help of area of rectangle with sides $(a+b)$ and $(c+d)$. This rectangle is divided into 4 rectangles and their areas are ac, ad, bc and bd. Then it is derived that $(a+b)(c+d)=ac+ad+bc+bd$. The same result is obtained when the distributive property is applied twice. Then the rule is formulated as follows: "To multiply binomial by binomial, multiply each term of one binomial by each term of the other and add the results". The English textbooks describe the FOIL method (First product, Outer product, Inner product, Last product) for quick (mental) multiplication of binomials.

The tasks "Multiply" and "Simplify" are practiced to understand multiplication of binomials.

The next subtopic is the reverse of multiplication of binomials, namely, factoring by grouping: "In some situations it is possible to take a polynomial with no apparent common factor and find one in two steps (when the terms are properly grouped):

1. rearrange and group terms (sometimes you have to rearrange the terms for several times to find proper groups);
2. remove common factor from each group;
3. take out the common factor to complete factoring.

The method is also suitable for trinomials, but before factoring a trinomial you have to write it out with four terms".

The problems "Factor by grouping" are solved under this subtopic.

Several subsequent subtopics in the Estonian textbooks describe different formulas for different (special) products:

- formula for difference of squares $(a+b)(a-b)=a^2-b^2$: the product of the sum of two monomials and the difference of the same monomials is the difference of squares of these monomials (English textbooks: To multiply two binomials which differ only in the sign between their two terms, simply subtract the square of the second term from the square of the first term);

- formula for square of sum $(a+b)^2 = a^2 + 2ab + b^2$: the square of the sum of two monomials is the square of the first monomial plus double product of the first and second monomials plus the square of the second monomial (English textbooks, formula for binomial squares: the square of binomial is the sum of the square of the first term, twice the product of the two terms, and the square of the last term);
- formula for square of difference $(a-b)^2 = a^2 - 2ab + b^2$: the square of difference of two monomials is the square of the first monomial minus double product of the first and second monomials plus the square of the second monomial.

The problems used to practice these formulas include: "Multiply"; "Use the formulas"; "Simplify".

After using formulas for simplifying polynomials, students are taught factoring by these formulas. Left and right sides of formulas are exchanged and formulas for perfect square trinomial ($a^2 + 2ab + b^2 = (a+b)^2$ and $a^2 - 2ab + b^2 = (a-b)^2$) and the difference of two squares ($a^2 - b^2 = (a-b)(a+b)$) that can be used for factoring are received. The problems "Factor out" are practiced after the formulas have been described.

The next rule is multiplication of polynomials, where multiplication of binomials is extended to an arbitrary polynomial: "To multiply two polynomials, multiply each term in the first polynomial by each term of the second polynomial and add the results". The English textbooks also present a method for multiplication that looks very similar to long multiplication with whole numbers: "The polynomials are lined up vertically, then the rule for multiplication of polynomial by monomial is applied and results are added in columns".

Again the problems "Multiply", "Simplify" and "Evaluate a polynomial for specific values of variables" are solved.

At the end of the topic "Polynomials", students are taught some more formulas based on multiplication of polynomials:

- formula for sum of cubes $(a+b)(a^2 - ab + b^2) = a^3 + b^3$: the product of the sum of two monomials and incomplete square of the difference of these monomials is the sum of cubes of these monomials;
- formula for difference of cubes $(a-b)(a^2 + ab + b^2) = a^3 - b^3$: the product of difference of two monomials and incomplete square of the sum of these monomials is the difference of cubes of these monomials;
- formula for cube of sum $(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$: the cube of binomial is plus triple product of the square of the first term and second term plus triple product of the first term and the square of the second term plus the cube of the second term;
- formula for cube of difference $(a-b)^3 = a^3 - 3a^2b + 3ab^2 - b^3$.

The use of these formulas after reversing the sides for factoring is demonstrated as well.

During the study of these formulas, the following typical tasks are practiced: "Multiply"; "Simplify"; "Use the formulas"; "Factor out"; "Simplify and evaluate".

The English textbooks present the subtopic "Division of polynomial by polynomial" under this topic. Since this subtopic is not described in the Estonian textbooks, we will not describe it here in greater detail.

At the end of the topic "Polynomial", typical problems "Simplify" and "Factor out", involving different operations and techniques, are solved for rehearsal.

Most typical simplification problems from the field of polynomials, found in textbooks, were implemented in T-algebra. The ones that were left out are based on definitions (transform to normal form) or require factorisation and division (reduction) of polynomials by polynomials. Division by monomials in the form of fractions is not supported (only the division sign is used). As far as factorisation problems are concerned, only the simplest factoring out monomial of polynomial is implemented and no grouping technique and use of special formulas for factoring is supported in the current version. For all other transformation rules presented here specific problems were created, and also a separate problem type was added for calculation of the values of expressions.

3.2 Designed rules in T-algebra

We started working with transformation rules for step-by-step problem solving environments already when designing the "Polynom" program, which was the main part of my Master's thesis (Lepp, 2003b, Lepp, 2003a). At that moment we designed and implemented rules for simplifying polynomial expressions (Lepp, 2006a). Although we tried to follow a similar scheme for the implementation of different rules, the result was not perfect. The implementation was slightly different for different rules and it was also different from the paper and pencil way of solving problems.

Before starting to design and implement rules for T-algebra, we tried to study the problems that caused differences in the implementation of rules in the Polynom system and define a certain scheme to follow when designing different rules.

The Polynom system used a strictly ordered Action – Object – Input scheme. The user first had to select an operation and mark the objects only after that. Implementation of the object selection was dependent on the selected transformation rule (action); input of the result differed between implemented rules – this all caused differences in the implementation of different rules.

In T-algebra we made changes to this scheme. First of all, we allowed selection of the rule and objects in an arbitrary order and it lead to the same object selection scheme for all rules. Then we implemented three different

standard input modes for each rule (described in section 2.5.3). Other changes included new substeps for the solution steps of some rules: additional information, intermediate result and adding terms (described in section 2.6).

3.2.1 Common checks for three stages of step

The solution step scheme we defined lead to a situation where T-algebra performs some common checks after the stages of solution steps. The common checks for different stages are listed in this subsection and only rule-specific checks are mentioned in the descriptions of particular rules.

As the rule and object selection can now be performed in an arbitrary order and a selected rule can even be changed, we do not check the selection of the rule separately; it is checked together with objects. Consequently, after the second stage of the step (marking parts of expression to apply the rule to) the program checks:

- whether the selected rule is applicable to the current expression (i.e., there at least exists a set of objects to which the rule is applicable);
- performance of marking (whether some parts are marked if needed);
- syntactical correctness of marked parts;
- number of marked parts (only one needed, at least two needed, *etc.*, described for every rule separately);
- form of marked parts (may differ depending on the rule, described for every rule separately);
- position of marked parts (in some cases, described for every rule separately).

The input stage in the Polynom system (the system designed and implemented for my Master's thesis) was also very different for different rules. The amount of input required and the structure of the input were variable. For some rules, only the most essential parts of result were entered by the student while for other rules, the whole result had to be entered. After studying the implementation of rules in Polynom, we extracted three different possibilities for the input stage and tried to implement all three input modes in T-algebra (see section 2.5.3). There are several common checks that T-algebra performs after the input stage in different input modes.

As my responsibility in T-algebra was to design and implement transformation rules that are specific to the monomial and polynomial topics, I defined the set of common checks for rules specific to these topics. The result of applying the simplification operations in these topics is always either a single monomial or the polynomial (which is a sum of monomials).

After the input stage in the **free input mode** T-algebra checks:

- completion of input of the result (the boxes are not empty) – in some cases boxes can be left empty (for example, when the result of combining like terms is 0 and there are other terms in this sum expression);
- syntactical correctness of entered parts;

- equivalence of the entered parts to the parts calculated by the computer.

Further checks depend on the form of the result. If the result is exactly one monomial then T-algebra checks separately:

- form of result (exactly one monomial);
- set of variables with powers;
- coefficient;
- sign.

If the result is a polynomial (sum of monomials) then the current version of T-algebra only diagnoses and alerts about non-equivalence of the entered expression to the one calculated by the computer. No further checks are performed in the free input mode. This actually leads to several issues worth mentioning:

- students can make additional simplification steps when applying rules (for example, combine like terms when multiplying polynomials), which are not possible in structured and partial input modes;
- students can omit some calculations (for example, when multiplying polynomials they write the coefficient as a product of two coefficients) while T-algebra requires a stricter form of the result in structured and partial input modes.

In the structured input mode T-algebra checks:

- completion of input of the result (the boxes are not empty) – in some cases boxes can be left empty (for example, coefficient 1);
- syntactical correctness of entered part in every box;
- form of every part (may differ depending on rule, described for every rule separately);
- equivalence of the entered parts to the parts calculated by the computer;
- every component of term (mark, coefficient, variable).

In addition, some rules include the possibility to add terms to the result during the input stage in the structured input mode. For example, in multiplication of polynomials, structures for inputting monomials can be added to the result. There is one common check that T-algebra performs in case of all rules that use adding of terms:

- number of added terms (in this mode, the student should not combine any like terms when multiplying polynomials, etc., therefore, the correct number of terms can be easily defined).

In the partial input mode T-algebra checks:

- completion of input of the result (the boxes are not empty) – in some cases boxes can be left empty (for example, power of variable 1);
- syntactical correctness of entered parts;
- equivalence of the entered parts to the parts calculated by the computer.

3.2.2 Transformation rule implementation principles

While designing the implementation for the transformation rules, some problems had to be solved. We had to create an interface that would allow simultaneous implementation of transformation rules for different topics separately by many developers. We also had to implement special expression object classes and a common functionality for creating input boxes in the expression structure and the expression editor (described in section 2.7).

Therefore, we created the interface for the transformation rules by creating the base class `TRule` and for storing additional step information `TRuleObject` with all the required functions and data structures declared. It also included implementation of functions that are common to different transformation rules, for example, a large number of functions for checking the type of subexpression: is the subexpression a monomial or polynomial, power of monomial, number or fraction, etc. The transformation rule also defines certain specific error messages that are displayed to the user while solving problems when checks are performed.

When a developer needed to add a new transformation rule they simply had to extend the `TRule` class and override methods with custom functionality. Then the newly created class should be registered on a special list to be made available to both the student and teacher programs. We tried to develop an interface where creation of separate rules would need very limited effort from the developer – mostly rule specific functions for objects selection, automatic rule application, creation of the structure of input boxes, and checks for user input and object selection.

As the number of transformation rules is quite high, the number of different classes is also high. Working with them is quite convenient as they all implement one interface `TRule`. The only problem is storing the state, solution files and reading solutions from the solutions file. Therefore, internally we used different constants (IDs) that define certain transformation rules and stored those IDs. For creating a rule from a given ID, also the scope (Problem) for the rule has to be passed as a parameter to the special method that was designed and no explicit calls to constructor are used at all:

- `class function` **`CreateRuleById`**`(RuleId: Integer; Problem: TProblem): TRule;`

The `TRule` class has many different methods for simpler creation of separate transformation rules. Some methods are specific and not used elsewhere. Here we present the main functions and data structures of the base class that are used in every rule and that are called from other places in T-algebra.

- `function` **`RuleSteps`**`(const Mode: Integer; const RuleObject: TRuleObject): Integer;`

The function `RuleSteps` defines whether a rule in this mode uses input of intermediate result (additional row for data input). The default return value is 1, meaning only 1 input row per step.

- `procedure` **`Check`**`(const Avaldis: TAvaldis);`

This function gets the object of expression as the parameter. This expression contains information on the selected objects. The function `Check` verifies the student's selection of objects and checks whether the rule is applicable to these objects. If an error is found, a special type of exception is thrown and a message displayed to the user.

- `function` **`FindApplySels`**`(const Avaldis: TAvaldis): TSelections;`

This function gets the expression as the parameter, finds and returns information on objects suitable for application of the rule. If a rule is not applicable then `NULL` is returned. When the function is called, the first suitable set of objects is returned. This function holds some heuristics. Some rules work in a slightly different way for different problems. The problem is given as a scope to the rule during creation. This function is used widely in the solution algorithm for checking whether the rule is applicable and in the problem type heuristic function for choosing the next rule to apply.

- `functions` **`DialogueInput`**`*`

Multiple functions that are responsible for input of additional information in the popup: window initialisation, input structure creation, automatic result / help, etc. In my three topics, these functions are used only in factorisation.

- `procedure` **`Apply`**`(var Avaldis: TAvaldis; const Mode: Integer; const RuleObject: TRuleObject);`

The procedure `Apply` modifies the initial expression given as the parameter and applies the rule in given mode using additional information entered in the popup (or RuleObject = `NULL`). The information on the structure of input boxes as well as on the `constraints` for different boxes is stored to the resulting expression object. That object is used for empty input structure generation as well as automatic result calculation.

- `procedure` **`Analyze`**`(const Avaldis: TAvaldis; const Substs: TStrings; const Mode: Integer; const RuleObject: TRuleObject);`

The procedure `Analyze` checks the result of application of the rule as entered by the student. The procedure gets the initial expression, additional rule information and input mode as the parameters. It is able to apply the rule and compare the correct result to the input of the student (given as parameter `Substs` that contains parts of expression entered in input boxes). If an error is found, a special type of exception is thrown and a message displayed to the user.

- `procedure` **`Apply2`**`(var Avaldis: TAvaldis; const Substs: TStrings; const Mode: Integer; const RuleObject: TRuleObject);`
- `procedure` **`Analyze2`**`(const Avaldis: TAvaldis; const Substs: TStrings; const PSubsts: TStrings; const Mode: Integer; const RuleObject: TRuleObject);`

The procedures `Apply2` and `Analyze2` work similarly to `Apply` and `Analyze` – those are used to generate the next expression (apply rule) and check student input (analyze), but are used in case of input of an intermediate result. Input of an intermediate result is handled by means of procedures `Apply` and `Analyze` (corresponds to the first input line of the rule). The procedures `Apply2` and `Analyze2` are responsible for the final input (2 corresponds to the second input line of the rule). Again, if an error is found, a special type of exception is thrown and a message displayed to the user.

3.2.3 Usage of transformation rules in T-algebra

As transformation rules (appropriate classes) in T-algebra contain a large number of attributes and algorithms, these are also widely used. Both the T-algebra student program and the teacher program require these transformation rules.

The student's program uses transformation rules while solving problems to:

- check the selection of objects for application of the rule;
- help the student and select correct objects for application of the rule automatically;
- generate structure for inputting the resulting expression, intermediate result or additional information for the rule;
- check the student's input in the boxes;
- fill in input boxes with correct parts of the result of application of the rule;
- generate an automatic solution – apply rules automatically to the current expression;
- check whether the rule is applicable to the current expression – mostly used by the problem solving algorithm and heuristic functions for problem types;
- check whether generated expression is suitable for the problem type – whether some rules are applicable.

The teacher's program uses transformation for two main reasons:

- generate an automatic solution – apply rules automatically to the current expression;
- check whether generated or entered expression is suitable for the problem type – whether some rules are applicable.

3.2.4 Scheme for presentation of transformation rules

In the following sections, we present transformation rules that I have designed and implemented in T-algebra for problems in two different fields: exponents, monomials and polynomials. Some of these rules are also used in other fields (for example, solving of linear expressions), therefore, there are no limitations on the expressions the rule can be applied to.

A similar scheme is used to describe each transformation rule. We describe the most important aspects and give some example of application of the rules in T-algebra in different input modes. The following attributes are discussed:

- Applications – possible applications of the rule, also topics (other than polynomial simplifications) where the rule is used;
- Expression – constraints, if any, on the expressions that can be used for applying the rule. Mostly there are no limitations on the expressions so this item is skipped;
- Instruction for marking – instructions that T-algebra displays for the student when this rule is selected;
- Marking – description of the marking process for this rule: what parts should be marked, checks that T-algebra performs after confirmation of the marking stage, help on selection of objects, and one example of the objects selection (screen capture);
- Instruction for input of additional information – instructions that T-algebra shows to the student when it requests input of some rule specific information in a popup window (if the rule does not use the additional information popup then this part is skipped);
- Input of additional information – description of input of some rule specific information in a popup window – input format, checks performed, etc. (if the rule does not use the additional information popup then this part is skipped);
- Instruction for input of intermediate result – instructions that T-algebra shows to the student when input of an intermediate result of the solution step is required (if the rule does not use this kind of input then this part is skipped);
- Input of intermediate result – description of input, form of the result, checks performed, example, etc. (if the rule does not use this kind of input then this part is skipped);
- Instruction for input (in each mode) – instructions that T-algebra shows to the student when input of the result of the solution step is required in a certain input mode;
- Input of result (in each mode) – the description of input process for this rule: what boxes are offered, checks that T-algebra performs after confirmation of input, help on input of the result (automatic solving) and one example of input (screen capture);
- Adding input boxes to the input structure – description of extending the input structure in the structured input mode: what kind of boxes are added, checks that T-algebra performs (description is skipped if adding is not used).
- Error messages – for two rules (*Combine like terms* (the result is monomial) and *Multiply polynomials* (the result is polynomial)) I also present all error messages that the program shows to the student in case of a mistake.

3.2.5 Rule Combine like terms

Applications: combine like terms; add/subtract numbers (as numbers are like terms), also add and subtract fractions. This rule is also used in solving of linear equations, linear inequalities and systems of linear equations.

Instruction for marking: Mark monomials for combining (only one group of like terms).

Marking: The student has to select like terms to combine (Figure 2.1). The student has to select objects precisely – only like monomials for combining should be selected. It means, for example, that it is impossible to select the whole sub-expression $2x+3y-5x$ for combining like terms in it, but different parts should be selected separately. However, if two or more objects are placed next to each other in the expression these can be selected as one part of selection (for example, like terms in the expression $2x-5x+3y$). If a monomial is put into parentheses (because of negative coefficient, for example $2x+(-5x)$) it can be combined directly without removing parentheses. In this case, such monomial should be selected with parentheses. Only one group of like terms can be combined at one step. During the selection of objects, the student can ask for help and the program selects one group of like terms for combining or shows a message if there are no terms to combine.

After confirming the selections, T-algebra checks correctness of the objects: whether all the selected objects are monomials, whether there are at least two monomials selected, these monomials belong to the same sum expression (also if a monomial in parentheses is selected without parentheses) and all of them are like terms. The program does not require all like terms of the same type to be marked. The program also considers as like terms the parts in which the variables are in different order. It means, for example, that the parts $5ab^2$, $5b^2a$ and $8bab$ are considered as like terms.

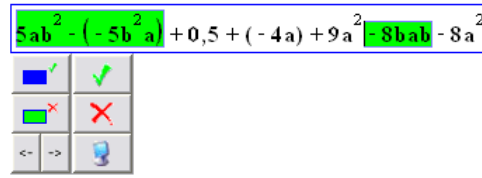


Figure 3.1. Marking stage in applying the rule Combine like terms

Error messages after marking (the message itself does not contain brackets; the text in brackets is given to explain the situation when and/or how this error is diagnosed):

- At least two terms should be selected for this operation (empty selection or not enough objects);
- Sign cannot be selected without following term (syntax check);

- The term should be selected together with parentheses (check for position of objects – same level in the tree);
- Selected terms should be monomials (check for the form of object);
- Selected terms are not members of a sum (check for position of objects – same level in the tree);
- At least one of the terms is not similar to others (rule specific check);
- Selected terms are not members of the same sum! Selected terms cannot be combined (check for position of objects – same level in the tree);
- Terms for collection should belong to one equation (special check for the field of equations, position of objects);
- This rule allows to collect terms only in one side of the equation (special check for the field of equations, position of objects).

Instruction for input of result (free input mode): Enter the result of combining.

Input of result (free input mode): After marking, if all checks are passed, T-algebra copies unchanged parts of the expression to the next line and offers input boxes for entering the resulting monomial. In case of the free input mode, one box for result is proposed. The position of the input box depends on the objects selected. In case of combining like terms (as also in case of most rules), the input box is placed to the position of the first selected object (see also the example in Figure 3.2).

The student should enter the whole monomial, including the preceding sign, in this box (Figure 3.2). The student can enter variables in an arbitrary order (even in an order that was not present in marked like terms) and with arbitrary powers, which give the right result (the right part of Figure 3.2). It means that the student is free to choose whether to enter the monomial in the normal form or to use the same variable several times in different powers. Both positive and negative variable powers are supported.

In special cases, some parts of the result or even the whole result can be left empty – actually, there are similar cases when solving on paper. If the result of combining is 0 (coefficient is 0) and there are other terms in the sum where the initial terms belonged to, then this resulting 0 can be omitted (the box can be left empty). However, if a result is entered then it should be entered either as 0 or 0 with variables in correct powers. If the power of variable is 0 then the whole variable can be omitted. In addition, if the power of variable is 1 then the power can be omitted. If the coefficient of the monomial is 1 or –1 then it can be omitted. The same omitting algorithm is used also in all the other rules where the result of the operation is a single monomial.

After confirming the input, the program performs common checks. As the result should be a single monomial, the program checks whether the entered result is a single monomial. After that, different parts of the entered monomial are compared to the correct ones: sign, coefficient and set of variables with powers. The program also controls, whether new variables were introduced. For example, in case of combining numbers, the student enters the right number

with a variable with power 0. The entered result is equivalent to the right result. In this case T-algebra shows a warning about the new variable introduced, and the student can correct the result or continue with the new variable. If he continues he has to make some unnecessary steps to remove the introduced variable.

During the input of the result, the student can ask for help and the program will put the right answers in the boxes. If monomials being combined have a different order of variables then the program will propose variables in an alphabetic order, only variables with power 0 will be eliminated from the result. T-algebra offers the normalized form of monomial in the result even if the monomials being combined are not in the normalized form. T-algebra also omits as many parts of result as possible (powers 1; coefficients 1, -1; variables with powers 0; whole result if coefficient is 0).

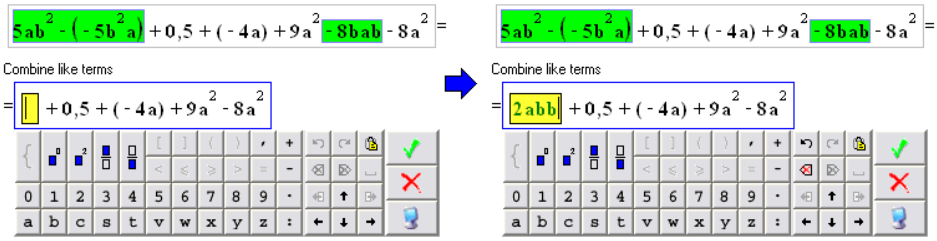


Figure 3.2. Input of result (free input mode) in applying the rule *Combine like terms*

Error messages after input of result (free input mode):

- Result is not entered (empty input but correct result is different from 0 or this is the only member);
- Sign is not entered (part of the result is not entered);
- Error in entering the expression (syntax error, incorrect expression);
- Entered expression is not a monomial (error in the form of result);
- Incorrect sign (sign calculation error – error in certain part of the result);
- Calculation error (coefficient calculation error – error in certain part of the result);
- Incorrect variable part (power calculation error or wrong variable used – error in certain part of the result).

Instruction for input of result (structured input mode): Enter the result of combining (sign before monomial, coefficient, variables and their exponents).

Input of result (structured input mode): In case of the structured input mode, the program offers several boxes (Figure 3.3). As the result of combining like terms should be a monomial, the offered structure has the general structure of a monomial. The first box is sign input box, the next is coefficient input box, followed by boxes for input of variables and exponents. The program offers the

same number of boxes for variables of the monomial as was the number of variables in the marked parts. Variables can be entered in an arbitrary order inside the monomial (the right part of Figure 3.3). However, the program requests the user to standardize the result to some extent, because the number of offered boxes is limited. For example, although the form of one monomial is bab in Figure 1.3, the program offers only two boxes for entering variables in the next line, i.e., the user must standardize the form bab and change it to ab^2 or b^2a.

After confirmation of input, the program performs common checks. If the student asks for help, the program will put the correct monomial in the boxes in the same way as in the free input mode.

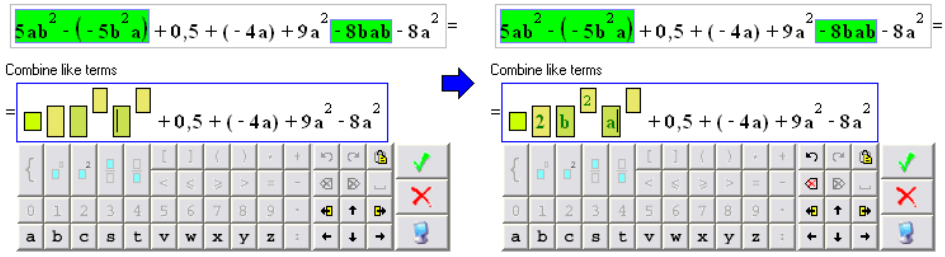


Figure 3.3. Input of result (structured input mode) in applying the rule *Combine like terms*

Error messages after input of result (structured input mode):

- Result is not entered (empty input but correct result is different from 0 or this is the only member);
- Sign is not entered (part of the result is not entered);
- Expression after the sign is needed (part of the result is not entered);
- Error in sign (syntax error, incorrect part of expression);
- Error in number (syntax error, incorrect part of expression);
- Error in variable (syntax error, incorrect part of expression);
- Error in exponent (syntax error, incorrect part of expression);
- Incorrect sign (sign calculation error – error in certain part of the result);
- Calculation error (coefficient calculation error – error in certain part of the result);
- Incorrect variable part (power calculation error or wrong variable used – error in certain part of the result).

Instruction for input of result (partial input mode): Enter the missing parts of the result of combining (sign before monomial, coefficient).

Input of result (partial input mode): In case of the partial input mode, only boxes for input of sign and coefficient are given (Figure 3.4). The variables and

exponents are written by the program. The program simplifies the work of the user also by doing the standardization of the variables of monomial, i.e., converting the monomial into normal form. As part of the result is already entered by the program and cannot be changed in the partial input mode, it is impossible to omit such monomial even if the coefficient is 0. In this case extra solution steps have to be made to eliminate the 0 term.

The correctness of sign and coefficient is checked when the correctness of the step is evaluated.

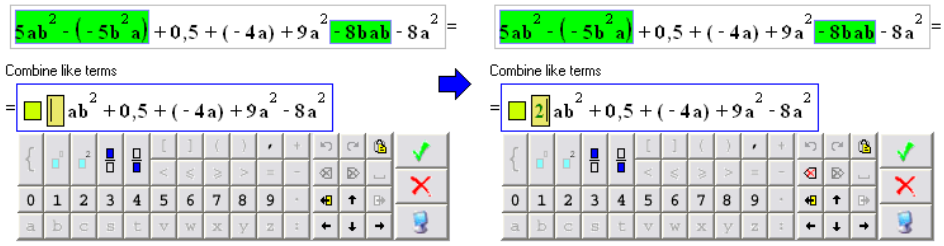


Figure 3.4. Input of result (partial input mode) in applying the rule *Combine like terms*

Error messages after input of result (partial input mode):

- Result is not entered (empty input but correct result is different from 0 or this is the only member);
- Sign is not entered (part of the result is not entered);
- Error in sign (syntax error, incorrect part of expression);
- Error in number (syntax error, incorrect part of expression);
- Incorrect sign (sign calculation error – error in certain part of the result);
- Calculation error (coefficient calculation error – error in certain part of the result).

3.2.6 Rule Multiply/Divide monomials

Applications: multiply monomials; divide monomials; multiply and divide (at the same time) monomials, also multiply and divide numbers.

Instruction for marking: Mark in a product/division the monomials to be multiplied/divided.

Marking: The student has to select monomials from one product to multiply or divide them (Figure 3.5). The student has to select objects precisely – if the product contains polynomials in parentheses then they cannot be selected as objects. Similarly to combining like terms, if a monomial with a negative coefficient is put into parentheses, it should be selected with them to multiply with others (left part of Figure 3.5). At each step, only monomials from one

product can be selected – T-algebra does not allow parallel applications of the rule. At school, students usually multiply and divide terms from left to right but in T-algebra it is possible to multiply or divide in any order. It is also possible to multiply and divide terms simultaneously in one product (right part of Figure 3.5). This makes this rule more similar to the rule *Combine like terms*.

After confirming the selections, T-algebra checks correctness of the objects: whether all selected objects are monomials, whether there are at least two monomials selected and these monomials belong to the same product. If two relevant objects are next to each other, it is possible to select them together. When selecting monomials, it is possible to select them with or without the preceding sign – T-algebra always includes the signs in the selection.

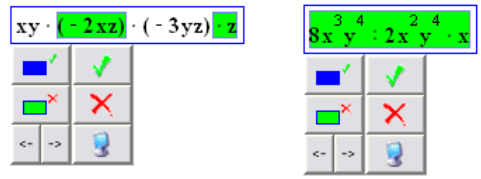


Figure 3.5. Two examples of marking stage in applying the rule *Multiply/Divide monomials*

Instruction for input of result (free input mode): Enter the result of multiplication/division (sign before the result and the monomial).

Input of result (free input mode): In the free input mode, the student gets two boxes for input of the result. The bigger box is intended for input of the monomial without sign. This box is located in the product on the place of the first marked monomial (left part of Figure 3.6). For the sign (plus or minus), there is a smaller box, which is located before the whole product – this is required because, in some cases, there are other terms in the product between the sign and the result (left part of Figure 3.6). Therefore, even if two boxes are located close to each other, they are still two different boxes (right part of Figure 3.6). The sign should be entered in this special box; the program does not accept the answer if the sign is entered in the monomial box.

T-algebra automatically inserts a multiplication sign before the resulting monomial and a correct sign after the resulting monomial before other terms in the product, if any. Therefore, it is not possible to leave the result empty even if the monomial is 0/1. This automatically added multiplication sign can cause problems for the student if he selects two or more divisor monomials and wants to multiply those – in this case, he will have to invert the result. However, students are not taught to simplify expressions in this way and T-algebra does not use it in the automatic solving algorithm, so this should not cause many problems.

As with the rule *Combine like terms*, the student can enter variables in an arbitrary order. In addition, the program does not require that variables should be presented once (i.e., the student can enter zz instead of z^2). Variables with power 0 can be skipped. The whole monomial result can be skipped only if it is 0 and the whole product was selected for multiplication. As the result should be a single monomial, the program performs common checks, which were described before (see section 3.2.1) – the form of the result and then separately each component of the monomial.

If the student asks for help, the program will fill in the boxes with the right sign and the right monomial, where variables are placed in an alphabetic order and only variables with power 0 are moved to the end of the monomial (they will be removed later, anyway, but the program does not delete them from the automatic result).

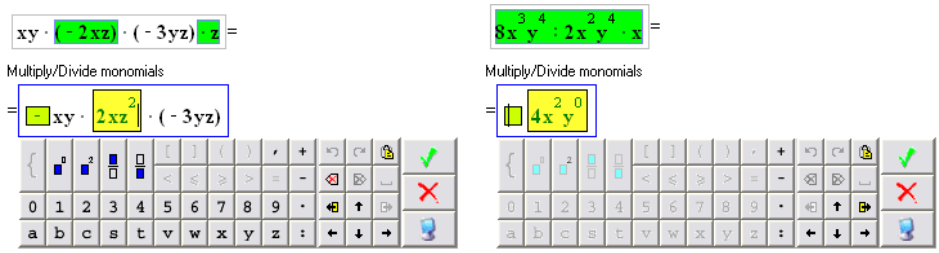


Figure 3.6. Input of result (free input mode) in applying the rule *Multiply/Divide monomials*

Instruction for input of result (structured input mode): Enter the result of multiplication/division (sign before monomial, coefficient, variables and their exponents).

Input of result (structured input mode): In the structured input mode, T-algebra gives the structure of the resulting monomial (Figure 3.7). The proposed structure is very similar to the structured mode of the rule *Combine like terms*. The only difference is that the box for the sign is placed before the whole product (as in the free input mode), not before the resulting monomial. Separate boxes are added for each variable letter from all initial monomials (only one box is added if the same variable exists in two monomials). The checks and help are accomplished in the same way as for the rule *Combine like terms* – sign, coefficient, variables and powers are checked separately.

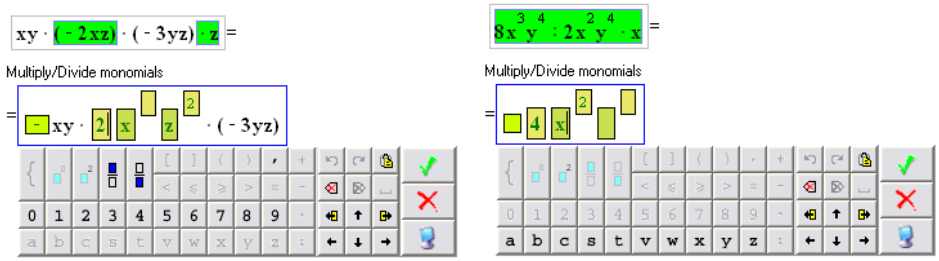


Figure 3.7. Input of result (structured input mode) in applying the rule *Multiply/Divide monomials*

Instruction for input of result (partial input mode): Enter missing parts of the result of multiplication/division (sign before monomial, coefficient, exponents of variables).

Input of result (partial input mode): In the partial input mode, the program proposes boxes for entering the sign (placed before the whole product like in other modes), the coefficient and the powers of variables (Figure 3.8). As T-algebra writes out all variables, which were in the marked parts, the powers 0 should be entered in the result. T-algebra performs common checks at the confirmation of input. There are no specific checks for this mode – the checks of coefficient, sign and the powers of variables are all common checks.

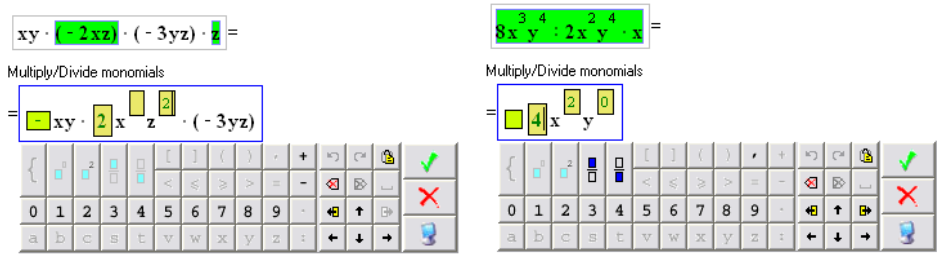


Figure 3.8. Input of result (partial input mode) in applying the rule *Multiply/Divide monomials*

3.2.7 Rule Raise monomial to a power

Applications: exponentiation of monomials; exponentiation of numbers (as a number is also a monomial).

Instruction for marking: Mark a monomial together with exponent for exponentiation.

Marking: For application of this rule, the student has to select exactly one monomial to be raised to a power (Figure 3.9). The student should mark both the base and the power (exponent). If the base is written in parentheses (it is in most cases, only single positive numbers can be without parentheses), the parentheses should be marked as well. Only one monomial can be raised to a power during one step. A monomial can consist only of one number (then it is exponentiation of the number), but cannot consist only of one variable, i.e., x^2 is not suitable for this rule because the result of this operation would be exactly the same. At the confirmation of marking, the program checks all the above mentioned attributes.

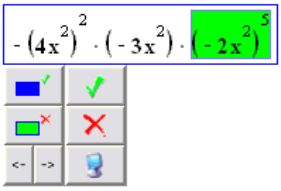


Figure 3.9. Marking stage in applying the rule *Raise monomial to a power*

Instruction for input of result (free input mode): Enter the sign and the resulting monomial.

Input of result (free input mode): As in the case of the rule *Multiply/Divide monomials*, two boxes are given for the student for input (Figure 3.10). The first box (placed before the whole product) is for input of the sign; the second box (placed on place of the marked monomial, may be separated from the other input box by some terms) is for input of the resulting monomial. In detection of the sign, the sign before the whole product should be taken into account (even though it might not have been selected). In the resulting monomial, the coefficient should be only a number (i.e., the number should be raised to a power and the answer should be calculated). For example, in Figure 3.10, the student cannot enter 2^5 into the result. After input is confirmed, the common checks used for monomials are performed (see section 3.2.1).

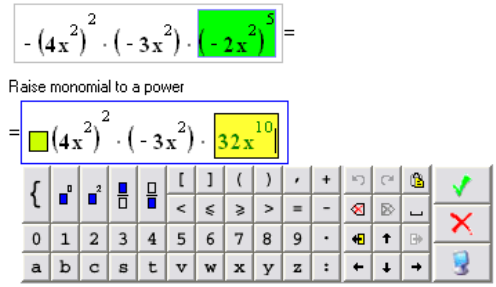


Figure 3.10. Input of result (free input mode) in applying the rule *Raise monomial to a power*

Instruction for input of result (structured input mode): Enter the sign and parts of the resulting monomial.

Input of result (structured input mode): The proposed boxes (Figure 3.11) and the checks after confirmation are the same as in case of the rule *Multiply/ Divide monomials*.

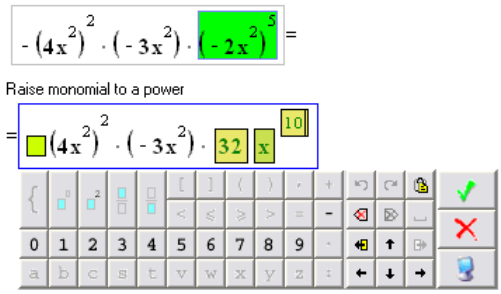


Figure 3.11. Input of result (structured input mode) in applying the rule *Raise monomial to a power*

Instruction for input of result (partial input mode): Enter the sign and parts of the resulting monomial.

Input of result (partial input mode): The partial input mode of this rule (Figure 3.12) is analogous to partial input mode of the rule *Multiply/Divide monomials*.

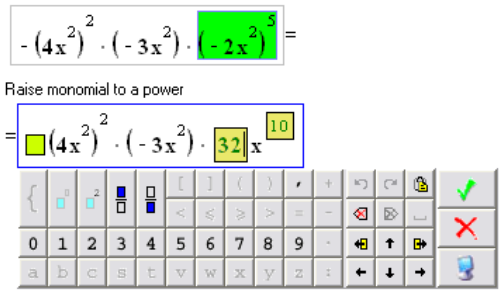


Figure 3.12. Input of result (partial input mode) in applying the rule *Raise monomial to a power*

3.2.8 Rule Clear parentheses

Applications: clear parentheses from the polynomial in the sum expression; clear parentheses from a single monomial in a sum (required in case of a negative coefficient); clear parentheses from the monomial in the product; clear parentheses from a complex expression (for example, a sum of monomials, products and other complex expressions), which is a part of the sum expression.

Instruction for marking: Mark one expression with parentheses for removing the parentheses.

Marking: The student has to mark an expression with parentheses (Figure 3.13). The parentheses should always be included in the marking. Only one pair of parentheses can be removed at one step. Priority of the main operation of expression inside the parentheses should be equal to or higher than the priority of the operation outside. For example, it is possible to remove parentheses from a polynomial if the operation outside the parentheses is addition or subtraction, and it is impossible if the operation outside is multiplication. This rule cannot be applied to multiplying a number or a monomial by a polynomial in parentheses; for this purpose, the rule *Multiply/Divide polynomial by monomial* (section 3.2.9) or the rule *Open parentheses* should be selected.

After confirming the selections, T-algebra checks correctness of the objects: only one expression with parentheses can be selected at one step. In addition, the program checks whether the student knows the priorities of operations. During the selection of objects, the student can ask for help and the program selects the first possible expression with parentheses or displays a message that there are no suitable parentheses.

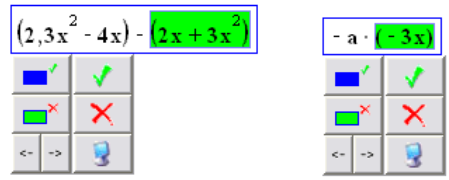


Figure 3.13. Two examples of marking stage in applying the rule *Clear parentheses*

Instruction for input of result (free input mode, structured input mode and partial input mode): Enter pluses and minuses.

Input of result (free input mode, structured input mode and partial input mode): In the current implementation of the rule, all three input modes are implemented exactly the same way (essentially, the partial input mode), because we are unable to offer any certain structure for the resulting expression as we did not want to put any restrictions on the form of the expression in parentheses. For example, it is possible to clear parentheses from a fraction in parentheses when it is a part of a sum or a product. Therefore, we could not prescribe a fixed structure for all cases.

After the marking stage is completed, T-algebra copies the unchanged parts of the expression to the next line and offers boxes for inputting the important parts of the resulting expression (Figure 3.14). The student has to fill in the yellow boxes with correct signs (pluses or minuses). There are two principal cases when this rule is used: to remove parentheses from a polynomial in a sum (left part of the figure; in this case, the sign before each monomial should be entered) or to remove parentheses from a single term (in a sum or in a product; right part of the figure; in this case, a sign should be entered before the term or the whole product accordingly).

The most common mistake that is made during application of this rule is not changing the sign of the monomial when there is a minus before the parentheses. Sign is the most important part in applying this rule and, in these modes, it is the only part of the result that is left for the student to enter. After confirming the input, the program checks correctness of all signs separately.

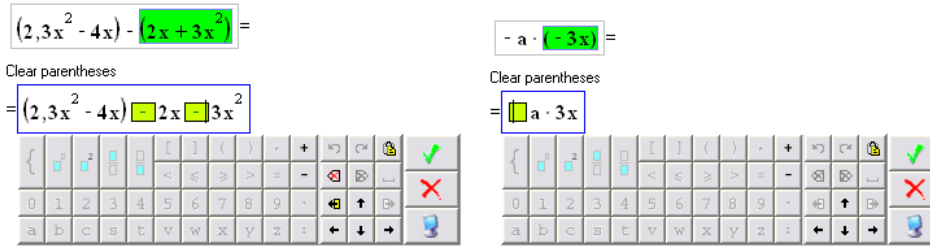


Figure 3.14. Input of result (in both structured and partial input modes) in applying the rule *Clear parentheses*

3.2.9 Rule Multiply/Divide polynomial by monomial

Applications: multiply a polynomial by monomial(s); divide a polynomial by monomial(s); multiply and divide (simultaneously) a polynomial by monomial(s); also multiply and divide a polynomial by numbers (single numbers are monomials as well).

Instruction for marking: Mark one polynomial and monomial(s) for multiplication/division.

Marking: In order to apply this rule, the student has to select exactly one polynomial and one or more monomials from one product to multiply or divide them (Figure 3.15). At each step, only objects from one product can be selected – T-algebra does not allow parallel applications of the rule. The student can decide how many monomials from one product he will use for polynomial multiplication/division in one step (either one by one or all together). At school, students usually multiply and divide terms from left to right but T-algebra enables to multiply or divide in any order. A polynomial should be marked together with parentheses. If a monomial has been put into parentheses for some reason (for example, it has negative coefficient), it has to be selected together with parentheses as well.

After confirming the selections, T-algebra checks correctness of the objects: whether the marked objects include one polynomial and all other objects are monomials, whether there are at least two objects marked and these objects belong to the same product.

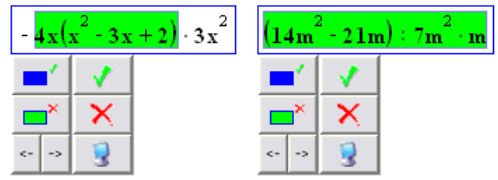


Figure 3.15. Two examples of marking stage in applying the rule *Multiply/Divide polynomial by monomial*

Error messages after marking:

- One polynomial and monomial(s) from the same product should be selected (empty selection);
- In addition, one polynomial should be selected (only one object selected);
- In addition, one monomial should be selected (only one object selected);
- Sign cannot be selected without the following term (object syntax error);
- Selected term is neither monomial nor polynomial (form of the object);
- Only members of one product/division should be selected (position of objects, same level in the tree);

- Mark the polynomial together with parentheses (position of objects, same level in the tree);
- For this rule, only one polynomial should be selected (number of terms).

Instruction for input of result (free input mode): Enter the result of multiplication/division.

Input of result (free input mode): In the free input mode, the whole resulting polynomial should be entered in a single yellow box (Figure 3.16). The student has to add parentheses himself if the polynomial is a part of a product (left part of Figure 3.16). After the input is confirmed, the program performs common checks used for polynomials as described above. Then it checks whether parentheses were added, if required. From the set of checks to be performed (only the equivalence is checked) it is clear that, in the free input mode, the student can multiply/divide a polynomial by monomial(s), but he can also only write out the right multiplication/division as shown on the right side of Figure 3.16 (monomials are multiplied, but the division of the polynomial by the resulting monomial is only written out). Another possibility for the student is to simplify the resulting polynomial, for example, by combining like terms.

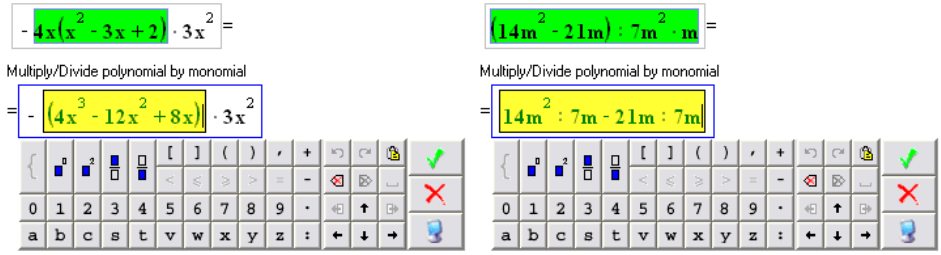


Figure 3.16. Two examples of input of result (free input mode) in applying the rule *Multiply/Divide polynomial by monomial*

Error messages after input of result (free input mode):

- Error in expression (syntactical error);
- Parentheses are needed (order of operations);
- Plus or minus sign is missing (special check);
- Error in answer (resulting subexpression is not equal to correct expression – equality check is performed using the automatic simplification algorithm, see section 2.8.1).

Instruction for input of result (structured input mode): Create the necessary number of boxes for monomials and enter the result.

Adding input boxes to the input structure: The resulting expression of this operation is a polynomial. The number of monomials in the polynomial is an essential attribute and we do not want to predict it by providing the complete

structure of the result. Therefore, in the structured input mode, we added the possibility to construct the result by adding new monomials to it. Initially in this additional stage, only the structure of the first monomial (one box for sign and one box for coefficient with variables) is given (left part of Figure 3.17) – the student has to add more structures by pressing a corresponding button on the virtual keyboard (right part of Figure 3.17). He can also remove added monomials if needed by pressing another button on the virtual keyboard. If the teacher has permitted it, the student can ask the program for help and T-algebra will create the correct number of structures that corresponds to the number of monomials in the correct result. The program controls whether the correct number of monomials was added after the input is confirmed. If less than the correct number of terms was added then the user is given an error message when confirming the input. Monomials with coefficient 0 can be omitted; therefore, no extra box has to be added for this term. The user can add more than the correct number of terms and leave some of them empty – empty boxes will be cleared in the same way as in other rules where some parts may be omitted. The student is not allowed to combine like terms in the result (if the initial polynomial contains like terms); all like monomials should be multiplied/divided separately.

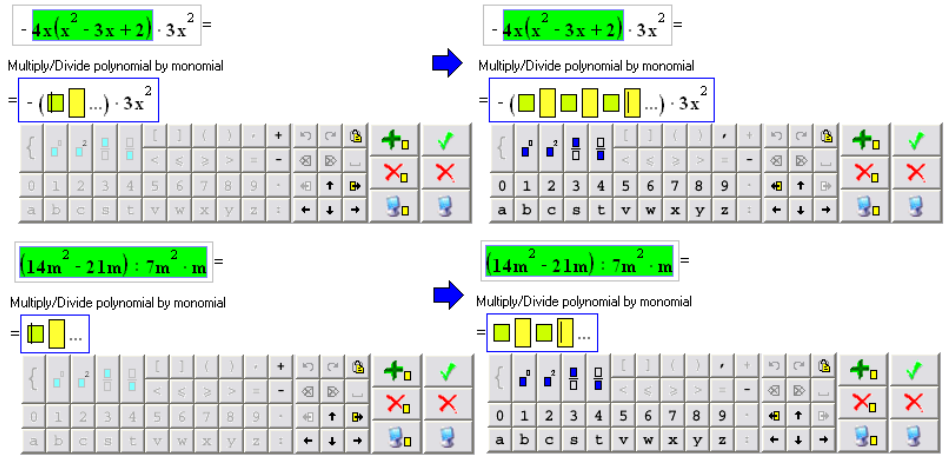


Figure 3.17. Two examples of adding terms to result in applying the rule *Multiply/ Divide polynomial by monomial*

Input of result (structured input mode): After adding the boxes to the result, the student has to fill them with parts of the resulting polynomial. If required, new boxes can be added during input stage. For each monomial, there are two kinds of boxes: small boxes are for input of plus and minus signs, larger boxes

are for entering numbers and variables (Figure 3.18). Terms can be placed in the boxes in an arbitrary order. As with other rules, the variables with power 0 can be skipped. Similarly, whole monomials with coefficient 0 can be skipped.

In the structured input mode, if the resulting polynomial is a part of product, it is put into parentheses (parentheses are added automatically by T-algebra) – it should be easier for the student if, for example, the sign before the selected product is a minus (left part of Figure 3.18) and the student does not have to change the signs of individual monomials at the same time. However, when the resulting polynomial is not a part of the product, even if there is a minus sign before the initial product, the student has to take that minus sign into account and change the signs when multiplying.

In the structured input mode, T-algebra requires from the student exact application of this rule only – no combining of like terms or other simplifications are allowed at this step. The program performs common checks. The form of every entered part should be a monomial and this is verified by performing a rule specific check (form of each part).

When comparing two polynomials (the correct one calculated by T-algebra and the one entered by the student), they are compared as two collections of monomials. For each monomial from the student's polynomial, T-algebra tries to find the corresponding monomial in the correct result. If an identical monomial is not found, the program tries to identify the error: either the sign is incorrect (an identical monomial with the opposite sign), coefficient is incorrect (a similar monomial is found), or powers of variables/variables themselves are incorrect (other cases).

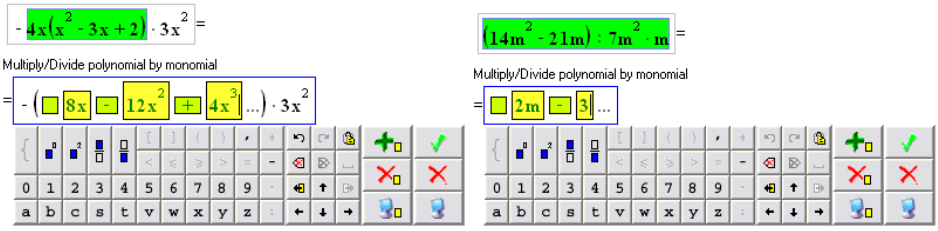


Figure 3.18. Two examples of input of result (structured input mode) in applying the rule *Multiply/Divide polynomial by monomial*

Error messages after input of result (structured input mode):

- Result should contain more terms (number of terms added and filled);
- Result should contain less terms (number of terms added and filled);
- Result cannot be empty (empty result);
- Monomial should be entered (empty monomial box with filled sign box);
- Sign is missing (empty sign box with filled monomial);

- Error in sign (syntax error);
- Error in expression (syntax error);
- There is a separate box for the sign (form of result);
- Incorrect sign (sign calculation error);
- Incorrect coefficient (calculation error);
- Incorrect variable part. Correct result does not contain such member (calculation error – power or variable letter);
- Result should not contain such monomial (calculation error, other error in monomial, completely different from the correct ones).

Instruction for input of result (partial input mode): Enter missing parts of the result.

Input of result (partial input mode): In the partial input mode, the student has to fill only gaps – coefficients, signs and powers of variables of the resulting monomials (Figure 3.19). The order of terms can be changed only if the variable part (letters) is the same as variables are already filled in the result. If the resulting polynomial is a part of product then it is put into parentheses. The checks in the partial input mode are the same as in the structured mode, except the check of the number of monomials – in the partial input mode, the number of monomials is already correct by design. As T-algebra writes out all variables that were in the marked parts, the powers 0 should be entered in the result. T-algebra writes out the variables for all possible terms and, therefore, it is impossible to omit monomials with coefficients 0 (an extra step could be made later to get rid of 0 monomials, but it should not be an issue, because monomials with 0 coefficients are usually removed from the polynomial first).

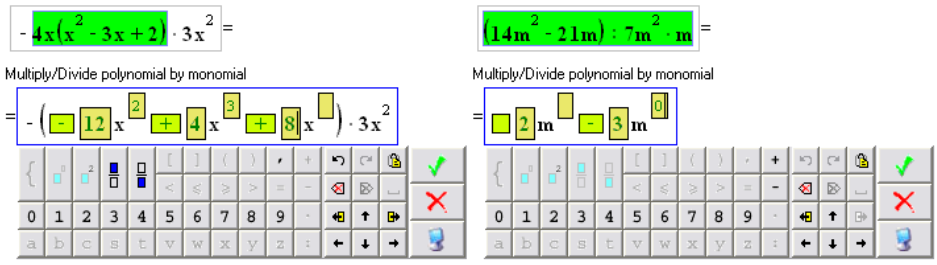


Figure 3.19. Two examples of input of result (partial input mode) in applying the rule *Multiply/Divide polynomial by monomial*

Error messages after input of result (partial input mode):

- Result cannot be empty (empty result);
- Result should contain more terms (number of terms added and filled);
- Monomial should be entered (empty monomial box with filled sign box);
- Sign is missing (empty sign box with filled monomial box);

- Error in sign (syntax error);
- Error in expression (syntax error);
- Coefficient should be a number (form of result);
- Exponent should be an integer (form of result);
- Incorrect sign (sign calculation error);
- Incorrect coefficient (calculation error);
- Incorrect variable part. Correct result does not contain such member (calculation error – power or variable letter);
- Result should not contain such monomial (calculation error, other error in monomial, completely different from the correct ones).

3.2.10 Rule Multiply polynomials

Applications: multiply polynomials.

Instruction for marking: Mark two polynomials (together with parentheses) for multiplication.

Marking: After selection of the rule, the student has to select exactly two polynomials from one product to multiply (Figure 3.20). At each step, only the objects from one product can be selected (no parallel rule applications are allowed). Polynomials should be marked together with parentheses. During the selection of objects, the student can ask for help as in all other rules.

At the confirmation of marking, T-algebra checks correctness of the objects: whether the selected objects are polynomials, whether exactly two objects were selected and these objects belong to the same product.

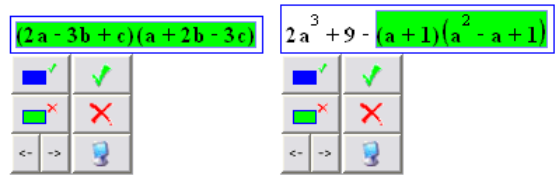


Figure 3.20. Two examples of marking stage in applying the rule *Multiply polynomials*

Instruction for input of result (free input mode): Enter the result of multiplication.

Input of result (free input mode): The design of the input is exactly the same as in the rule *Multiply/Divide polynomial by monomial*. In the free input mode, the whole polynomial should be entered in a single yellow box (Figure 3.21). The student has to add parentheses himself in case the polynomial is a part of a product. After confirming the input, T-algebra checks equivalence of the entered resulting subexpression to the correct polynomial (subexpression) and also whether parentheses were added if required (order of operations). As

with the rule *Multiply/Divide polynomial by monomial*, the student is left a possibility to do more or less than just multiply the polynomials. For example, he can combine like terms (left part of Figure 3.21) or simply write out the correct multiplication (right part of Figure 3.21).

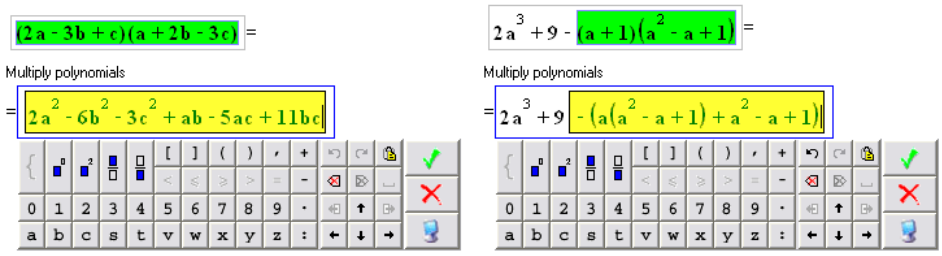


Figure 3.21. Two examples of input of result (free input mode) in applying the rule *Multiply polynomials*

Instruction for input of result (structured input mode): Create the necessary number of boxes for monomials and enter the result.

Adding input boxes to the input structure: Implementation of the adding stage is similar to the adding stage of the rule *Multiply/Divide polynomial by monomial*. If the marking was correct then the structure of the first monomial (one box for a sign and one box for a coefficient with variables) is given in the position of the first (leftmost) selected polynomial (left part of Figure 3.22). The student has to add more monomials using the appropriate button (right part of Figure 3.22).

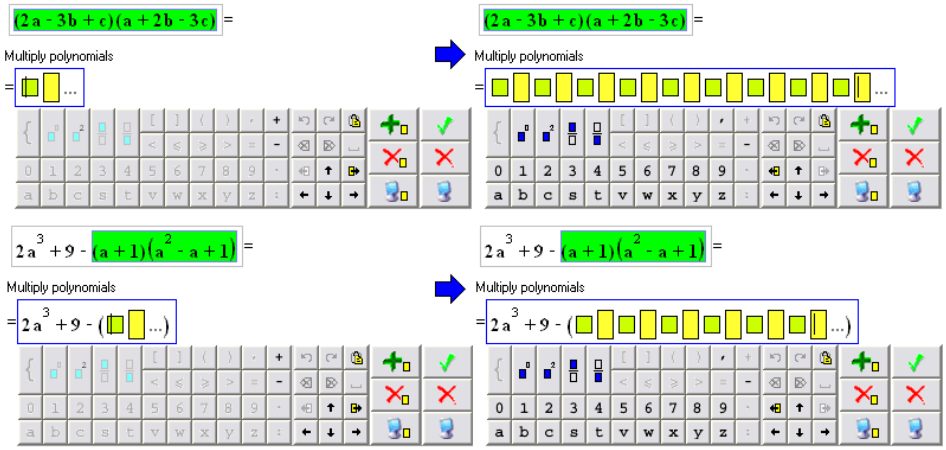


Figure 3.22. Two examples of adding terms to result in applying the rule *Multiply polynomials*

Input of result (structured input mode): The structured input mode is also implemented similarly as in the rule *Multiply/Divide polynomial by monomial*. The student has to enter signs and monomials (Figure 3.23). Parentheses are added by the program if the result is a part of a product or a sum. The program insists on exact application of the rule. Monomials can be entered in an arbitrary order (for example, in the upper part of Figure 3.23, the first monomial of the first polynomial is multiplied by all terms of the second polynomial, then the second monomial of the first polynomial is multiplied by all terms of the second polynomial and so on; the lower part of Figure 3.23 illustrates the other possible manner of multiplication: all terms of the first polynomial are multiplied by the first monomial of the second polynomial, then all terms of the first polynomial are multiplied by the second monomial of the second polynomial and so on). If the student asks for help then the program fills the boxes with the result, found by the same manner of multiplication as shown in the upper part of Figure 1.24. The checks are the same as the checks used for the rule *Multiply/Divide polynomial by monomial*.

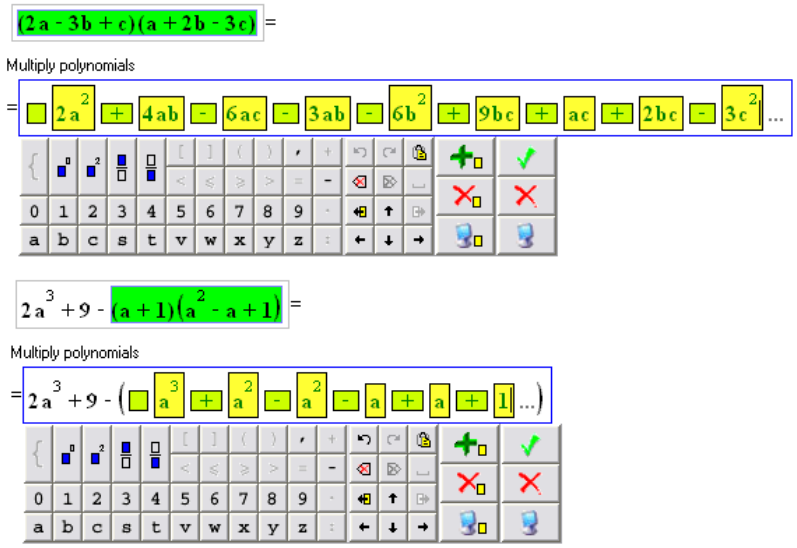


Figure 3.23. Two examples of input of result (structured input mode) in applying the rule *Multiply polynomials*

Instruction for input of result (partial input mode): Enter missing parts of the result.

Input of result (partial input mode): The partial input mode (boxes, checks) (Figure 3.24) is accomplished in the same way as in the rule *Multiply/ Divide polynomial by monomial*.

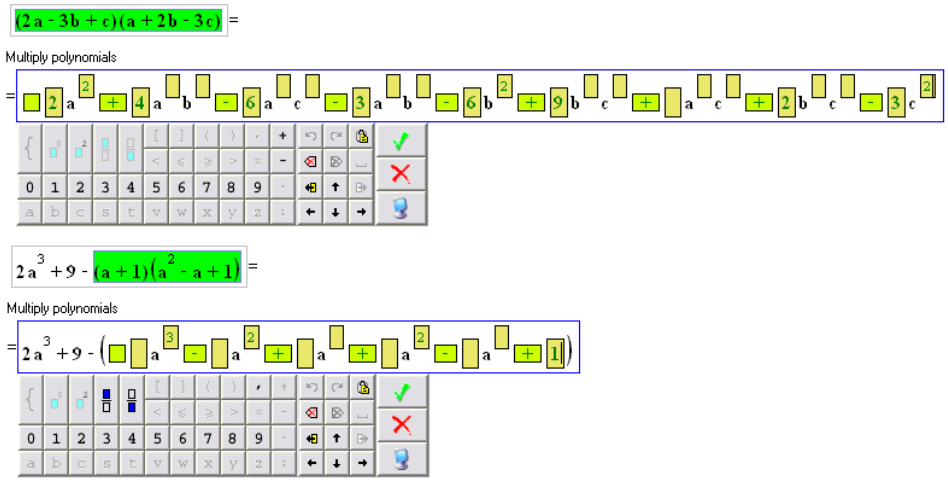


Figure 3.24. Two examples of input of result (partial input mode) in applying the rule *Multiply polynomials*

3.2.11 Rule (a±b)² →

Applications: expand the square of sum/difference of two monomials.

Instruction for marking: Mark a square of a binomial (together with exponent) for raising to a power.

Marking: For application of this rule, the student has to select exactly one sum or difference of two monomials, which he wants to square, together with exponent (number *2*) (Figure 3.25). The sum/difference should be marked together with parentheses. On confirmation of the marking, the program checks whether the selected parts are appropriate: whether exactly one sum/difference is selected; whether this sum/difference consists of exactly two monomials; whether the exponent is *2*; whether the sum/difference is selected together with parentheses.

Figure 3.25. Marking stage in applying the rule *(a±b)²* →

Instruction for input of result (free input mode): Enter the result of exponentiation.

Input of result (free input mode): In the free input mode, the whole result should be entered in a single yellow box (Figure 3.26). As in other rules, where the result of multiplication is entered in one box and the multiplied result is a

polynomial, T-algebra allows partial application of the rule. Dealing with parentheses, if needed, is the student's task.

The resulting polynomial is subjected to all common checks when the input is confirmed. The existence of parentheses is checked as well, if required. If the student asks the program for help, T-algebra fills the box with the squared result and adds parentheses, if required.

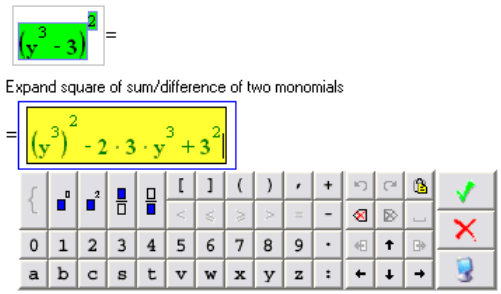


Figure 3.26. Input of result (free input mode) in applying the rule *(a±b)²* →

Instruction for input of result (structured input mode): Create the necessary number of boxes for monomials and enter the result.

Adding input boxes to the input structure: As in other rules, where the result is a polynomial, the student should, in the structured input mode, create the correct number of boxes (initially, only one monomial box is given)(Figure 3.27).

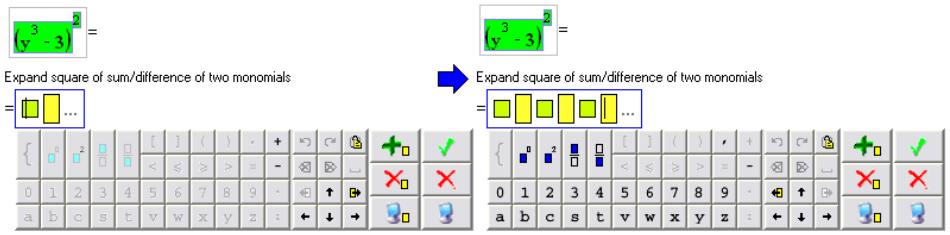


Figure 3.27. Adding terms to result in applying the rule *(a±b)²* →

Input of result (structured input mode): After adding the input boxes, the student has to enter the result in these boxes (Figure 3.28). As in other rules, where the result is a polynomial, the program insists on exact application of the rule. The input (arbitrary order of terms), parentheses (if the result is part of a sum or product) and checks are handled as in the rules *Multiply/Divide polynomial by monomial* and *Multiply polynomials*.

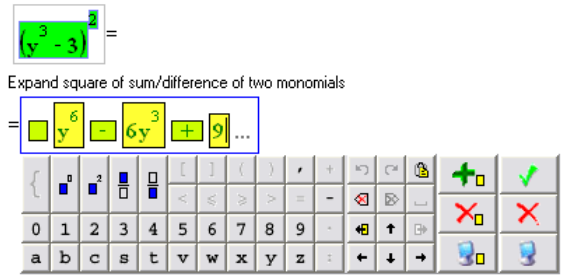


Figure 3.28. Input of result (structured input mode) in applying the rule *(a±b)²* →

Instruction for input of result (partial input mode): Enter missing parts of the result.

Input of result (partial input mode): As in other rules, the partial input mode here requires the student only to fill gaps (Figure 3.29). The result is put in parentheses by the program, if needed. Common checks are performed when the input is confirmed.

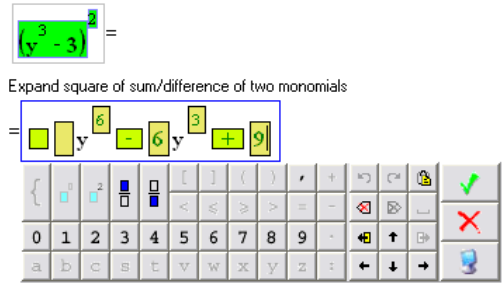


Figure 3.29. Input of result (partial input mode) in applying the rule *(a±b)²* →

3.2.12 Rule (a±b)³ →

Applications: expand the cube of sum/difference of two monomials.

Instruction for marking: Mark a cube of a binomial (together with exponent) for raising to a power.

Marking: The marking stage (Figure 3.30) is identical to the marking stage of the rule *(a±b)²* → (the only difference is that the exponent should be number *3*).

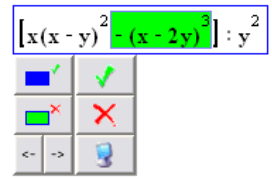


Figure 3.30. Marking stage in applying the rule *(a±b)³* →

Instruction for input of result (free input mode): Enter the result of exponentiation.

Input of result (free input mode): The free input mode (Figure 3.31) is identical to the free input mode of the rule *(a±b)²* → (except, of course, that the result should contain raising to the cube).

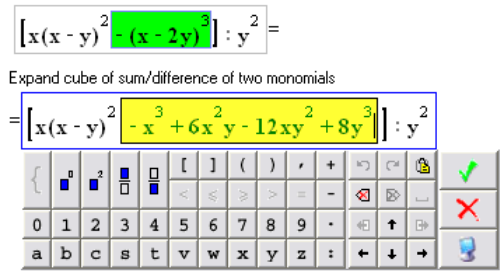


Figure 3.31 Input of result (free input mode) in applying the rule *(a±b)³* →

Instruction for input of result (structured input mode): Create the necessary number of boxes for monomials and enter the result.

Adding input boxes to the input structure: The adding stage (Figure 3.32) is carried out in the same way as during application of the rule *(a±b)²* →.

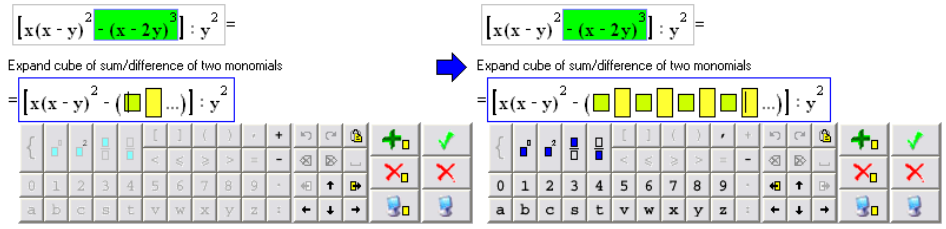


Figure 3.32. Adding terms to result in applying the rule *(a±b)³* →

Input of result (structured input mode): The structured input mode (Figure 3.33) is also analogous to the structured input for the rule *(a±b)²* →.

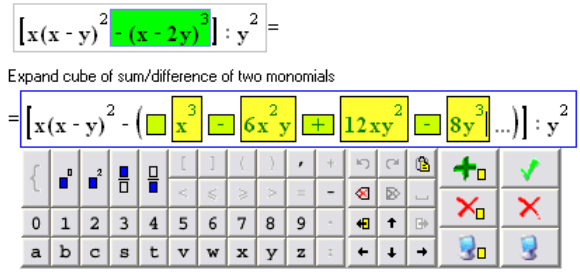


Figure 3.33. Input of result (structured input mode) in applying the rule *(a±b)³* →

Instruction for input of result (partial input mode): Enter missing parts of the result.

Input of result (partial input mode): The partial input mode (Figure 3.34) is similar to the partial input for the rule *(a±b)²* →.

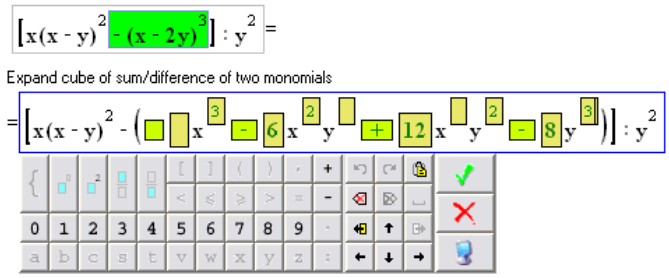


Figure 3.34. Input of result (partial input mode) in applying the rule *(a±b)³* →

3.2.13 Rule (a+b)(a-b) →

Applications: multiply the sum and difference of two monomials.

Instruction for marking: Mark a product of the sum and difference of two monomials to be multiplied.

Marking: For application of this rule, the student has to select the product of the sum and difference of two monomials (Figure 3.35). The monomials inside the sum and difference can be in an arbitrary order and the sum and difference of these monomials can be in an arbitrary order inside the product (i.e., the program allows to apply this formula, for example, to *(-b+a)(a+b)* and does not insist on transforming it to *(a+b)(a-b)* beforehand). The sum and difference should be marked together with parentheses. Of course, the sum and difference should belong to one product.

On confirmation of the marking, the program checks whether the marked parts are appropriate: whether exactly one pair (the sum and difference of the same monomials) is marked; whether the sum and difference belong to one product; whether the sum and difference consist of exactly two monomials; whether these monomials are the same in the sum and difference; whether the sum/difference is selected together with parentheses.

Figure 3.35. Marking stage in applying the rule *(a+b)(a-b)* →

Instruction for input of result (free input mode): Enter the result of multiplication.

Input of result (free input mode): In the free input mode, the student is given one box for entering the result (Figure 3.36). Like in other rules, where the result is a polynomial, parentheses are the responsibility of the student. The program performs common checks when the input is confirmed. The parentheses are checked as well, if needed. In the free mode, the program does not require exact application of the formula, as only equivalence is checked. The student can proceed as if applying the rule *Multiply polynomials*: multiply the polynomials (Figure 3.36) or just write out the correct multiplication.

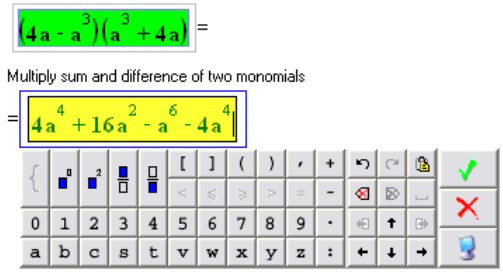


Figure 3.36. Input of result (free input mode) in applying the rule *(a+b)(a-b)* →

Instruction for input of result (structured input mode): Create the necessary number of boxes for monomials and enter the result.

Adding input boxes to the input structure: For entering result in the structured input mode, the student has to add the necessary number of boxes for monomials (Figure 3.37). The program initially gives boxes for only one monomial. The number of boxes is checked after the input is confirmed.

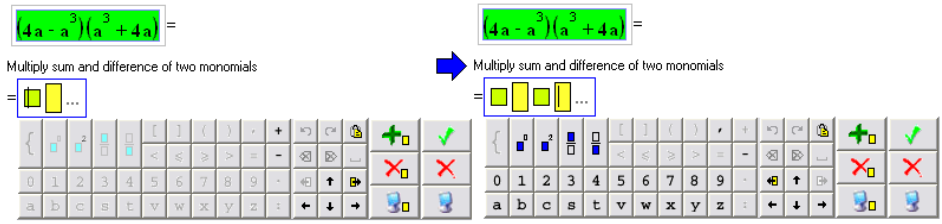


Figure 3.37. Adding terms to result in applying the rule *(a+b)(a-b)* →

Input of result (structured input mode): After input boxes have been added, the student should enter the result of application of this rule in these boxes (Figure 3.38). In the structured input mode, the program allows only exact application of the formula, i.e., the result should be the difference of squared monomials. Parentheses are added by the program, if needed. Monomials can be entered in an arbitrary order.

After the input is confirmed, the program performs common checks. In addition, during rule-specific check (the form of every part), the program checks whether the monomials are entered.

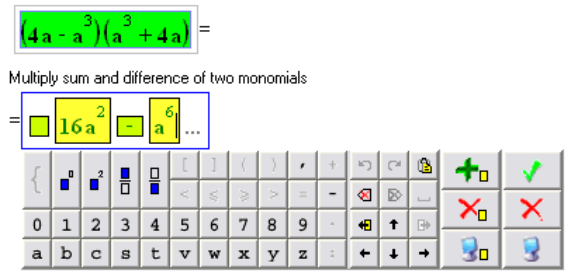


Figure 3.38. Input of result (structured input mode) in applying the rule *(a+b)(a-b)* →

Instruction for input of result (partial input mode): Enter missing parts of the result.

Input of result (partial input mode): In the partial input mode, the program pre-fills some parts (variables) and the student should enter the rest (signs,

coefficients and powers of variables) (Figure 3.39). Only exact application of the formula is allowed. The parentheses around the resulting binomial are added by the program if a polynomial is a part of the sum or product. After confirmation, the common checks are performed.

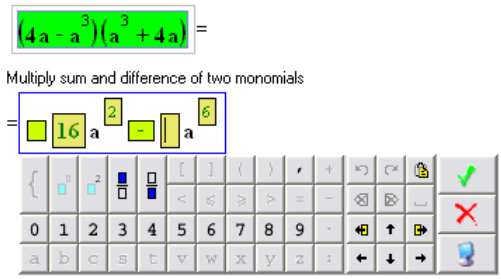


Figure 3.39. Input of result (partial input mode) in applying the rule *(a+b)(a-b)* →

3.2.14 Rule (a±b)(a²±ab+b²) →

Applications: multiply the sum/difference of two monomials by incomplete square of the difference/sum.

Instruction for marking: Mark a sum/difference of two monomials and an incomplete square of the difference/sum of the same monomials.

Marking: Marking (Figure 3.40) is accomplished in the same way as for the rule *(a+b)(a-b)* →.

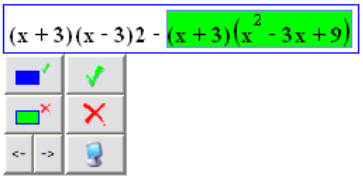


Figure 3.40. Marking stage in applying the rule *(a±b)(a²±ab+b²)* →

Instruction for input of result (free input mode): Enter the result of multiplication.

Input of result (free input mode): The free input mode (Figure 3.41) is identical to the free input mode of the rule *(a+b)(a-b)* →.

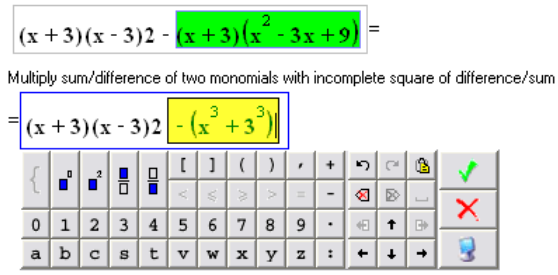


Figure 3.41. Input of result (free input mode) in applying the rule *(a±b)(a²±ab+b²)* →

Instruction for input of result (structured input mode): Create the necessary number of boxes for monomials and enter the result.

Adding input boxes to the input structure: Adding of terms (Figure 3.42) is performed in the same way as during application of the rule *(a+b)(a-b)* →.

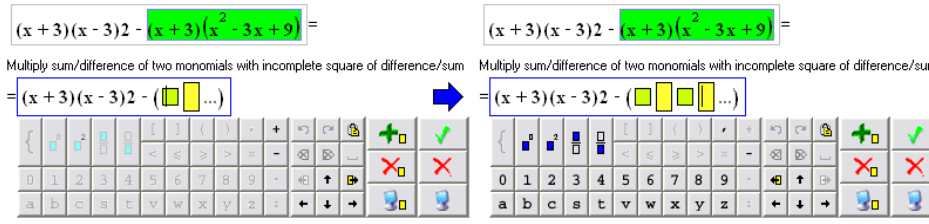


Figure 3.42. Adding terms to result in applying the rule *(a±b)(a²±ab+b²)* →

Input of result (structured input mode): Implementation of the structured input mode (Figure 3.43) is similar to the structured input for the rule *(a+b)(a-b)* →.

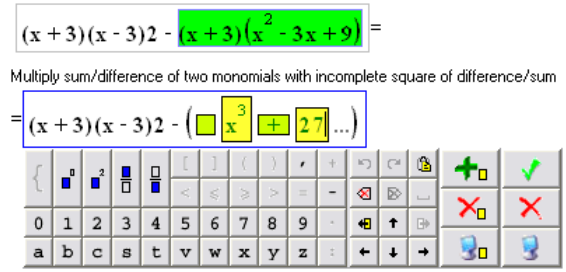


Figure 3.43. Input of result (structured input mode) in applying the rule *(a±b)(a²±ab+b²)* →

Instruction for input of result (partial input mode): Enter missing parts of the result.

Input of result (partial input mode): The partial input mode (Figure 3.44) functions similarly to the partial input mode of the rule *(a+b)(a-b)* →.

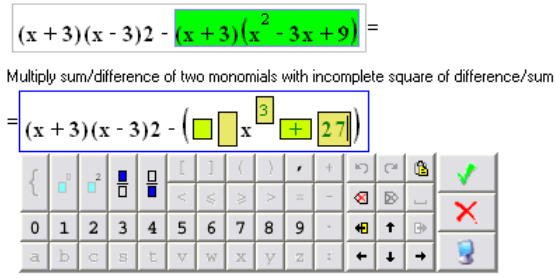


Figure 3.44. Input of result (partial input mode) in applying the rule *(a±b)(a²±ab+b²)* →

3.2.15 Rule Multiply/Divide terms with the same base

Applications: multiply powers with the same base; divide powers with the same base; multiply and divide (at the same time) powers with the same base.

Instruction for marking: Mark at least two powers with the same base to be multiplied/divided.

Marking: For application of this rule, the student has to mark at least two terms with exponents, which have the same base (Figure 3.45). The base can be a number (right part of Figure 3.45), a variable (left part of Figure 3.45), a monomial, or a more complex expression in parentheses. Terms should be marked together with exponents. If the base is written in parentheses, it should be selected together with parentheses. On confirmation of the marking, the program checks whether the selected terms are appropriate: whether at least two terms are marked; whether terms are marked together with exponents; whether the terms belong to one product; whether the terms have equivalent bases (for example, the rule accepts terms with bases $(b+3)$, $(3+b)$, $(b+2+1)$, etc.).

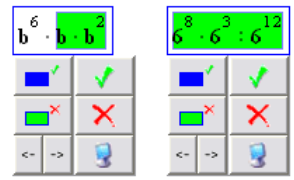


Figure 3.45. Two examples of marking stage in applying the rule *Multiply/Divide terms with the same base*

Instruction for input of result (free input mode): Enter the result of multiplication/division.

Input of result (free input mode): In the free input mode, the student has to enter the whole result in one box (Figure 3.46). Entering the correct sign before the result (multiplication/division sign in a product (left part of Figure 3.46), addition or subtraction sign in a sum) is also the student's task. During application of this rule, the student can only add exponents; no further simplification (like raising a number to a power) is allowed. The resulting power can also be presented as a sum expression (for example, b^{1+2}). The program separately checks equivalence of the entered base to the correct base and correctness of the power, as it would in case of a monomial (the only difference being that a base can be more complex than a single variable).

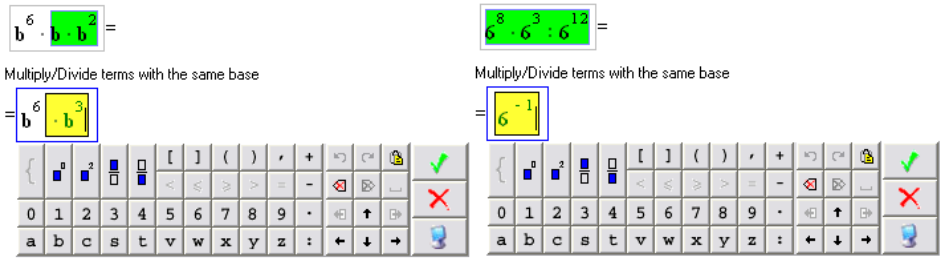


Figure 3.46. Two examples of input of result (free input mode) in applying the rule *Multiply/Divide terms with the same base*

Instruction for input of result (structured input mode): Enter parts of the result.

Input of result (structured input mode): In the structured input mode, the program proposes the structure of the result by giving three boxes (Figure 3.47). The first smaller box is for entering the sign (addition, subtraction, multiplication or division, depending on context). The second box is intended for input of the base. The third box is designed for exponent. The content of these boxes is checked on confirmation of the input. The power is checked together with the sign and T-algebra allows entering the result with either a multiplication or a division sign if the power is inverted accordingly ($*b^3$ or $:b^{-3}$). The content of the base box has to be equivalent to the base of the selected terms.

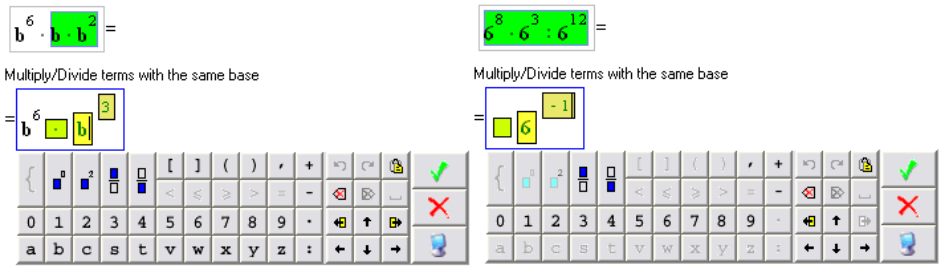


Figure 3.47. Two examples of input of result (structured input mode) in applying the rule *Multiply/Divide terms with the same base*

Instruction for input of result (partial input mode): Enter exponent in the result.

Input of result (partial input mode): In the partial input mode, only the box for exponent is given (Figure 3.48) and only the exponent is checked on confirmation of the input. T-algebra always uses a multiplication sign even if it leads to negative exponents.

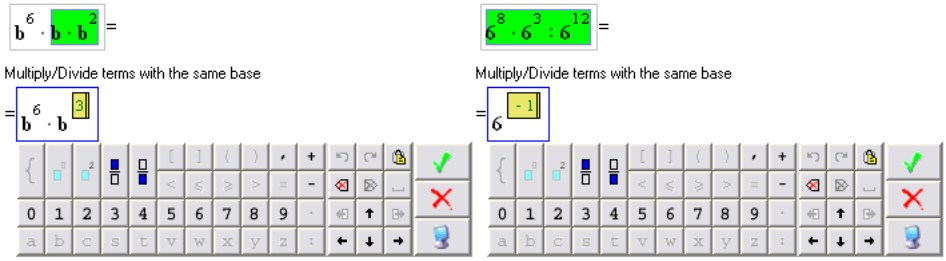


Figure 3.48. Two examples of input of result (partial input mode) in applying the rule *Multiply/Divide terms with the same base*

3.2.16 Rule Raise product/quotient/power to a power

Applications: raise a product to a power; raise a quotient to a power; raise a power to a power, raise a complex expression containing products, quotients, fractions and powers to a power.

Constraints for expression: the expression may not contain a fraction with the same variable in the numerator and denominator, because the implemented fraction reduction rule only works with numbers.

Instruction for marking: Mark a power of product/quotient/power (together with parentheses and exponent), to be raised to a power.

Marking: After selecting this rule, the student has to mark exactly one exponent expression to raise it to a power (Figure 3.49). The exponent base can be a product (of variables, powers, more complex expressions), a quotient or a fraction (of expressions of the same kind) or a power (of expressions of the same kind). The base expression should be marked together with parentheses. After confirmation of marking, the program checks whether exactly one suitable expression is selected.

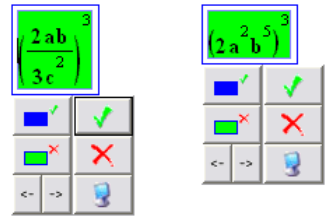


Figure 3.49. Two examples of marking stage in applying the rule *Raise product/ quotient/power to a power*

Instruction for input of result (free input mode): Enter the result of exponentiation.

Input of result (free input mode): In the free input mode the student has to enter the whole result in one box (Figure 3.50). T-algebra only checks equivalence of the entered result to the corrected one. It means that other simplifications, like multiplication of monomials (powers of variables) etc., are allowed. If the student asks the program for help, the program fills the box with the correct result. T-algebra does not simplify anything in the result; the structure (fractions, parentheses, order of variables, etc.) remains exactly the same as in the original expression.

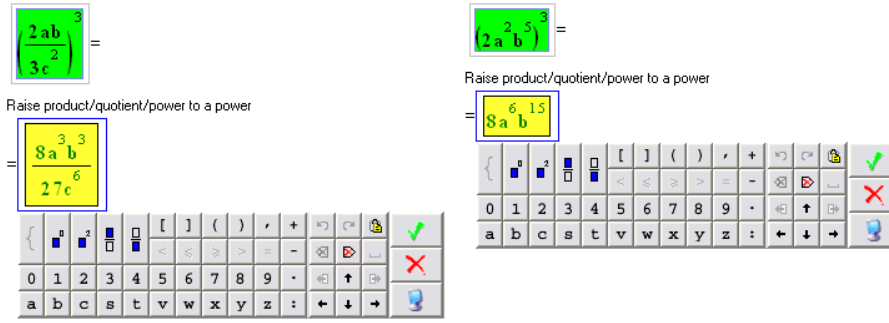


Figure 3.50 Two examples of input of result (free input mode) in applying the rule *Raise product/quotient/power to a power*

Instruction for input of result (structured input mode): Enter missing parts of the result.

Input of result (structured input mode): In the structured input mode, the student has to enter the result in multiple input boxes, forming the structure of the result (Figure 3.51). The structure of the result (fractions, parentheses, order of variables, etc.) remains exactly the same as the structure of the exponent base in the original expressions. As the structure of the result can be very complex (3 level fractions, multiplication of such fractions, etc.), checking the input would be quite difficult if changing the order of terms in the product or similar changes were allowed. Therefore, the order of terms in the result has to be exactly the same as in the original expression. Any part of the result entered into a specific box is checked independently from others (only for equivalence). If the student asks the program for help, the program fills the boxes with the correct result.

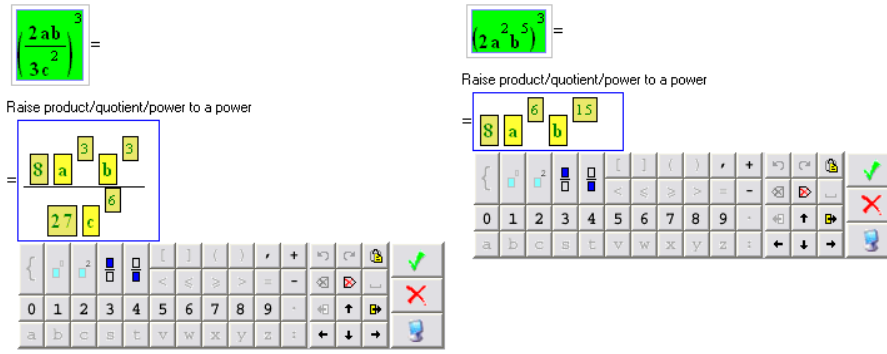


Figure 3.51. Two examples of input of result (structured input mode) in applying the rule *Raise product/quotient/power to a power*

Instruction for input of result (partial input mode): Enter missing exponents and coefficients in the result.

Input of result (partial input mode): The partial input mode is similar to the structured one, but some parts are filled in by the program. Only numeric parts of the expression (coefficients and exponents) should be entered by the student.

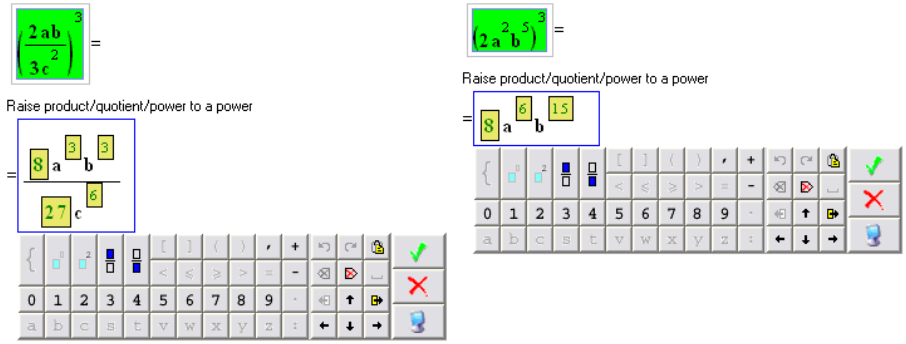


Figure 3.52 Two examples of input of result (partial input mode) in applying the rule *Raise product/quotient/power to a power*

3.2.17 Rule Raise number to a power

Applications: raise a number (fraction) to a power.

Instruction for marking: Mark a number together with exponent for exponentiation.

Marking: After selecting this rule, the student has to mark a number and power for exponentiation (Figure 3.53). Only one number can be raised to a power during one application of the rule. If the number is negative and written in parentheses, it should be marked together with parentheses. After confirmation of marking, the program checks all the mentioned aspects.

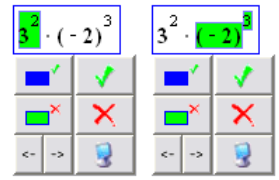


Figure 3.53. Two examples of marking stage in applying the rule *Raise number to a power*

Instruction for input of result (free input mode, structured input mode and partial input mode): Enter a sign and the resulting number.

Input of result (free input mode, structured input mode and partial input mode): The free input here coincides with the structured and partial input modes (structured input is used in all cases). The program gives two boxes for entering the result: one for the sign and the other for the number (Figure 3.54). Correctness of the sign and the number is checked separately when correctness of the step is evaluated.

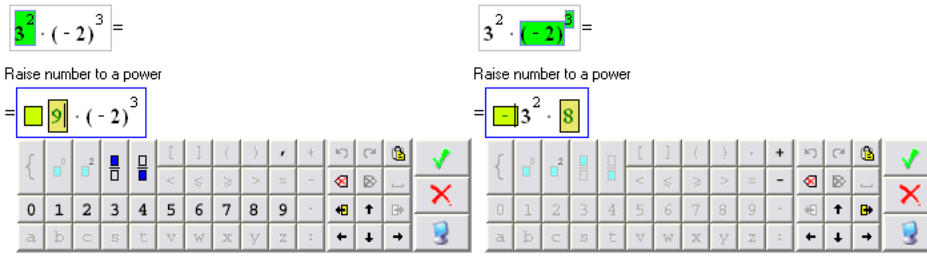


Figure 3.54. Two examples of input of result (free input mode, structured input mode and partial input mode) in applying the rule *Raise number to a power*

3.2.18 Rule Factor out common factor

Applications: factor out a common factor from all monomials in a polynomial; factor out a variable from one side of an equation (for solving a literal equation for a given variable).

Expression: a polynomial expression (sum of monomials) or a more complex expression including a polynomial (factoring out a common factor should be possible); an equation where a common factor can be factored out from one side of the equation.

Constraints for expression: a polynomial should consist only of monomials with a common factor (different from 1 or -1); terms (in any expression) should contain a common factor, which is a number, a variable or a monomial. In case of an equation, it should contain a variable for solving.

Instruction for marking: Choose terms for factoring out a common factor.

Marking: In order to carry out this operation, the student has to mark terms (monomials) for factoring out a common factor (Figure 3.55). If the expression is a polynomial (sum of monomials) and the problem type is *Factor out common factor* then the student has to mark all monomials (the whole polynomial). T-algebra does not allow factoring out a common factor of only some monomials in a polynomial if there are other monomials. If the expression is an equation then the student has to mark terms only on one side of the equation (which should be a polynomial as well). All these attributes are checked on confirmation of the marking. If the student asks for help, the program marks all possible terms that have a common factor (different from 1).

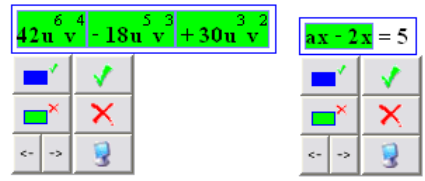


Figure 3.55. Two examples of marking stage in applying the rule *Factor out common factor*

Instruction for input of additional information: Enter common factor (number, variable or monomial).

Input of additional information: If the selected parts are suitable for application of the rule then, in structured and partial input modes, a separate window will be opened for entering the common factor (number, variable or monomial) (Figure 3.56). The common factor should not be the greatest common factor (left part of Figure 3.56). If an expression is entered, the program checks whether it is a common factor of all marked terms. If the student asks for help, the program writes the greatest common factor of marked terms in the box.

Many conditions are taken into account when calculating the greatest common factor. If all monomials contain the same variable in positive powers then the maximum power that is allowed in the common factor is the smallest power of this variable among all monomials (otherwise, negative exponents will appear). If at least one monomial contains a variable with a negative exponent then T-algebra, if asked for help, still offers the smallest exponent, but does not prevent the user from entering a different one (even if it is larger). Regarding the coefficient, if all coefficients are integers, the greatest common divider is calculated and used as the coefficient of the greatest common factor (this limitation also applies to user-entered coefficients).

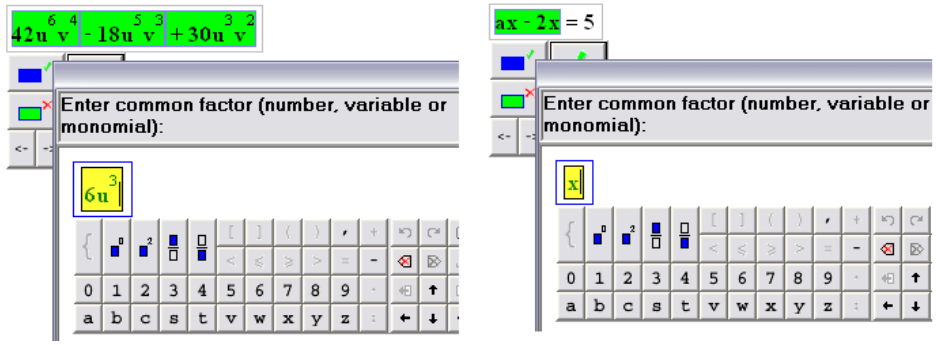


Figure 3.56. Two examples of input of additional information in applying the rule *Factor out common factor*

Instruction for input of result (free input mode): Enter the result of factorisation.

Input of result (free input mode): Input of additional information is skipped in the free input mode. If the marked parts are suitable for application of this rule then, in the free input mode, the program copies the expression onto the next line of the main window, replacing the marked parts with one empty box (Figure 3.57). The student has to enter the result of factorisation in this box.

The result should always be a product. The common factor should not be the greatest common factor.

After the input is confirmed, besides common checks the program checks whether the student has factored something out (whether the result is a product). If the student asks for help, the program composes a product of the greatest common factor and terms in parentheses.

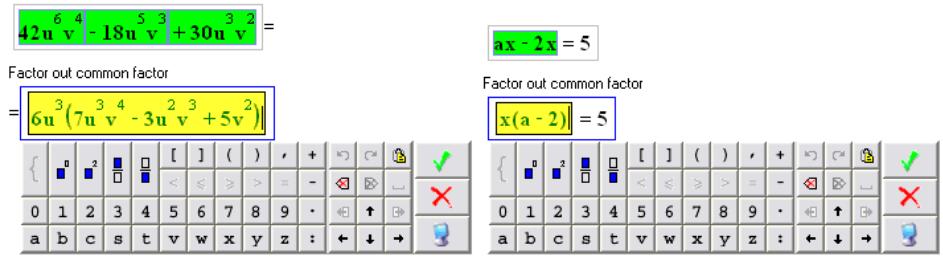


Figure 3.57. Two examples of input of result (free input mode) in applying the rule *Factor out common factor*

Instruction for input of result (structured input mode): Enter the result of factorisation.

Input of result (structured input mode): After the input of additional information has been confirmed, the program composes the expression on the next line of the main window. The program composes a product in place of the marked parts (the leftmost selected part): writes the previously entered and checked common factor and parentheses and leaves one empty box in parentheses for entering the result (Figure 3.58). After confirmation of the input, T-algebra performs common checks of the free input mode on the entered expression (verification of input, syntactical correctness and equivalence). If the student asks for help, the program writes the right polynomial in the box (taking into account the common factor entered in the additional window).

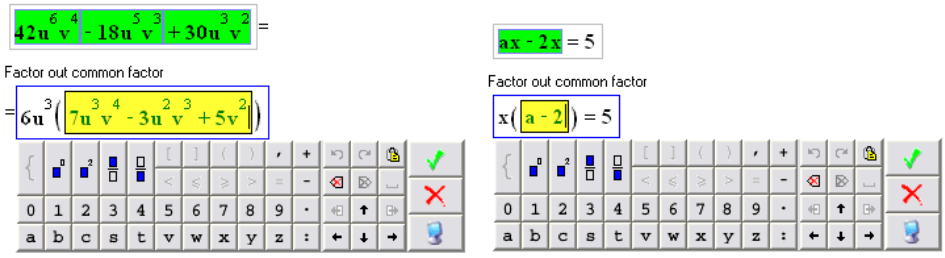


Figure 3.58. Two examples of input of result (structured input mode) in applying the rule *Factor out common factor*

Instruction for input of result (partial input mode): Enter the result of factorisation.

Input of result (partial input mode): The partial input mode is similar to the structured input mode but, instead of one box inside parentheses, the program proposes several boxes (Figure 3.59). There are two kinds of boxes: small boxes for input of signs + and –, larger boxes for entering numbers and variables with exponents (similar to the structured mode used for entering polynomial expressions). The number of boxes corresponds to the number of terms in the result (equal to the number of marked monomials). Terms can be placed in boxes in an arbitrary order. The program performs common checks of the structured input mode. The form of every entered part should be a monomial and this is verified by performing a rule specific check (form of every part). If the student asks for help, the program writes the correct terms in the boxes.

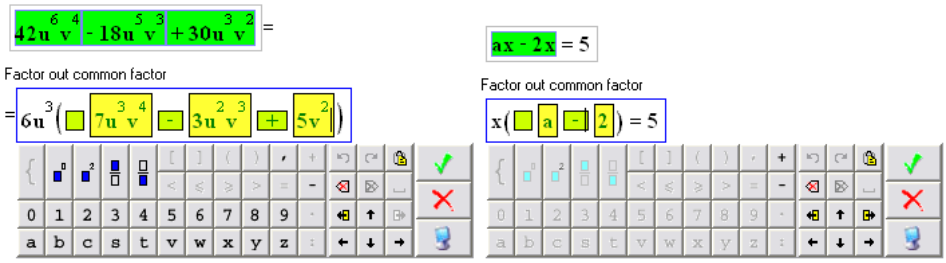


Figure 3.59. Two examples of input of result (partial input mode) in applying the rule *Factor out common factor*

3.3 Designed problem types in T-algebra

When designing T-algebra, we studied different problems that can be found in textbooks under the chosen topics. The problems were variable: there were various tasks (solve equation, calculate, simplify, etc.), different types of given expressions (equation, fractional numeric expression, polynomials, etc.), different kinds of expressions suitable for answer (a single number in calculation tasks, a polynomial in simplification tasks, etc.). For most of the tasks, the schoolbooks or teachers define, among other things, a recommended solution algorithm to follow.

In the design of T-algebra, we defined the problem types as sets of different attributes. The problem types in T-algebra actually correspond to typical problems that we have extracted from textbooks. There are 60 problem types implemented in T-algebra, grouped into 7 topics (corresponding to the same topics in school programme). I was responsible for designing and implementing problems in two fields: exponents and monomials (9 problem types) and polynomials (9 problem types). There were some typical problems that I did not implement in T-algebra, for example, trivial tasks of transforming a monomial/ polynomial to normal form. Different books and teachers use different

definitions of this normal form and, therefore, it was difficult to make it suitable for all teachers. Furthermore, reordering operations are quite trivial and do not reappear later in school programmes. Other typical problems involve calculating the result of an expression when the values of variables are given. Some problems require students to simplify an expression first and then substitute a variable, while others require substitution of variables first and then simply calculations. We implemented only the first problem type in T-algebra. The second differs only in terms of its solution algorithm but, as it is less used, I decided not to implement it.

If problem types are mentioned in any descriptions anywhere in this thesis, without specifying the topic, then it refers to these 18 problem types that were my responsibility.

In the following subsections we describe the implementation issues of problem types but, before that, we describe the main attributes of a problem type. A problem type in T-algebra contains the following attributes and algorithms:

- topic of the problem;
- default text of the problem (shown to the student);
- constraints for the initial expression;
- set of error messages for showing to the teacher during composition of problems;
- information on additional parameters (values) needed for the problem;
- constraints for the resulting expression;
- separate answering dialog (used, for example, in giving solution to an equation; possible choices: “no solution”, “any number is solution”, “certain number is solution”);
- set of rules that can be used by the student / T-algebra solution algorithm;
- heuristic function for selecting a transformation rule for the solution path;
- solution algorithm;
- procedure for checking the form of the answer;
- set of error messages for showing to the student when giving final answer to a problem;
- sign to be shown between rows in the solution (in case of the problems in my two fields it is, typically, the equality sign).

There are further two, slightly separate aspects from the viewpoint of implementation that are, nevertheless, connected to problem types and to problem type specific attributes and algorithms:

- description of random task generation variants;
- random task (expression and parameter values) generation algorithm.

Later in this thesis, we present all the problem types of these two fields: exponents and monomials (9 problem types) and polynomials (9 problem types). However, before proceeding with specific types, some general issues are discussed.

3.3.1 Problem type implementation principles

When designing the implementation for the problem types, some problems had to be solved. We had to create an interface that would allow separate implementation of problem types for different topics simultaneously by many developers.

Thus, we created an interface for problem types by creating the base class `TProblem` with all the required functions and data structures declared. It also included an implementation of functions that are common to different problem types. A problem type also defines certain error messages that are displayed to the user (both teacher and student) when checks are performed.

When a developers had to add a new problem type, they simply had to extend the `TProblem` class and override the methods with custom functionality. Then the newly created class had to be registered in a special list to be available to both student and teacher programs. We tried to create an interface where creation of separate problem types would require very little effort from the developer.

As the number of problem types is quite large, the number of different classes is also large. Working with them is quite convenient, because they are all implemented through one interface, `TProblem`. The only problems are storing the state and the problem files and reading problems from the problem files. Therefore, we used different internal constants (IDs) that define certain problem types and stored those IDs. A special method was designed for creating a problem from a given ID, initial problem expression and parameters, and no explicit calls to constructor are used at all (here `TAvaldis` is the base class for all algebraic expressions in T-algebra):

- `class function` **`CreateProblemById`**`(ProblemId: Integer; InitialAvaldis: TAvaldis; Parameters: TTStrings): TProblem;`

The `TProblem` class has many different methods for simpler creation of separate problem types. Some methods are problem type specific and not used elsewhere. Here we present the main functions and data structures of the base class that are used in every problem type:

- `class function` **`GetBaseRules`**`(): TIntegers;`

The method `GetBaseRules` returns the list of rules (list of rule IDs) allowed by a given problem type. Only these rules are used by the automatic solver and only these rules are listed in the menu to the student.

- `procedure` **`PreCheck`**`();`

This method is called after creation of each new problem to check whether the form of the initial expression is suitable for this problem type. Some general checks of the problem type are performed here, e.g., whether the expression includes some like terms (in the problem type *combine like terms*) or whether addition is the main operation of the expression, etc. (see the constraints for the initial expression in the descriptions of problem types). The expression is stored

internally in the class object and, therefore, no parameters are provided. In case of an unsuitable expression, an exception is raised (`EProblemException`), but an expression can be unsuitable even if no exception is raised. This is verified by trying to solve the problem from the given initial expression by calling the automatic solving function:

- `function` **`Solve`**`(const Avaldis: TAvaldis): Integer;`

This method solves the problem, starting from the expression given as the parameter `Avaldis`; the method returns the number of steps to be performed. If the result is 0, it means that `Avaldis` is also the answer – the expression is in the form of the answer and no rules from the algorithm can be applied. This method raises an exception if the problem cannot be solved. The `solve` method actually implements the general solution algorithm (section 3.3.3) and uses a problem type specific heuristic function to get information on the rule to be applied next. Although there is a possibility for the problem type to override this general solution algorithm, it appeared to be good enough to be used in all problem types.

- `function` **`GetHelp`**`(const Avaldis: TAvaldis): Integer;`

The method `GetHelp` provides the student with one specific kind of help available in T-algebra – information on which rule to apply next. Other kinds of help, e.g., selection of suitable objects and automatic rule application, are provided by `TRule` objects (rules). Consequently, this function is called when the user asks, from the user interface, for help on which rule to apply next.

The method `GetHelp` actually contains the heuristic for choosing the next rule to be applied to the expression given as the parameter `Avaldis` and is used by the automatic solving algorithm (function `Solve`). This method returns either ID of the rule to be applied or -1 if no suitable rule is found (either the expression is a solution to the problem or an unsuitable initial expression was given). This heuristic function is the most important element in the implementation of each problem type.

- `procedure` **`CheckAnswer`**`(const Avaldis: TAvaldis);`

The method `CheckAnswer` checks whether the expression given by the parameter is suitable as an answer to a problem of the current problem type (form of answer in the description of problem types). Some checks of the structure of expression are usually performed (for example, is it a polynomial or a single monomial – a very typical answer in simplification problems).

The other methods in the problem type base class are less important for simplification problem types. These methods include, for example, parsing parameter values (for instance, values of variables to be replaced by), returning the sign to be displayed between expressions on different lines (in simplification problems usually the equality sign), generating a special form of the answer (for instance, any number is a solution for equation), and displaying a separate window for giving answer to a problem if it differs from the expression on the

last line (e.g., in solution of an equation usually a choice of the correct answer from the list).

An average class (problem type) of a simplification problem contains 4 important methods (`PreCheck`, `GetHelp`, `GetBaseRules`, `CheckAnswer`) and only around 200–300 lines of code, depending on the complexity of the heuristic function and the number of allowed rules. Such small amount of code is possible because of a quite large basic class for problem types (>2000 lines of code) with all reusable functions to ease implementation of separate problem types. In addition, use of rule class API enables checking of different constraints in only three lines of code (for example, whether an expression contains like terms, whether sum is the main operation of an expression, etc.), which makes checking procedures (initial expression and result) quite declarative and easy to read. All the above factors make it quite simple to define new problem types in T-algebra after they are designed.

3.3.2 Usage of problem types

Problem types (appropriate classes) in T-algebra contain a large number of attributes and algorithms. Therefore, problem type classes are also widely used. Both the T-algebra student program and the teacher program require definitions of problem types.

The student's program uses the problem type definition for several purposes during the solving process: to show the student the allowed set of rules, to use the solution algorithm defined in the problem type to assist the student when he asks for help (for selecting the next rule), to generate an automatic solution, to check the student's answer, to generate a random problem for solving when the file is opened, etc.

The teacher's program also uses the problem type for many operations: to check the suitability of the initial expression entered by teacher for the problem type, to generate an automatic solution, to calculate an answer, to generate random task examples, etc.

Different values and attributes are also used in different situations: sets of error messages are used for displaying messages to the student during solving and to the teacher during composing. The sign between expressions is used for formatting the solutions (both automatic solutions in the teacher's program and student solutions in the student's program). A set of rules is used to limit the number of available rules displayed to the student when solving problems.

3.3.3 Automatic solving – general algorithm

As we have said in previous sections, each implementation of a problem type defines its own procedure with heuristics for the problem solving algorithm. This is implemented through a function that returns the most suitable transformation rule for a particular expression.

The general solution algorithm is quite simple. It consists of sequential calls to this function for returning a suitable rule for the next step, application of that transformation rule (through the transformation rule interface), checking whether the resulting expression has the required form of the answer of this problem type (through the problem type interface). The algorithm is quite straightforward; no recursion is used. The following diagram helps to explain the general algorithm (Figure 3.60).

However, there is a problem in the algorithm. It can possibly end in an infinite loop. There are some pairs of transformation rules in T-algebra that change expressions in opposite ways (for example, reducing and expanding fractions, etc.). If two such rules are used, it can lead to a situation where the expression remains the same after consecutive application of these rules.

For example, the *algorithm for combining* (section 3.3.4) is used in most simplification problems. It contains 3 rules in the following order:

- Extend common fraction;
- Combine like terms;
- Reduce.

If it would extend a fraction in the expression by means of the first rule and then reduce the same fraction by means of the third rule then we would end up with an infinite loop. Actually, such extending is not needed for T-algebra itself, because the rule of combining like terms is able to add fractions with different denominators or monomials with fractional coefficients. However, this was added in order to generate proper automatic solutions. Thus, to avoid looping, we defined such heuristics that the rule of extending is applied only in such instances and to those fractions that are combined at the next step. Otherwise, no fractions are extended.

Looping would also occur if a recursive algorithm for searching a solution path would be used. We could try to implement a looping detection functionality in the algorithm to avoid looping. However, we decided to keep the general algorithm as simple as possible. The general algorithm does not know anything about the expression and does not take it into account when applying rules.

The whole knowledge of expressions and transformation rules comes from a special heuristic function for selecting a suitable transformation rule. This function is specific for different problem types. The looping problem is also solved through this heuristic function.

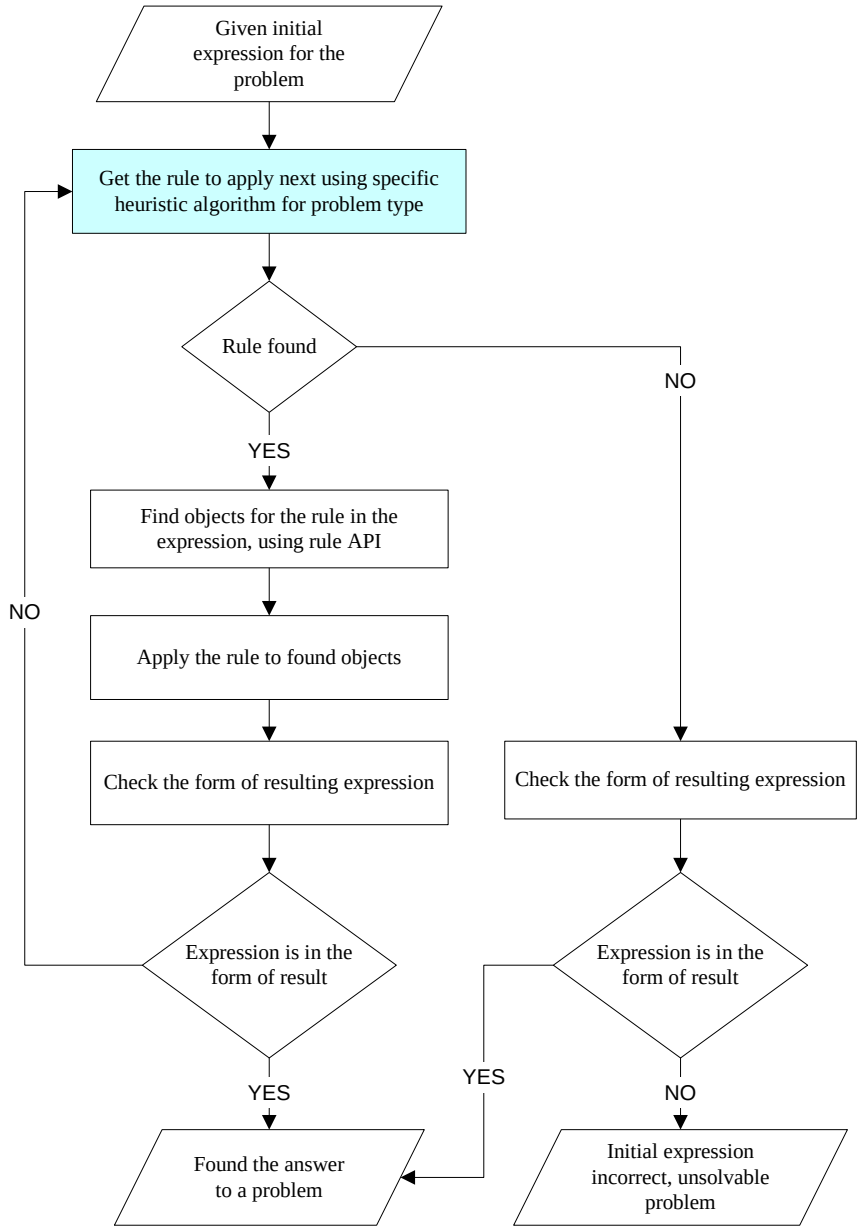


Figure 3.60. Automatic solving algorithm using the heuristic function for rule selection

If any of the recursive algorithms would be used (either depth-first or breadth-first), we could get the answer to a problem easily but we would face other problems. First of all, we would have the mentioned looping problem in case of depth-first search. Another problem is the speed of the solution algorithm if the number of allowed rules increases. There is also a further issue that made us

decide not to use recursion. As automatic solving is also used for assisting students (help for selecting the next rule) and generating automatic solutions, we did not want to have simply "any" solution. Of course, we could always get the shortest possible solution. However, we wanted to have some control over the solution path to be able to follow the "official" solution algorithms presented to the students in textbooks. Therefore, we decided to use the heuristic function.

This heuristic function is also quite simple in most cases. In the description of problem types (see sections 3.3.7 and 3.3.8), there is a preference list. A typical heuristic function cycles the list, searching for the first transformation rule that is applicable to the current expression. For checking whether a rule is applicable or not, a special method of the rule API is used (described in section 3.2.2). If the first such rule is found, it is returned as a result of the function (and the rule is applied). If no suitable rule is found then the algorithm stops (maybe the current expression is already a solution to the problem) and the form of answer is checked. After the rule is applied, a search for the next rule is performed from the start of the preference list. In some sophisticated cases, different heuristics is used to avoid looping.

Such heuristic function solution can be used quite well in polynomial simplification problems. Problems can arise in factorisation problems and in more complex simplification problems where factorisation operations are needed in order to reduce a fraction. However, such problems are beyond the scope of the school programme for 5–9th grades and, therefore, we did not implement them in T-algebra.

3.3.4 Sets of rules for the problem types

For the topics I was responsible for (exponents, monomials and polynomials) I have created set of new specific rules. For each new section (as in schoolbooks), a new transformation rule is introduced. As polynomials are the last topic from the school programme that is implemented in T-algebra, the students should already be familiar with the transformation rules designed for other topics. Many rules from previous topics are also required to be able to solve different polynomial problems. For example, most rules for operations with fractions (reducing, adding / subtracting, multiplying / dividing, etc.) are needed. Therefore, I used rules designed and implemented by other team members in my problem types.

There are two large groups of rules that I have used in my problem types. The whole list of rules from those groups is provided here. Later, when describing certain problem types, we simply refer to these groups. If at least one rule from the group is needed for solving a certain problem type, the whole group is added. Therefore, it is possible to refer to them later in descriptions of problem types.

The first group of rules is meant for manipulations with fractions (referenced later as the group of *rules for fractions*):

- Decrease integer part;
- Extend common fraction;
- Reduce;
- Improper fraction to mixed number;
- Mixed number to improper fraction;
- Common fraction to decimal;
- Decimal fraction to common.

Another large group of rules is meant for simplification of expressions with *0*, *1* and redundant pluses (referenced later as the group of *trivial simplification rules)*:

- Add/Subtract 0;
- Multiply/Divide 0;
- Multiply by 1;
- Divide by 1;
- Eliminate fraction with 0 in numerator;
- Eliminate denominator 1;
- Remove redundant pluses;
- Raise to power 1;
- Raise to power 0;
- Raise 1 to power;
- Raise 0 to power.

The order of transformation rules in the last group is important. The automatic solving algorithm tries to apply these rules in exactly the same order. Automatic algorithms for all problem types use this as a sub-algorithm at some stage, so this group of rules in this order is referenced also from the algorithm descriptions of different problem types (*algorithm for trivial simplification*).

Another typical algorithm (as a sequence of rules) used in the simplification problems is the so-called *algorithm for combining*. It actually includes the rules for operations with fractions. As the rule *combine like terms* even enables adding fractions with different denominators (or monomials with fractional coefficients), without converting those to one common denominator, we decided to have at least the automatic solution algorithm which will use rules for manipulating fractions – convert those to fractions with same denominators, etc., before adding and reducing after adding. The following order of rules is used for the algorithm, which will be referenced as the *algorithm for combining*:

- Decrease integer part;
- Add/Subtract numbers;
- Common fraction to decimal;
- Decimal fraction to common;

- Extend common fraction;
- Combine like terms;
- Reduce;
- Improper fraction to mixed number.

It is worth mentioning here that there are 3 pairs of rules that can possibly cause infinite loops:

- Decrease integer part and Improper fraction to mixed number;
- Extend common fraction and Reduce;
- Common fraction to decimal and Decimal fraction to common.

The heuristics used in these cases are similar – the first rules in these pairs (except Decimal fraction to common) are applied to certain fractions or decimals only if the same object would be used for combining like terms (Figure 3.61) or forming an answer (for example, if the problem type is to extend two fractions to the same denominator). In other cases, these rules are not used even if it were possible. This is done to avoid looping.

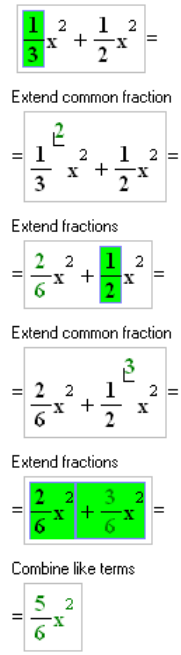


Figure 3.61. Example of heuristic (extend – combine – not reduce) in automatic solution

3.3.5 Typical constraints for initial and resulting expression in simplification problems

As mentioned above (section 3.3), the problem types contain numerous attributes, including constraints for the resulting expression. The topics of monomials and polynomials are somewhat different from other topics implemented in T-algebra. The form of the resulting expression is similar (or even almost the same) in all problem types. We have extracted three different types of constraints for the resulting expression or the form of answer. Here we describe them thoroughly and later, while describing the problem types, we will refer to these constraints.

Before specifying the form of answer, we define how T-algebra checks that problem is solved: the constraints for the resulting expression should be valid and the heuristic function for the automatic solving algorithm for the given problem type should not return any transformation rule that can be applied (meaning that T-algebra is not able to simplify the expression further). This enables problem type specific checks, depending on the set of rules. For example, if the rule set defined by the problem type contains a rule for reducing fractions, all fractions in the answer (for example, all coefficients of monomials) should be reduced. As fraction-related rules are added to almost all problem types. the following is true for almost all problem types. All fractions, if any, in the result should be reduced, integer part separated from the improper fraction.

The first typical form of answer is the so-called *simplified polynomial* or an expanded and reduced polynomial expression. It is used in almost all problem types in the topic of polynomials and in some problem types of the topic of monomials. The resulting expression should be either a single monomial (a single number or fraction is also a monomial) or a polynomial (a sum of monomials). All like terms in the polynomial should be combined. No multiplication signs are allowed in monomial. Only one instance of each variable should be present in a monomial (otherwise, it is possible to multiply monomials), except in the problem type *Combine like terms* (because no multiplication rule is available).

Some teachers require students to give answers in normal forms – all monomials in normal forms, variables placed in a specific order, monomials within polynomials also placed in a strict order. At the same time, there are quite many teachers who do not require answers in normal forms. T-algebra does not require any special order of variables in monomials or monomials inside polynomials.

Another typical form that is used in some problem types from the field of exponents and monomials is derived from simplified polynomials. It is a polynomial or single monomial as in previous case where the coefficients of monomials do not have to be raised to a power, i.e., it can be presented as a power of a number. In general, T-algebra allows the user to raise coefficients to a power but also allows giving an answer without it. In addition, a single power

with a base of any complexity is allowed as a form of answer, because some tasks from schoolbooks have similar answers. We will refer to this form as *simplified polynomial with powered coefficients*.

One more typical form of the resulting expression, which is mostly used in the topic of operations with fractions but also in polynomials, is a *single number*. It is either a single integer, decimal, reduced fraction or a mixed number with reduced fractional part.

If some problem types use their specific form of answer, it will be presented in the description of that particular problem type.

The presented forms of the resulting expression also define the constraints for the initial expression. When a problem is composed (or generated), T-algebra checks whether it is possible to solve this problem with the set of rules defined for that problem type. This is done by running the automatic solving algorithm. If T-algebra is able to reach the required form of answer using the automatic algorithm (which uses only the set of rules defined in the problem type) then the initial expression suits this problem type. Some extra checks are still performed to check whether the main operation for the problem type is utilized (for example, for the problem type *Combine like terms*, T-algebra checks whether there are at least two terms to combine). These extra checks are presented in description of each problem type.

These extra checks of the initial expression for the problem type usually check existence or non-existence of a certain subexpression. For example, the problem type *Multiply monomials* checks whether the initial expression contains at least two monomial multiplications and does not contain any monomial divisions. These checks are performed with the help of transformation rules. To ensure existence of a certain type of subexpression, T-algebra checks that a certain rule is applicable to the initial expression. In case of non-existence check, the same rule should be inapplicable. For combined rules (multiply/ divide something), a special checking function is created that tries to find only the objects for division. This way it is possible to check whether an expression contains a certain quotient that is either allowed or prohibited by the problem type.

3.3.6 Scheme for presentation of problem types

In the next two sections we present the problem types for two topics. Each problem type is presented according to the same scheme. We describe the most important aspects and provide an example of an automatically generated solution. The following attributes are discussed:

- Typical text – the typical text of such problems, extracted from schoolbooks; this is used as the default text when adding problems in the teacher's program;

- Constraints (for expression) – special constraints or conditions for the initial expression (other than it should be possible to reach the required form of answer using the set of rules for this problem type);
- Parameters – any specific parameters for the problem type (for example, values of variables); usually, no parameters are defined and this section is skipped;
- Rules – set of allowed rules for this problem type; uses references to the groups of rules (section 3.3.4);
- T-algebra algorithm – actually, the description of the heuristic function, preferred order of transformation rules, stating which rules are to be applied to the expression in which order (see also section 3.3.3);
- Example of generated solution – a figure with a sample solution from T-algebra, some comments for certain cases; the figure also presents one example of the problem (a suitable initial expression);
- Form of the resulting expression – the required form for the resulting expression, additional constraints, if any.

All problem types in T-algebra correspond to certain typical problems presented in Estonian schoolbooks under particular topics. When designing problem types, we tried to define the constraints for initial expressions based on the expressions used in these typical problems. In addition, sets of rules are limited to the rules the student should have learnt by that moment in the school programme.

3.3.7 Problem types for the field of exponents and monomials

3.3.7.1 Problem type Multiplication of powers

Typical text: multiply powers.
Constraints (for expression): the initial expression for the problem

- should contain at least one product of powers with the same base;
- should not contain a quotient of powers with the same base.

Rules:

- Multiply/Divide terms with the same base;
- Combine like terms;
- Multiply/Divide monomials;
- Raise number to a power;
- Clear parentheses;
- Add/Subtract numbers;
- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4).

T-algebra algorithm (preferred order of rules for heuristic function):

1. Rule *Multiply/Divide terms with the same base*;
2. *Algorithm for trivial simplification* (section 3.3.4);
3. Rule *Multiply/Divide monomials*;
4. Rule *Clear parentheses*;
5. *Algorithm for combining* (section 3.3.4).

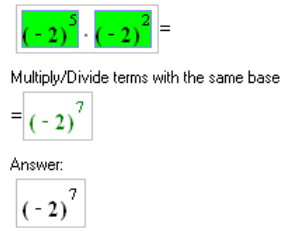


Figure 3.62. Example of generated solution for the problem type *Multiplication of powers*

Even though the rule *Raise number to a power* is available, the automatic algorithm of T-algebra does not use it. This is due to the form of the answer in schoolbooks: they instruct students only to calculate the correct power and give the answer as is.

Form of the resulting expression is *simplified polynomial with powered coefficients* (for details, see section 3.3.5).

3.3.7.2 Problem type Division of powers

The problem type *Division of powers* is actually very similar to *Multiplication of powers* (section 3.3.7.1). A separate problem type was created mainly for convenience of the teacher. The problems are separated in schoolbooks: at first, only multiplication problems are presented, then division is explained and division problems are introduced. Therefore, in this description we present only the differences from the multiplication type.

Typical text: divide powers.

Constraints (for expression): the initial expression for the problem should contain a quotient of powers with the same base.

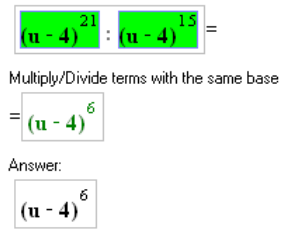


Figure 3.63. Example of generated solution for the problem type *Division of powers*

3.3.7.3 Problem type Raising a product to a power

Typical text: raise a product to a power.

Constraints (for expression): the initial expression for the problem

- should contain at least one product, which should be raised to a power;
- should not contain a quotient, which should be raised to a power;
- should not contain a power, which should be raised to a power.

Rules:

- Multiply/Divide terms with the same base;
- Raise product/quotient/power to a power;
- Combine like terms;
- Multiply/Divide monomials;
- Clear parentheses;
- Add/Subtract numbers;
- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4);
- Move minus before the fraction.

T-algebra algorithm:

1. *Algorithm for trivial simplification* (section 3.3.4);
2. Rule *Raise product/quotient/power to a power*;
3. Rule *Multiply/Divide terms with the same base*;
4. Rule *Multiply/Divide monomials*;
5. Rule *Clear parentheses*;
6. *Algorithm for combining* (section 3.3.4);
7. Rule *Move minus before the fraction*.

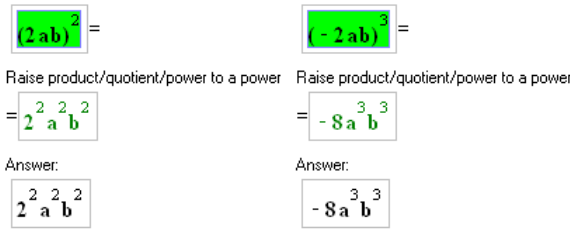


Figure 3.64. Two examples of solution for the problem type *Raising a product to a power*: student solution on the left, generated solution on the right

There are no special restrictions on the **form of the resulting expression** for this problem type. As the main transformation rule that is used and taught for this problem type is *Raise product/quotient/power to a power* and that rule has almost no restrictions on the complexity of the power base expression (the main operation of the subexpression should be a quotient, product or a fraction), we did not want to restrict the user. The expression is counted as an answer to the

problem if the solution algorithm for the problem type is unable to find any rule to be applied. Thus the expression should be simplified as far as possible.

As we can see from the example (Figure 3.64), there are different possibilities for the resulting expression. The main rule gives limited freedom for entering the result (present the coefficient as a power of a number as in the left part of the figure) and, as there is no rule for raising the number to a power, T-algebra accepts such answers.

3.3.7.4 Problem type Raising a quotient to a power

The problem type *Raising a quotient to a power* is actually very similar to the previous problem type *Raising a product to a power* (section 3.3.7.3). A separate problem type was created mostly for convenience of the teacher. These problems are separated in schoolbooks. Therefore, in this description we present only the differences from the previous problem type.

Typical text: raise a quotient to a power.

Constraints (for expression): the initial expression for the problem

- should contain at least one quotient, which should be raised to a power;
- should not contain a power, which should be raised to a power.

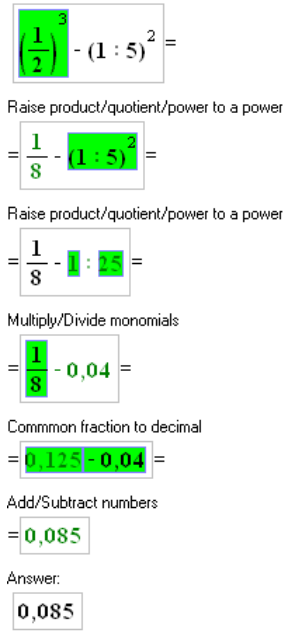


Figure 3.65. Example of generated solution for the problem type *Raising a quotient to a power*

3.3.7.5 Problem type Raising a power to a power

The problem type *Raising a power to a power* is actually very similar to the previous problem type *Raising a product to a power* (section 3.3.7.3). A separate problem type was created mostly for convenience of the teacher. These problems are separated in schoolbooks. Therefore, in this description we present only the differences from the previous problem type.

Typical text: raise a power to a power.

Constraints (for expression): the expression should contain at least one power, which should be raised to a power.

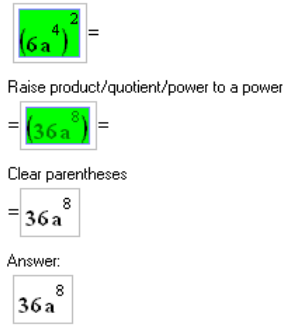


Figure 3.66. Example of generated solution for the problem type *Raising a power to a power*

3.3.7.6 Problem type Multiplication of monomials

Typical text: multiply monomials and simplify if possible.

Constraints (for expression): the initial expression for the problem

- should contain at least one product of monomials;
- should not contain a quotient of monomials.

Rules:

- Combine like terms;
- Multiply/Divide monomials;
- Clear parentheses;
- Add/Subtract numbers;
- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4).

T-algebra algorithm:

1. *Algorithm for trivial simplification* (section 3.3.4);
2. Rule *Multiply/Divide monomials*;
3. Rule *Clear parentheses*;
4. *Algorithm for combining* (section 3.3.4).

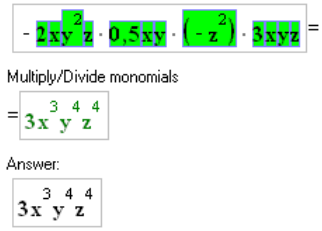


Figure 3.67. Example of generated solution for the problem type *Multiplication of monomials*

Form of the resulting expression is *simplified polynomial* (section 3.3.5).

3.3.7.7 Problem type Division of monomials

The problem type *Division of monomials* is actually very similar to the previous problem type *Multiplication of monomials* (section 3.3.7.6). A separate problem type was created mostly for convenience of the teacher. These problems are separated in schoolbooks. Therefore, in this description we present only the differences from the previous problem type.

Typical text: multiply and divide monomials and simplify if possible.

Constraints (for expression): the initial expression for the problem should contain at least one quotient of monomials (division should be expressed with the sign ":", division as a fraction is not suitable).

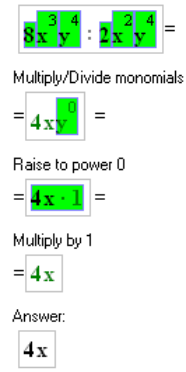


Figure 3.68 Example of generated solution for the problem type *Division of monomials*

3.3.7.8 Problem type Raising monomials to a power

Typical text: raise monomials to a power and simplify if possible.

Constraints (for expression): the expression should contain at least one monomial, which should be raised to a power.

Rules:

- Combine like terms;
- Multiply/Divide monomials;
- Raise number to a power;
- Raise monomial to a power;
- Clear parentheses;
- Add/Subtract numbers;
- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4).

T-algebra algorithm:

1. *Algorithm for trivial simplification* (section 3.3.4);
2. Rule *Raise number to a power*;
3. Rule *Raise monomial to a power*;
4. Rule *Multiply/Divide monomials*;
5. Rule *Clear parentheses*;
6. *Algorithm for combining* (section 3.3.4).

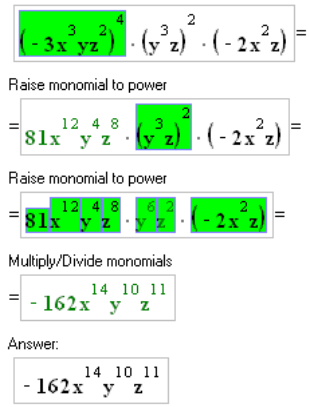


Figure 3.69. Example of generated solution for the problem type *Raising monomials to a power*

Form of the resulting expression is *simplified polynomial* (section 3.3.5).

3.3.7.9 Problem type Calculation of value of expression with integer exponents when values of variables are given

Typical text: simplify and find the value of expression if the variable values are...

Constraints (for expression): any expression, which is not a linear equation, a linear inequality or a system of linear equations. The initial expression should contain at least one variable to be substituted with a given value.

Parameters: values of variables.
Rules:

- Substitute variable;
- Combine like terms;
- Multiply/Divide monomials;
- Clear parentheses;
- Raise number to a power;
- Raise monomial to a power;
- Multiply/Divide terms with the same base;
- Raise product/quotient/power to a power;
- Add/Subtract numbers;
- Multiply/Divide numbers;
- Move minus before fraction;
- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4).

T-algebra algorithm:

1. *Algorithm for trivial simplification* (section 3.3.4);
2. Rule *Move minus before fraction*;
3. Rule *Multiply/Divide numbers*;
4. Rule *Multiply/Divide monomials*;
5. Rule *Raise number to a power*;
6. Rule *Raise monomial to a power*;
7. Rule *Raise product/quotient/power to a power*;
8. Rule *Clear parentheses*;
9. *Algorithm for combining* (section 3.3.4);
10. Rule *Substitute variable*.

Example of generated solution:

Text of the problem: Simplify and find the value of the expression if the variable values are *a=2*, *b=-3*.

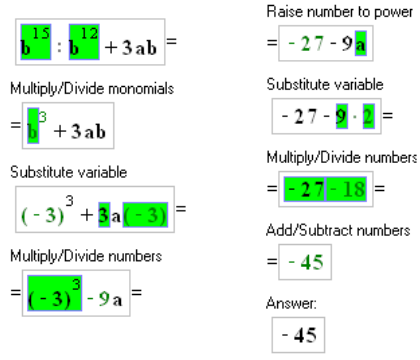


Figure 3.70. Example of generated solution for the problem type *Calculation of value*

Form of the resulting expression is *single number* (section 3.3.5).

3.3.8 Problem types for the field of polynomials

3.3.8.1 Problem type Combine like terms

Typical text: combine like terms.

Constraints (for expression):

- the sum should contain like terms;
- parentheses can only be around one monomial.

Rules:

- Combine like terms;
- Add/Subtract numbers;
- Clear parentheses;
- Move minus before fraction;
- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4).

T-algebra algorithm:

1. *Algorithm for trivial simplification* (section 3.3.4);
2. *Algorithm for combining* (section 3.3.4);
3. Rule *Clear parentheses*.

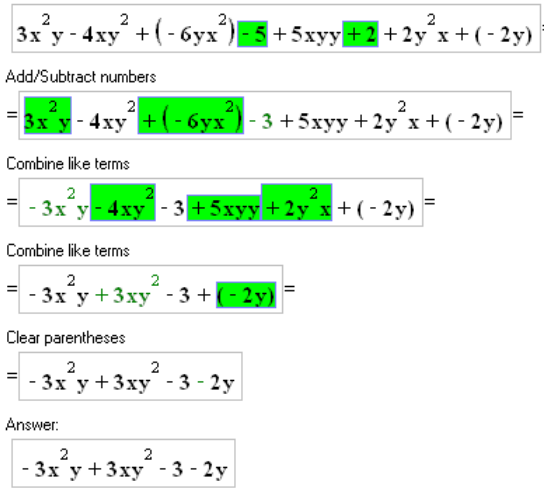


Figure 3.71. Example of generated solution for the problem type *Combine like terms*

Form of the resulting expression is *simplified polynomial* (Section 3.3.5).

3.3.8.2 Problem type Addition and subtraction of polynomials

Typical text: add and subtract polynomials.
Constraints (for expression): the expression should contain an addition or/and a subtraction of polynomials – at least one polynomial should be in parentheses and the main operation in the expression should be adding/subtracting.
Rules:

- Combine like terms;
- Clear parentheses;
- Add/subtract number;
- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4).

T-algebra algorithm:

1. *Algorithm for trivial simplification* (section 3.3.4);
2. Rule *Clear parentheses*;
3. *Algorithm for combining* (section 3.3.4).

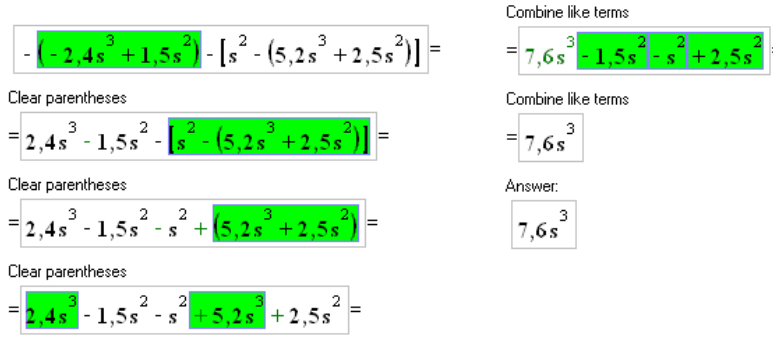


Figure 3.72. Example of generated solution for the problem type *Addition and subtraction of polynomials*

Form of the resulting expression is *simplified polynomial* (Section 3.3.5).

3.3.8.3 Problem type Multiplication of polynomial by monomial

Typical text: multiply polynomials by monomials and combine like terms.
Constraints (for expression): the expression

- should contain at least one product of polynomial by monomial;
- should not contain quotients.

Rules:

- Combine like terms;
- Multiply/Divide monomials;
- Raise number to a power;
- Raise monomial to a power;
- Clear parentheses;

- Multiply/Divide polynomial by monomial;
- Add/Subtract numbers;
- Multiply/Divide numbers;
- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4).

T-algebra algorithm:

1. *Algorithm for trivial simplification* (section 3.3.4);
2. Rule *Multiply/Divide polynomial by monomial*;
3. Rule *Multiply/Divide numbers*;
4. Rule *Multiply/Divide monomials*;
5. Rule *Raise number to a power*;
6. Rule *Raise monomial to a power*;
7. Rule *Clear parentheses*;
8. *Algorithm for combining* (section 3.3.4).

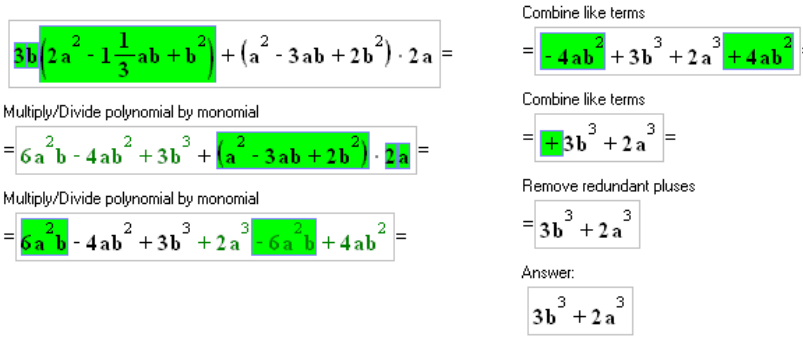


Figure 3.73. Example of generated solution for the problem type *Multiplication of polynomial by monomial*

Form of the resulting expression is *simplified polynomial* (Section 3.3.5).

3.3.8.4 Problem type Division of polynomial by monomial

Typical text: multiply and divide polynomials by monomials and combine like terms.

Constraints (for expression): the expression should contain at least one quotient of polynomial by monomial (division should be expressed with the sign ":", division as a fraction is not suitable).

Rules:

- Combine like terms;
- Multiply/Divide monomials;
- Raise number to a power;
- Raise monomial to a power;
- Clear parentheses;

- Multiply/Divide polynomial by monomial;
- Add/Subtract numbers;
- Multiply/Divide numbers;
- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4).

T-algebra algorithm:

1. *Algorithm for trivial simplification* (section 3.3.4);
2. Rule *Multiply/Divide polynomial by monomial*;
3. Rule *Multiply/Divide numbers*;
4. Rule *Multiply/Divide monomials*;
5. Rule *Raise number to a power*;
6. Rule *Raise monomial to a power*;
7. Rule *Clear parentheses*;
8. *Algorithm for combining* (section 3.3.4).

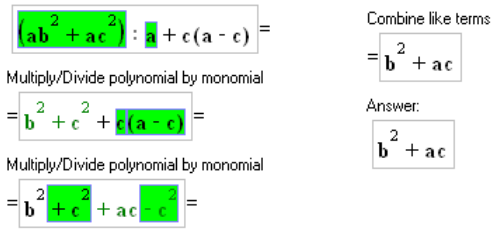


Figure 3.74. Example of generated solution for the problem type *Division of polynomial by monomial*

Form of the resulting expression is *simplified polynomial* (section 3.3.5).

3.3.8.5 Problem type Multiplication of polynomials

Typical text: multiply polynomials and combine like terms.
Constraints (for expression): the expression should contain at least one product of polynomials.
Rules:

- Combine like terms;
- Multiply/Divide monomials;
- Raise number to a power;
- Raise monomial to a power;
- Clear parentheses;
- Multiply/Divide polynomial by monomial;
- Multiply polynomials;
- Add/Subtract numbers;
- Multiply/Divide numbers;

- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4).

T-algebra algorithm:

1. *Algorithm for trivial simplification* (section 3.3.4);
2. Rule *Multiply polynomials*;
3. Rule *Multiply/Divide polynomial by monomial*;
4. Rule *Multiply/Divide numbers*;
5. Rule *Multiply/Divide monomials*;
6. Rule *Raise number to a power*;
7. Rule *Raise monomial to a power*;
8. Rule *Clear parentheses*;
9. *Algorithm for combining* (section 3.3.4).

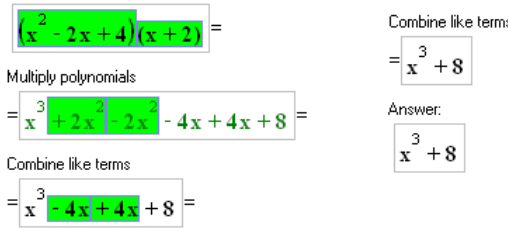


Figure 3.75. Example of generated solution for the problem type *Multiplication of polynomials*

Form of resulting expression is *simplified polynomial* (section 3.3.5).

3.3.8.6 Problem type Multiplication of polynomials with the help of formulas

Typical text: multiply polynomials with the help of formulas and then combine like terms.

Constraints (for expression): the expression should contain at least one product/power of polynomials, which can be simplified by one of four simplification formulas (expand the square of the sum/difference of two monomials, expand the cube of the sum/difference of two monomials, multiply the sum and difference of two monomials, multiply the sum or the difference of two monomials by incomplete square).

Rules:

- Combine like terms;
- Multiply/Divide monomials;
- Raise number to a power;
- Raise monomial to a power;
- Clear parentheses;
- Multiply/Divide polynomial by monomial;
- Multiply polynomials;

- $(a\pm b)^2$ =>;
- $(a\pm b)^3$ =>;
- (a+b)(a-b) =>;
- $(a\pm b)(a^2\pm ab+b^2)$ =>;
- Add/Subtract numbers;
- Multiply/Divide numbers;
- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4).

T-algebra algorithm:

1. *Algorithm for trivial simplification* (section 3.3.4);
2. Rule $(a\pm b)^2$ =>;
3. Rule $(a\pm b)^3$ =>;
4. Rule (a+b)(a-b) =>;
5. Rule $(a\pm b)(a^2\pm ab+b^2)$ =>;
6. Rule *Multiply polynomials*;
7. Rule *Multiply/Divide polynomial by monomial*;
8. Rule *Multiply/Divide numbers*;
9. Rule *Multiply/Divide monomials*;
10. Rule *Raise number to a power*;
11. Rule *Raise monomial to a power*;
12. Rule *Clear parentheses*;
13. *Algorithm for combining* (section 3.3.4).

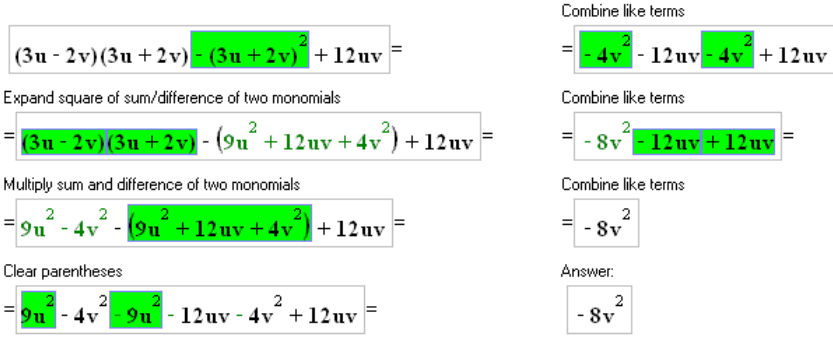


Figure 3.76. Example of generated solution for the problem type *Multiplication of polynomials with the help of formulas*

Form of the resulting expression is *simplified polynomial* (Section 3.3.5).

3.3.8.7 Problem type Calculation of value of polynomial when values of variables are given

Typical text: simplify and calculate the value of the expression if the values of variables are...

Constraints (for expression):

- any polynomial expression (not linear equation, linear inequality or system of linear equations);
- expression should contain at least one variable.

Parameters: values of variables.
Rules:

- Substitute variable;
- Combine like terms;
- Multiply/Divide monomials;
- Clear parentheses;
- Raise number to a power;
- Raise monomial to a power;
- Multiply/Divide terms with the same base;
- Raise product/quotient/power to a power;
- Multiply/Divide polynomial by monomial;
- Multiply polynomials;
- $(a \pm b)^2 =>$;
- $(a \pm b)^3 =>$;
- $(a+b)(a-b) =>$;
- $(a \pm b)(a^2 \pm ab+b^2) =>$;
- Add/Subtract numbers;
- Multiply/Divide numbers;
- Move minus before fraction;
- Group of *rules for fractions* (section 3.3.4);
- Group of *trivial simplification rules* (section 3.3.4).

T-algebra algorithm:

1. *Algorithm for trivial simplification* (section 3.3.4);
2. Rule *Move minus before fraction*;
3. Rule $(a \pm b)^2 =>$;
4. Rule $(a \pm b)^3 =>$;
5. Rule $(a+b)(a-b) =>$;
6. Rule $(a \pm b)(a^2 \pm ab+b^2) =>$;
7. Rule *Multiply polynomials*;
8. Rule *Multiply/Divide polynomial by monomial*;
9. Rule *Multiply/Divide numbers*;
10. Rule *Multiply/Divide monomials*;
11. Rule *Raise number to a power*;
12. Rule *Raise monomial to a power*;
13. Rule *Raise product/quotient/power to a power*;
14. Rule *Clear parentheses*;
15. *Algorithm for combining* (section 3.3.4);
16. Rule *Substitute variable*.

Example of generated solution:

Text of the problem: Simplify and calculate the value of the expression if the values of variables are $x=-0.6$, $y=0,3$.

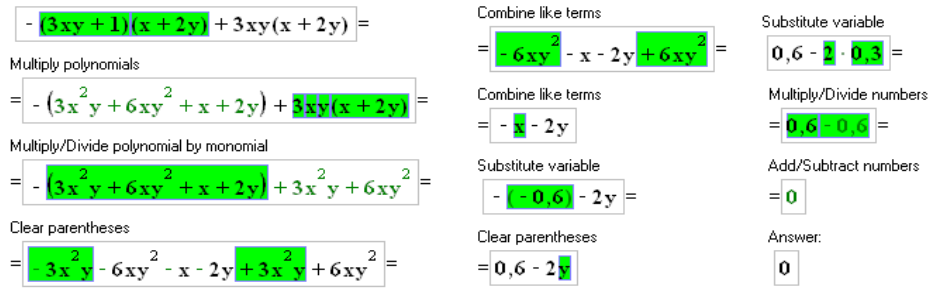


Figure 3.77. Example of generated solution for the problem type *Calculating of value of polynomial*

Form of the resulting expression is *single number* (Section 3.3.5).

3.3.8.8 Problem type Factoring out common factor

Typical text: factor out common factor.

Constraints (for expression): the expression should contain the sum of such monomials that have a common factor different from *1*.

Rules:

- Factor out common factor;
- Multiply/Divide monomials;
- Multiply/Divide numbers;
- Clear parentheses.

T-algebra algorithm:

1. Rule *Factor out common factor*;
2. Rule *Multiply/Divide numbers*;
3. Rule *Multiply/Divide monomials*;
4. Rule *Clear parentheses*.

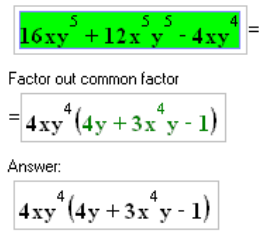


Figure 3.78. Example of generated solution for the problem type *Factoring out common factor*

Form of the resulting expression should be a product (an expression where the greatest common factor is placed before parentheses). All other typical constraints on numbers, monomials and polynomials apply. For example, numbers and coefficients should be reduced if they are fractions.

4 CONDUCTED EXPERIMENTS

We have conducted different experiments with students for different purposes while designing and developing T-algebra. I have participated in some that were related to the topic of polynomials. This section covers the following experiments and goals:

1. to identify mistakes made by 7th and 8th grade students (during solving polynomial simplification problems) when working with pencil and paper and their possibility in T-algebra;
2. to validate the designed user interface and to study the distribution of errors between stages of a solution step;
3. to try to use T-algebra with students while explaining new material;
4. to learn about errors made in solving polynomial simplification problems by 11th grade students and verify implementation of transformation rules and error diagnosis in T-algebra.

Results of these experiments are published (Issakova et al., 2006; Lepp, 2007a; Lepp, 2007b; Prank and Lepp, 2010). The last experiment was conducted later – after T-algebra was distributed to Estonian schools. One of our goals was also to perform testing in larger groups.

4.1 Study of student mistakes on paper

Prior to designing the transformation rules for monomial and polynomial simplification problems, we conducted a study among different groups of students, collected and classified different mistakes that most students make. Later we used the collected information to design the general rule dialogue, certain transformation rules and error diagnosis procedures for different transformation rules that we have implemented in T-algebra. Designing the rules, we have attempted to leave an opportunity for students to make the same mistakes in T-algebra as they do in paper solutions. When students apply these rules, T-algebra checks many different attributes and tries to detect certain typical errors. In case of typical errors for which we have implemented special diagnostic procedure, T-algebra shows the student an appropriate message. T-algebra can also check for non-equivalence of expressions. We have also preserved the possibility to add specific diagnostic procedures for different rules in the future if we find further typical errors.

In order to make the T-algebra intelligent enough to diagnose different student errors and help to correct them, we first had to understand these errors ourselves. We studied the results of similar researches (Tall et al., 1993; Payne et al., 1990; Weitz et al., 2007; Hall, 2002) and conducted our own experiment with the students to collect typical errors that students make when solving problems (Issakova, 2005). For example, Lewis (Lewis et al., 1987) mention a study of errors in factoring problems where they propose to use three input modes similar to those we have implemented in T-algebra.

The experiment on errors in simplification problems took place in Estonian schools (selected schools in Tartu) in the winter of 2005 (experiments related to other themes of T-algebra were conducted at the same time). For this study, mathematics teachers (Mart and Maire Oja) composed two different tests in two variants each. Two different groups of students participated in the tests. The tests consisted of different types of problems that were later implemented in T-algebra.

A total of 33 students, aged 13 years (7th grade), participated in the first test. The test included calculation problems as well as some easiest simplification problems. Simplification problems required application of two simplification operations at most: combining like terms (7 problems with at most one variable in a monomial, example $6b-10b$) and multiplication of a monomial (usually a single number or a variable) by a polynomial (8 problems, example $-a(2b-4b+5)$). The list of problems is presented in Appendix A. The same problems were later used to verify the designed user interface of T-algebra in subsequent experiments.

A total of 54 students, aged 14 years (8th grade; two different classes with different math teachers), participated in the second test. The test included different types of problems: combine like terms (4 problems, many variables, example $3u^2v^3-2uuv^2v+u^2vvv$), multiply or divide polynomial by a monomial (7 problems, example $((20x^3y^2+12x^2y^3-4xy^2):(-4xy^2))$), multiply polynomials (10 problems, example $(u-3)\cdot(2u^2+3u-1)$), problems requiring application of all the mentioned operations (4 problems), as well as some easiest factorisation problems (see the list in Appendix A).

Both groups of students had learned the topics of the tests in autumn 2004 and the material of the test was not new. The students did not know about the test beforehand and had 45 minutes to complete the test. We collected all the solutions, checked them and tried to identify typical errors among solutions of students in both age groups.

Here I present the typical errors of all students in applying the following operations: combine like terms, and multiply or divide polynomial by a monomial. I was also able to compare the results of two different groups of children – what errors are typical at early stages of learning the simplification rules and solving problems and what errors become more typical at further stages of the education process.

The result of the test confirmed our assumptions that students make both topic-specific mistakes that occur only in simplification problems as well as mistakes related to previously studied material. The following two tables present typical errors (that were made at least by two students) in applying the two mentioned operations. The columns with numbers of students show the number and the percentage of students (in an age group) who made this mistake. These tables do not reflect if the students made these mistakes more than once.

Table 4.1. Mistakes in combining like terms

No	Nature of mistake	Example of mistake	Number of students 7th grade	Number of students 8th grade
1	Combines non-like terms, combines terms with different variable parts	$3m+7=10m$ $3ab^2-a^2b=2ab^2$	4 (12%)	14 (26%)
2	Forgets to take into account some signs before monomials	$-7b+2b-2b=-3b$ $5ab^2-5ab^2=10ab^2$	8 (24%)	9 (17%)
3	Error in calculating the sign of the resulting monomial	$...-4x+2x=...+2x$ $7b^2-12b^2+7b^2=-2b^2$	8 (24%)	5 (9%)
4	Arithmetical error in calculating the coefficient	$9x+4x=12x$ $10xy-yx-2xy=8xy$	9 (27%)	26 (48%)
5	Error in powers of variables	$9x^2z+4x^2z=13x^4z^2$ $10xyx-x^2y=9xy$	0 (0%)	6 (11%)
6	Does not combine all terms, does not recognize like terms if they are like	$3x^2z+x^2z-2xzx=4x^2z-2xzx$ $2a-b+a-2b=3a-b-2b$	6 (18%)	18 (33%)
7	Forgets to copy some unchanged terms, copies terms with mistakes	$y+2y+x^2y=3y+y^2x$ $2x-3x-2=-x$	8 (24%)	3 (6%)

Table 4.2. Mistakes in multiplying or dividing of a polynomial by a monomial

No	Nature of mistake	Example of mistake	Number of students 7th grade	Number of students 8th grade
8	Does not multiply or divide one of the terms of the polynomial by the monomial	$(3u-2)\cdot 4=3u-8$ $2x\cdot(3x-2)=6x^2-2$	8 (24%)	2 (4%)
9	Does not change signs of some monomials in result	$-m(3x+2y-4)=-3mx-2my-4m$ $-2x-(y-2x)=-2x-y-2x=-y-4x$	14 (42%)	8 (15%)
10	Multiplies the polynomials or polynomials by monomials instead of adding	$16-(5x+3y-1)=80x-48y+16$ $(x-2)+(x-y)=x^2-xy-2x+2y$	4 (12%)	11 (20%)

No	Nature of mistake	Example of mistake	Number of students 7th grade	Number of students 8th grade
11	Mistake in calculating the coefficient of single monomial in the result	$a(2b-3c)=2ab-2ac$ $3x(4y-3z)=7xy-6xz$	8 (24%)	3 (6%)
12	Mistake in calculating the power of a variable in a single monomial in the result	$ab^2(2a^3b^2-3ab)=2a^3b^4-3a^2b^3$ $3x^2(4x^5-3y)=12x^6-9y$	0 (0%) impossible in 7th grade problems	13 (24%)
13	When dividing the same monomials the result is 0	$(20x^3y^2+12x^2y^3-4xy^2):(-4xy^2)=$ $=-5x^2-3xy$	0 (0%) impossible in 7th grade problems	8 (15%)

As can be seen from the tables, the errors made by the students are of different kind. In some errors (1, 6 and 10), the student does not recognize correct objects of transformation (correct like terms, monomial and polynomial product, etc.). Another group contains errors where the student calculates the result of operation incorrectly. The error can be associated with the signs of monomials (2, 3 and 9), coefficients of monomials (4, 11 and 13), or powers of variables (5, 12). The errors where the student forgets to copy unchanged parts of expression belong to a different group and are probably caused by oversight. These errors almost disappear in the 8th grade.

Let us compare the numbers of students who made different types of errors in 7th and 8th grade. We can see that the percentage of students who made errors in recognising like terms (1, 6) has greatly increased. This is probably because of the complexity of the problems (in 8th grade, more variables are used, different forms of monomials are used, such as xyx and x^2y, etc.). We can also see that the number of arithmetical errors (4) has grown, probably owing to the fact that the problems contain larger numbers and more negative numbers than the 7th grade problems, but also because the students tend to combine three or more terms at the same time. A new kind of errors (5) also appears in the 8th grade – errors in calculating the powers of variables. This is probably because, in the 7th grade, there are only variables (in the power of 1) and also because of the new operations that the students have learnt in the 8th grade (about half of the errors are such where the student adds the coefficients and also adds the powers of variables as if he was trying to multiply the monomials). New kinds of errors (12, 13) in application of the operation "multiply or divide polynomial by monomial" were found that were not present in the 7th grade solutions – these errors are caused by more complex problems (in the 7th grade, variables usually do not have powers; division of polynomial by a monomial is not introduced yet). There is an increase in errors associated with the recognition of a correct operation or correct operands. At this stage, the

students learn the operation "multiply the polynomials" and apply it even if it is not applicable (to the sum of polynomials).

4.1.1 Design decisions for transformation rules and typical error diagnosing in T-algebra

As we have seen, the difficulties that the students have when solving problems are of different kinds. They either cannot find the correct operation or the operands for it, or they make mistakes in calculating the resulting expression. Therefore, designing T-algebra, we had to leave a possibility for making both groups of errors and to make it easy for T-algebra to detect these errors. This resulted in an action-object-input scheme for a solution step.

T-algebra checks for correctness of each stage of every single solution step and can respond to errors with appropriate error messages. For every rule implemented in T-algebra, we have designed a specific set of input boxes for different input modes as well as diagnostic procedures in order to be able to diagnose different typical errors of students. Description of two rules mentioned in the study is given in section 3.2 (rule *Combine like terms* – section 3.2.5 and rule *Multiply/Divide polynomial by monomial* – section 3.2.9).

Let us consider which of the typical mistakes that students make on paper can be made in T-algebra. By saying that the mistake can be made, I mean not only the possibility for the student to make such errors but also that T-algebra can diagnose it, respond to it accordingly, or at least inform the user about the non-equivalence of expressions. The error numbers in the following table correspond to those from tables above. For each error type, I indicate the possibility of making it in T-algebra, how it is diagnosed and the error message shown to the student.

When applying the rule *Combine like terms*, most mistakes are made in recognizing like terms (can be easily diagnosed at the object selection stage), calculating the sign and coefficient of the resulting monomial (diagnosed at the input stage, separate boxes in structured and partial input modes). Less frequent among students and less important are mistakes in variables and their powers (can still be diagnosed in free and structured modes).

When applying the rule *Multiply/Divide polynomial by monomial*, most mistakes are made in recognizing the objects (can be easily diagnosed at the object selection stage), calculating the signs, coefficients and powers of variables in the monomials forming the resulting polynomial (diagnosed at the input stage, separate boxes for signs and monomials or their parts in structured and partial input modes) and in the number of monomials in the result (can be diagnosed in free and structured input modes). Less frequent among students and less important are mistakes in variables and their powers (can still be easily diagnosed in free and structured modes).

Table 4.3. Possibility of making typical errors made on paper when solving problems in T-algebra

No	Possible	Error diagnosis procedure or comment	Error message
1	yes	Diagnosed at object selection stage	"At least one of the terms is not similar to others"
2	yes	Compare combinations of coefficients with different signs with student input	specific check is not implemented, reports "Calculation error"
3	yes	Diagnosed at input stage	"Incorrect sign"
4	yes	Diagnosed at input stage	"Calculation error"
5	yes	Diagnosed at input stage, check whether resulting monomial is like with objects	"Incorrect variable part"
6	yes	Diagnosed when student tries to give an answer to a problem	"Like terms are not yet combined"
7	no	T-algebra copies unchanged parts	-
8	yes	Compare each resulting monomial with each initial one	currently not implemented, reports "Result should not contain such monomial"
9	yes	Diagnosed at input stage	"Incorrect sign"
10	yes	Diagnosed at object selection stage, selected unsuitable objects	"One monomial and one polynomial from the same product should be selected"
11	yes	Diagnosed at input stage, compared coefficient of each resulting monomial	"Incorrect coefficient"
12	yes	Diagnosed at input stage	"Incorrect variable part"
13	yes	Diagnosed at input stage, number of terms	"Result should contain more terms"

4.1.2 Conclusions

The study on student errors has indicated that even a small group of students can provide us with information on typical errors that students make when solving simplification problems on paper. The mistakes that two different groups of students make are very similar (8th grade students make some additional mistakes). I have collected sets of typical mistakes that students make when applying different transformation rules and later used this information to design the common rule dialogue, three input modes, diagnostic procedures and the input stage for each rule separately for the step-by-step problem-solving environment T-algebra. In addition, I have shown that all typical important errors that students make on paper can also be made when solving in the T-algebra.

4.2 Experiment for validation of user interface

In the spring of 2005, the same students from the first experiment participated in the trial of T-algebra. T-algebra was in the development phase at that time and, therefore, the objective of this trial was to validate only the user interface of the program from the point of view of its usability. Two topics were chosen for that purpose: operations with fractions and simplification of polynomials (the same topics were covered in paper tests). In this trial, the students were given exactly the same problems as in previously completed tests on paper. In addition, the problem set contained some demonstration examples from other chapters. The trial was conducted in two different classes. A 6th grade class was chosen for the topic of operations with fractions and an 8th grade class for the topic of simplification of polynomials. The students already had sufficient experience with computers (using the keyboard, mouse, Windows), but it was the first time they had seen T-algebra. The students could choose whether they wanted to sit at the computer alone or in pairs. For operations with fractions we had 25 computers occupied by the students and for simplification of polynomials 21 computers were occupied.

The sessions lasted one hour. During the first five minutes we demonstrated T-algebra and the solution processes in T-algebra and wrote our general dialogue scheme on the blackboard. In the first ten minutes, the students asked questions concerning the use of the computer (keyboard), the use of T-algebra tools (how to mark the objects and what to enter into the boxes), and mathematical questions about the solution steps. After that, questions concerning the use of software disappeared. Questions about mathematics (on operations with fractions and polynomials) continued after the first ten minutes. Questions relating to which rule to select in the menu continued to be asked throughout the trial. At that time, the concrete problem types were not yet implemented in our program and the menu contained all the rules needed for the actual topic. In most cases, the students even knew how they wanted to change the expression but they were often unable to find the name of the necessary operation. It is clear that we should pay attention to this issue when preparing the teachers for using rule-based software.

We collected the records of this trial – files with data about errors made by the students – for further study. The collected data included initial expression, current expression, selected rule, marked objects, entered parts (in case of errors at the input stage) and any error messages shown to the student. We also had some notes taken by the observers during the trial (two mathematics teachers and the four authors of T-algebra). When reviewing the files containing the students' mistakes, we initially noticed that almost all the students had made mistakes in marking the objects for applying the rule. The reason was probably that the students did not understand how to use the software – how and which parts of the expressions had to be marked for applying the rules. The mistakes of this type occurred two or three times in the beginning and then disappeared. Almost all subsequent mistakes were due to a lack of mathematical knowledge

(how to calculate the result of applying the rule, arithmetic errors, etc.) – students made the same mistakes as they made in paper tests.

While observing the trial, we noticed that many students preferred to mark the objects of the rule before selecting the rule itself (despite the "Select the rule" instruction on the screen and the instruction "1. Select the rule. 2. Mark the operands. 3. Enter the result" on the blackboard). At that time, our program gave no opportunity for marking more than one part in the expression before the rule was selected – this confused some students and they asked questions about that. After the trial, we added the possibility to select objects for applying the rule before the rule itself is selected. Yet, hints on selection of objects become available only after selection of a rule.

4.2.1 Distribution of student mistakes between three stages of solution step

From the previous experiment on paper, we saw that in some rules up to 30% students make mistakes in choosing the correct transformation and objects for it (mistakes 1, 2, 6 in Table 2.1 and mistakes 8, 10 in Table 4.2). While calculating and writing the result of transformation, in some cases up to 50% students make errors (mistakes 3, 4, 5 in Table 2.1 and mistakes 9, 11, 12, 13 in Table 4.2). I tried to compare it with the distribution of student mistakes between stages of a solution step in T-algebra.

After the user interface trial with students, I collected information on student errors in T-algebra from the solutions. As already mentioned, 21 students from 8th grade participated in the polynomial test. In this trial, the students were given exactly the same problems as in paper tests (see Appendix A). We have collected the students' solutions (error logs) and I studied them: what mistakes were made by the students, what errors were made at each stage of solution step, how many errors can be diagnosed before the input stage, etc.

In the following tables I present some results of this study, grouped by the rules used. By the moment of trial, typical errors in T-algebra were not classified; therefore, I tried to figure out the common nature of mistake from the log files. These are different from the typical errors identified in the first experiment (some typical errors are combined, etc.) but allow comparing the distribution of errors between the stages of the step. I calculated the percentage of users who made this type of mistake and also listed the stage of the action-object-input scheme where T-algebra could diagnose this error. These tables do not reflect if the students made these mistakes more than once.

Table 4.4. Mistakes in combining like terms when solving problems in T-algebra

No	Nature of mistake	% of students	Stage
1	Unsuitable operation – rule combine like terms cannot be applied	19%	action
2	Unsuitable objects – selected objects are not like	42%	object
3	Unsuitable objects – selected monomials do not belong to the same sum, or one of the monomials is a part of product	28%	object
4	Mistake in calculating the coefficient of single monomial in the result	47%	input
5	Mistake in calculating the power of a variable in a single monomial in the result	23%	input
6	Mistake in calculating the sign before single monomial in the result	33%	input

Table 4.5. Mistakes in multiplying the monomials when solving problems in T-algebra

No	Nature of mistake	% of students	Stage
1	Unsuitable operation – rule multiply the polynomials cannot be applied	14%	action
2	Unsuitable objects selected	23%	object
3	Mistake in calculating the coefficient of single monomial in the result	29%	input
4	Mistake in calculating the power of a variable in a single monomial in the result	47%	input
5	Mistake in calculating the sign before single monomial in the result	10%	input

Table 4.6. Mistakes in raising the monomial to a power when solving problems in T-algebra

No	Nature of mistake	% of students	Stage
1	Unsuitable operation – rule raise monomial to a power cannot be applied	29%	action
2	Unsuitable objects selected	19%	object
3	Mistake in calculating the coefficient of single monomial in the result	10%	input
4	Mistake in calculating the power of a variable in a single monomial in the result	28%	input
5	Mistake in calculating the sign before single monomial in the result	10%	input

At that moment, T-algebra did not diagnose the possibility of applying the selected rule separately after the rule was selected. The user had a possibility to

change the selected rule and confirmed this selection together with the selection of objects – therefore, T-algebra actually checks that after the object stage.

I also tried to compare (for one rule, *Combine like terms*) distributions of errors between the stages of a solution step on paper and in T-algebra. As students from 8th grade participated in the test with T-algebra, we took only the paper test of 8th grade (in fact, the same students) for comparison. In T-algebra, we can say exactly at what stage an error was made, but on paper it is very difficult to distinguish whether a mistake was made at the Action or the Object stage, because we do not have explicit information on students' thoughts (the students do not write the operation and often do not mark the objects for operation). This is why we combined the first two stages together in the following table.

Table 4.7. Distribution of mistakes in combining like terms on paper and in T-algebra

Stage	Paper	T-algebra
Action-Object	65%	55% (11% Action + 44% Object)
Input	35%	45%

From this comparison, we can see that checking errors at early stages is as important as checking errors in the input of the result. In both, paper solutions and in T-algebra, more than half of errors were caused by wrong selection of rule or objects. Therefore, the use of the action-object-input scheme where the user has to explicitly select the rule and the objects, with appropriate checks before the input of result, could be useful.

4.2.2 Conclusions

Summarising the results of the first user interface trial with T-algebra, we can say that the time required for learning the dialogue stages is quite short. In the first hour with T-algebra, most of the students had solved the same number of problems that were given to them in paper sessions. However, unlike in the paper tests, the students corrected all the mistakes they made. Error messages shown by the program were clear enough for the students to correct the mistakes. Different input modes of different rules were tested during the trial – all input modes were found useful. When solving the problems, no questions were asked on why all three stages of the dialogue are needed; the idea of the first two stages was clear to the students. All the students (even the weakest in mathematics) were using the program with great interest. A possible reason is that it was something new and different from ordinary school lessons. After the experiment, the students were asked how they liked the software – most of them answered that, "the program was great".

The Action – Object – Input scheme and error diagnosis after each step was found useful. First of all, we make the students learn the names of operations. T-algebra is able to diagnose errors before the result of transformation is entered, thereby preventing unnecessary computation and input by students.

4.3 Trial with T-algebra while explaining new material

In November 2005, we organized one more trial, this time in one class of so-called "difficult" children, who were studying in the 8th grade for the second year. There were 15 students and they had 45 minutes to try T-algebra. The topic was addition and subtraction of polynomials. We did not plan this trial in advance – the class teacher Mart Oja who was also engaged in the development of T-algebra wanted to try it when explaining the new material. The problem set contained 20 problems: 5 problems on combining like terms (this topic was already covered before) and 15 problems covering a new topic (5 on addition of polynomials, 5 on subtraction of polynomials, and 5 problems combining both addition and subtraction). Their teacher usually prepares the same number of problems for the pencil and paper work in the same topic.

During the first 5 minutes, the students were demonstrated the T-algebra and the solution process – the first problem (combine like terms) was solved by the teacher from the beginning till the end. Then they were given 10 minutes to complete four other combining problems. When solving these problems, students apply either the rule *Combine like terms* or the rule *Add/Subtract numbers*. After an introduction to T-algebra and solving the first 5 problems, the teacher explained the new rule – *Clear parentheses*. He solved one problem on the blackboard and after that the students solved the remaining fifteen problems (based on the new material) in T-algebra by themselves. By the end of the trial, almost everyone had solved all the problems; the students were solving the problems with great interest, although mathematics is not one of their favourite subjects. As the set of possible rules was limited, the students did not have difficulties in selecting the correct rules.

In this session, we saw that when the students made a mistake and the program displayed an error message, many of them were closing the message window without reading the diagnosis. They were then unable to correct the error and they even thought that their result was the correct one. Therefore, we have now added a small delay for the error messages – students cannot close the window for the first 3 seconds and some of them will now probably read the message. In the teacher's opinion, the ability of the students to recognize like terms in expressions improved after this session. He thought it was probably because they had to mark like terms explicitly when working in T-algebra. The use of T-algebra helped the teacher – it reduced his workload in correcting the students' solutions and all the errors made by the students were corrected (compared to pencil and paper work, where some errors remain uncorrected on the paper).

4.4 Study of student mistakes in T-algebra

In March 2009, we organized one more trial with T-algebra. This time one class of 11th grade students (31 students) came to the university computer lab to try T-algebra. This class was specialized in math (they had more than average math lessons per week). The material for the trial was learnt 3 years ago and actually those transformation rules are used in many other topics.

This was the first time when the students had seen T-algebra. Therefore, we first demonstrated T-algebra, then the students were given the first problem file (a set of 24 problems from different fields implemented in T-algebra) to try and learn to use T-algebra (for one hour). After that the students were given the second problem file (a set of 46 problems from the field of exponents and monomials, see Appendix B for the complete list) and one hour to complete the test. The problems were quite short, most of the problems required application of one main operation only (and some simplifications if needed). The problems used were actually the same as in paper tests of the first experiment. During the test students were still able to use help (but not generate automatic solutions). The free input mode was used.

After the experiment we collected student solution files and studied them. First, we collected some statistical data. Not all problems were solved by all students (on average, 40 problems per student were solved), although there was time left. The average time spent on solving problems was 41 minutes (out of 60 minutes given, maximum time spent was 47 minutes). After some of the first students completed the test and started to leave the room, some others ended their test and did not solve all problems (this is the reason why the average is less than 60 minutes, even though all problems were not solved).

Although the students were able to use help features of T-algebra, there were only a few cases of usage (meaning that the students tried to solve problems themselves). Only one student used the help function 80 times (he used 96% automatic filling-in of the result) – that student was the first to complete the test (only 26 minutes) but still made more than average number of mistakes. Other students used help features less than 10 times (11 students) or did not use them at all. Therefore, in order to be more objective, we excluded the student who used too much help from any further mistake studies and statistics.

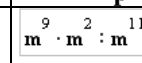
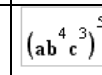
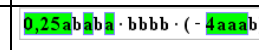
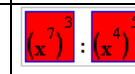
We thoroughly studied the mistakes of 30 students (the results of 1 student were not included, as he used help features too many times). A total of 739 mistakes were made (which makes 24 mistakes per student on average). It is very difficult to compare the number of mistakes with the number of mistakes made on paper, as on paper we usually check until we find the first error in a solution but in T-algebra students can make many errors on each step before it is correct (as T-algebra does not allow to proceed until errors are corrected). Out of these mistakes, T-algebra classified 534 mistakes (17.8 in average per student) as “mathematical” (where the possible cause had a mathematical background, for example, calculation error, and not incorrect use of the user interface, etc.). We studied all the errors and tried to divide those into

categories. We used a slightly different classification for errors from that used in T-algebra. In T-algebra the same classification is used for all topics; therefore, categories are quite general (for example, calculation errors, etc.). Here we used more specific categories for simplification problems and even for certain operations.

Table 4.8 contains a description of mistake categories with examples of mistakes (in further tables, we used short category names from the first column). Table 4.9 contains some general statistical data from student solutions (number of completed problems, spent time, number of help usages and number of mistakes made by student, both per student and total / average values). Table 4.10 contains information about mistakes (by student, total) grouped by categories that we have studied and discussed further. The table contains at least the following columns:

- total (total number of mistakes for each category),
- percentage of this mistake out of all mistakes,
- average number of mistakes of this category per student,
- number of students who made a mistake from this category at least once,
- percentage of students who have made errors from this category.

Table 4.8. Statistics grouped by different nature of mistake

Category	Name	Examples of mistake	Expression
ACTION1	Selected unsuitable operation	Student selects the rule “Combine like terms”	
ACTION2	Tries to apply the monomial multiplication rule to a power of monomial	Student selects the rule “Multiply monomials”	
OBJECT1	Syntactical mistake in marking	Student selects incorrect part of expression, for example, only one parenthesis, etc.	
OBJECT2	Mistakes in cancelling selection	Student tries to deselect object but does not mark anything	
OBJECT3	Objects not selected	Student does not select any object for the rule and tries to proceed	
OBJECT4	Selection errors related to parentheses	Student selects objects from different levels for one operation	
OBJECT5	Parallel application of "Raise monomial to a power"	Student selects more than one / one group of objects for operation	

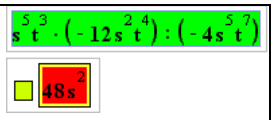
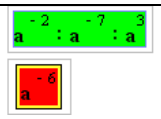
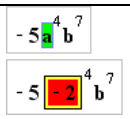
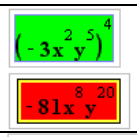
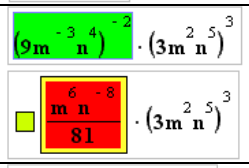
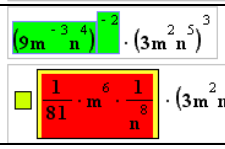
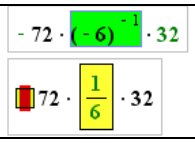
INPUT1	Calculation error (coefficient)	Student calculates a coefficient of a monomial incorrectly (here actually he had to divide 12 by 4, not multiply)	
INPUT2	Calculation error (power)	Student calculates a power of a variable incorrectly (here the student does not take into account division and simply adds all powers)	
INPUT3	Syntactical mistake in entering result	Here, instead of power 7, the student enters it to the same level	
INPUT4	Sign or parentheses missing	Here the student replaces a variable with its value. As he did not add parentheses, the expression is changed	
INPUT5	Error in sign	Student calculates the sign of coefficient or power incorrectly	
INPUT6	Mistakes in the form of monomial	According to definition, a monomial should not contain fractional parts (only coefficient can be fraction)	
INPUT7	Power in denominator instead of negative power	The student moves powers of variables to denominator	
OTHER1	Offers an unsimplified answer	The student tries to give answer by confirming an expression that can still be simplified somehow	
OTHER2	Error in T-algebra implementation	Here T-algebra required entering of sign (-). It is actually (+) and is not needed here.	

Table 4.9. General statistics from student solutions

Student	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	Total	Average
Problems completed	40	43	40	40	31	44	39	38	40	37	39	42	37	30	38	40	46	46	37	35	43	46	40	45	41	43	45	40	38	37		**40**
Spent time	40:13	45:02	44:16	42:30	42:02	42:15	43:03	45:37	47:28	42:05	44:05	41:25	45:20	36:33	44:38	46:41	41:35	42:32	42:32	43:19	43:37	45:51	45:33	46:02	46:57	40:14	45:44	42:10	41:41	43:15		**43:29**
Number of help usages	0	0	0	1	0	9	4	0	0	1	0	0	0	0	1	2	12	3	0	0	1	0	0	0	0	2	0	0	0	7	**43**	**1,43**
Number of mistakes	12	26	7	16	18	14	34	38	15	32	13	31	27	20	28	44	28	29	25	28	18	20	19	44	17	46	43	13	14	20	**739**	**24,63**
Mathematical mistakes	12	16	5	14	11	9	28	30	14	22	9	27	21	15	18	36	25	18	14	13	13	14	16	26	14	35	26	8	13	12	**534**	**17,80**

Table 4.10. Statistics grouped by different nature of mistake

Student		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	Total	% of mistakes	Average	Students	% of students
Number of mistakes		12	26	7	16	18	14	34	38	15	32	13	31	27	20	28	44	28	29	25	28	18	20	19	44	17	46	43	13	14	20	**739**		24,6		
Act	ACTION1	5			4	2	2	10	7	3	11		4	6	1	5	15	9	6	7	4	1	3	5	6	6	6	16		2	8	**154**	**20,8**	5,9	26	86,7
	ACTION2		1	1	1		1		2		1		7	2		1		2	1					2	3	1	2	1				**29**	**3,9**	1,8	16	53,3
Object	OBJECT1		4	1				1			1			2					2	1	5						3	2	2			**24**	**3,2**	2,2	11	36,7
	OBJECT2		3								3			1		2																**9**	**1,2**	2,3	4	13,3
	OBJECT3		2		2	4		2									3				6			1								**20**	**2,7**	2,9	7	23,3
	OBJECT4		1			1					1			2			1			1	1				1			1		1		**11**	**1,5**	1,1	10	33,3
	OBJECT5							1		1	1	1			1				2	1	1		1	1		1	1		1	1		**15**	**2,0**	1,1	14	46,7
Input	INPUT1	1	5			1	1	3	7			1	3	6	1	4	7	3	2	3		4	2	2	6	1	3	4		4	2	**76**	**10,3**	3,2	24	80,0
	INPUT2	1		1	2	3	3	5	2	3	5	5	2	1	3	3	7	3		1	3	2	2	3	4	1	9	2		2	2	**80**	**10,8**	3,0	27	90,0
	INPUT3			1	1							1		2		1			2		1	1			3		4			1	2	**20**	**2,7**	1,7	12	40,0
	INPUT4		3				2		1			1	2						2			2	1		1		1	4	1			**21**	**2,8**	1,8	12	40,0
	INPUT5		1		1	1	2	2	3	3	2	1	2	2			5	1	3			1	1	2	1	1	3			1	1	**40**	**5,4**	1,8	22	73,3
	INPUT6	2	2	3	4	3	1	1					2		5		1	1				1	2	1		1	1	4	3			**38**	**5,1**	2,1	18	60,0
	INPUT7		3										3										1		2	2				1		**12**	**1,6**	2,0	6	20,0
Oth	OTHER1	1			1			6	6	1	2	1	3	2	3		4	2	1	1	3	3	3	1	4	1	5			1	1	**56**	**7,6**	2,4	23	76,7
	OTHER2		1				1						2					3	2			1	1				1	4				**16**	**2,2**	1,8	9	30,0
Total classified		10	26	7	16	15	13	28	28	10	26	10	30	26	14	16	40	24	23	15	18	16	17	17	31	15	39	38	7	14	16	605		20,2		
Unclassified		2	0	0	0	3	1	6	10	5	6	3	1	1	6	12	4	4	6	10	10	2	3	2	13	2	7	5	6	0	4	134		4,5		

We have reviewed some topmost categories of errors and tried to explain the numbers or thought of some ways to improve T-algebra to respond better to those mistakes. Students made quite many mistakes in solutions. Of course, some are caused by the user interface and certain special requirements of T-algebra, but there were also quite many actual mistakes. We have mentioned that students preferred to work fast, even if that caused extra errors due to oversight.

Most of the problems in the test required application of one main transformation rule only and its name was also quite similar to the text of the problem (for example, "multiply monomials", etc.). Nevertheless, as we can see from Table 4.10, students had difficulties in choosing the correct (applicable) rule (the most frequent mistake, made 154 times (21% of all mistakes) by 26 students (87% of all students)). In many cases, we have seen that students were just mixing terminology (combine or collect VS reduce, multiply numbers VS multiply monomials, etc.). T-algebra did not diagnose separately for a selected rule whether that rule was applicable – it only checked selection of the rule together with objects. As we could see, students selected an incorrect rule that could not be applied and tried to guess a different set of objects for the rule or to select the same objects in a different form, for example, selecting a monomial with sign and without the sign, etc. (making 3-4 mistakes in a row), before realizing that this was a wrong rule. Similar results were actually seen during the first trial (Table 4.4, Table 4.5 and Table 4.6). After this trial, we implemented the check for selection of an unsuitable rule in T-algebra.

The next most frequent mistakes were calculation errors: mistakes in calculating the power of a variable (total 80 (11%) mistakes made by 27 (90%) students) and mistakes in calculating the coefficient of a monomial (total 76 (10%) mistakes made by 24 (80%) students). Sign errors are also worth mentioning (total 40 (5%) mistakes made by 22 (73%) students). Such errors are made by students on paper as well, so there is nothing strange in similar results in T-algebra (compare, for example, with those in Table 2.1, up to 48% of students made calculation errors).

Another typical error (total 56 (8%) mistakes made by 23 (77%) students) is offering an unfinished (un-simplified) solution as an answer, which is also quite frequent on paper. Students perform the main operation in the problem, leave the result as it is, and do not simplify it further. When checking solutions on paper, teachers are not so strict with these errors because there are no common rules on what exactly needs to be simplified. T-algebra is stricter and it makes students think what else can be simplified.

Another common mistake we collected was error in the form of monomial (total 50 (7%) mistakes made by 20 (67%) students). Students tried to enter monomials that did not match the definition (for example, $\frac{2x^2y^3}{5}$ instead of

$\frac{2}{5}x^2y^3$ or $\frac{2x^2}{y^3}$ instead of $2x^2y^{-3}$, etc.). This can be explained by the fact that we had students of 11th grade, who were already used to more difficult tasks and to having more freedom in applying rules. However, T-algebra required strict application of the rule (as it is taught in 7th and 8th grades) and inputting the result as one monomial (according to definition). Actually, these mistakes could be avoided if another input mode would be used for the problems – in the structured and partial input modes, the structure for inputting the monomial is given.

Another typical error we have also seen in paper tests is that students try to apply the monomial multiplication rule incorrectly (only 29 (4%) mistakes but made by 16 (53%) students). When multiplying monomials, students try to multiply not only coefficients but also powers of variables (instead of adding) – this is probably caused by the influence of other rules learned subsequently (raising monomial to a power when it is really needed to multiply powers). Quite many students make that mistake, but usually once or twice – after that they remember how to apply rule correctly. This is a positive effect of using T-algebra with immediate feedback to students about errors – they learn by making mistakes and getting feedback.

Other mistakes appeared less frequently but still worth mentioning are such mistakes that are caused by the usage of T-algebra and would not be possible on paper. Those are, for example, different mistakes connected to selection of objects (syntactical error, UI problems when selecting / deselecting objects, trying to apply a rule with no objects selected, total 40 (5%) mistakes, made by 11 (37%) students), incorrect selection of object connected to powers and parentheses (need to select both power and expression, do not need to select parentheses when doing operations inside them, etc., total 11 (2%) mistakes, made by 10 (33%) students). In addition, we have mentioned that quite many students were trying parallel application of the rules (raise 2 monomials to a power in one solutions step), as they would probably do on paper (15 (2%) mistakes, made by 14 (47%) students). Such mistakes are not frequent (only 1–2 mistakes per student, less than 5% of all mistakes, but made by many students) and were probably caused by the fact that the students were new to T-algebra – such errors should disappear after some experience. Teachers may also consider explaining those issues to students better before they start working. We think that such mistakes would not be made after this trial anymore as they were made 1-2 times only (the students learned that parallel application of rules is not possible, understood how object selection works in the UI, etc.).

As an additional goal of this trial, we wanted to test T-algebra with a larger group of students under our control (because other users may not report all errors, even if they find those) to check if there are still some errors in implementation. This goal was successfully achieved. We were able to figure out some minor errors in solution algorithm, implementation of some rules and

found a possibility for students to enter an incorrect response in the free input mode. Those were fixed after the experiment.

4.4.1 Conclusions

As a result of this trial, we have shown that students make the same typical mistakes in T-algebra as on paper. In addition, students quite often selected unsuitable rules and we added the possibility for T-algebra to diagnose such errors and inform the student. In the latest version of T-algebra, when checking for objects selected by the student, T-algebra first checks if the rule is applicable at all (maybe to some other parameters) and only then starts checking the objects selected by the student. Through this check of non-applicable rules, a positive effect of using the environment would be knowledge of the names of operations.

We have mentioned some positive effects of using T-algebra and giving immediate feedback to the students when making mistakes. We identified certain groups of errors that were made by many students, but only once or twice, mostly on the first usage of a rule. It means that the students learned from the feedback and did not make the same mistakes in further problems.

In addition, we have seen that the amount of errors caused by the use of T-algebra (various UI usage errors and T-algebra specific restrictions) was minimal (around 5%) even during first trial and would probably almost disappear during further usage. Some restrictions caused more errors (for example, 7% in the form of monomial) but this can be altered if a different input mode is used. Furthermore, those errors would not appear if T-algebra were used all the time during teaching new material.

In this study, we tried to use a different error categorisation (more field specific) for most common mistakes. We observed the mistakes and created a new set of categories. However, we later found that almost all categories that we identified are actually separate error categories in T-algebra as well. Only some very field specific errors (for example, ACTION1 and ACTION2, OBJECT5 and INPUT7) required categories different from those identified by T-algebra (those are special cases, like subcategories for categories in T-algebra). Therefore, we can conclude that categorisation in T-algebra is quite useful to some extent for reviewing errors.

As a bonus of testing the environment with a larger group of students, we were able to detect some errors in the implementation of transformation rules and correct them.

CONCLUSIONS

This thesis is based on the work that has been done for the T-algebra project. The main goal of the project was to create an interactive environment for simplification problem solving in four fields of school mathematics and algebra:

- calculation of the values of numerical expressions;
- operations with fractions;
- solving of linear equations, inequalities and linear equation systems;
- operations with exponents, monomials and polynomials.

Prior to making any decisions and implementations, we studied related works: existing software and different solution step approaches used in them. We studied experiments related to student errors in the fields of mathematics that we decided to implement in T-algebra and also conducted our own experiments. As a result, we formulated the requirements and key features for T-algebra:

- enable students to solve problems step-by-step and line-by-line in a manner similar to solving problems on paper;
- allow students to make all the necessary decisions and calculations at each solution step and explicitly provide this information to the system;
- leave an opportunity for students to make the same mistakes as on paper;
- give the possibility to learn both the algorithms and their steps in detail;
- include such dialogue that allows the program to understand all decisions made by students (collect direct information about chosen operation, selected operands, entered result);
- contain such domain expert module, which would be able to not only give an answer, but to show a solution path using the designed interface;
- be intelligent enough to check the knowledge and skills of students, understand mistakes, offer feedback and advice.

These main requirements were achieved by designing and implementing a special three-stage solution step dialogue (called action-object-input scheme) and also other key components to support this scheme (for example, an expression editor). We improved the input stage with three different input modes and extended the dialogue with additional steps for some rules. Solving problems in T-algebra by making steps according to the dialogue is very similar to solving problems on paper. A student has to make all decisions himself and also has the opportunity to make various errors. However, in comparison to solving on paper, T-algebra is able to assist the student and perform some steps automatically if the student is lost. Furthermore, T-algebra is able to diagnose student errors and provide feedback. Based on the information entered by the student at different stages of the steps, it is possible to make quite adequate diagnosis of student errors and, in future developments, possibly to diagnose misconceptions with very small amount of guesswork and computational efforts.

The solution step dialogue, design of different field specific transformation rules and problem types as well as support for adequate error diagnosis are key attributes that distinguish T-algebra from other similar environments. In addition to those, T-algebra includes a cognitively faithful domain expert that provides hints and generates automatic solutions to problems corresponding to algorithms taught in classroom.

This thesis describes different aspects of the created system, design decisions of different components and some implementation details. These also include parts mainly contributed by the author of the thesis, which could be summarized in three large parts:

- design decisions and implementation of some general T-algebra features;
- study, design decisions and implementation of problem types and rules for a specific domain – the domain of exponents, monomials and polynomials;
- experimenting efforts in evaluating the general features, like solution dialogue of T-algebra, as well as domain specific decisions, problem types, transformation rules, etc.

When designing and developing T-algebra, some features were designed and implemented mainly by the author of thesis:

- participation in design and implementation (project seminars with school teachers and authors of school textbooks) of the action-object-input solution step dialogue (presented in section 2.5);
- implementation of expression parsing and rendering in the expression editor;
- design and implementation of expression editor features to support the solution step dialogue (presented in section 2.7);
- design and implementation of an extension to the action-object-input dialogue (presented in section 2.6);
- design and implementation of the general principle of error diagnosis and categorisation (presented in sections 2.8.6 and 2.9);
- internal design and implementation of the basic classes of rule and problem type and their usage in general solution algorithm, error diagnosis, etc. (presented in sections 3.2.2 and 3.3.1).

One of the main contributions of the author was domain specific part of the system for the domain of exponents, monomials and polynomials. This included the following tasks, which are thoroughly described in the thesis:

- study of problems solved at school in the chosen domain and design of problem types for T-algebra (presented in sections 3.1 and 3.3);
- study of school textbooks and student solutions in order to extract the transformation rules needed for this domain (both domain specific and learned before), design of transformation rules in T-algebra, discussion of

the design with school teachers, and publication of decisions (presented in sections 3.1 and 4.1);
- investigation of the typical errors for the selected domain, based on experiments with students (presented in section 4.1) and related works in order to design error diagnosis for the designed transformation rules (presented in section 2.8.6);
- implementation of identified problem types, including error diagnosis, conditions for starting and ending expressions, solution algorithm, etc. (presented in section 3.3);
- implementation of domain specific transformation rules, including error diagnosis and a domain expert for application of implemented rules (presented in section 3.2).

The author of the thesis participated in numerous experiments and trials with students and teachers (results are presented in Chapter 4):
- experimental validation of created dialogues with students and teachers (presented in sections 4.2 and 4.3);
- evaluation of the environment in the chosen domain of exponents, monomials and polynomials, trials with real students (presented in sections 4.2 and 4.3);
- investigation of student solutions and their mistakes when solving problems in T-algebra (particularly problems of the chosen domain) and comparison with the results of the experiment of collecting mistakes from paper solutions (presented in section 4.4).

The main result of the experiments with students was that the introduced solution step dialogue was easy to learn and use, error diagnosis and messages were helpful. The latest study of student errors in T-algebra gave us positive results. However, we definitely cannot make judgements about the usability and effectiveness of the created environment, based on the results of brief experiments because, at this stage, results can be influenced by the novelty of the program for students and teachers. Teachers need to experiment with different ways of using the system, such as explaining new material, self-study by students, rehearsing old material, and assessment. The development of the current version of T-algebra was completed in 2009 and the environment is now available to all schools in Estonia.

Finally, we have identified some future development possibilities for the T-algebra environment.

First of all, we could implement an even more refined error diagnosis where different typical misconceptions are identified. In the current version of T-algebra, in some transformation rules, the diagnosis in the structured and partial input modes is more detailed than in the free input mode. We could implement a more detailed diagnosis for the free input mode as well, which would add some constraints and reduce the freedom of students when entering

step results. For example, at the moment, when multiplying two polynomials in the free input mode, it is possible to enter a partial multiplication result – the sum of products with monomials (members of the first polynomial) and the second polynomial. Or it is possible to combine like terms in the result of multiplication. In the structured and partial input modes, T-algebra requires the student to enter the result of multiplication as a polynomial. Thus, one additional constraint, that the resulting subexpression has to be a polynomial, would give us the same result in this case.

Secondly, we could improve the student-modelling component, which would reflect the system's understanding of students' conceptions and misconceptions and would change in the course problem solving. This component could be used in automatic assessment of students and, for example, in random problem generation to provide the student with expressions that have been most problematic for that student.

We could also implement a students' tutoring module for T-algebra, which would contain some explanations and examples of all simplification rules, and presentations of solution algorithms for all implemented problem types. This would allow us to upgrade T-algebra from a task oriented system to a fully qualified intelligent tutoring system.

Finally, we could implement certain automatic components that would allow us to collect statistics on some larger groups of students in a central storage, analyse it and use it for future experiments (Prank and Lepp, 2010).

REFERENCES

Alessi, S.M. and Trollip, S.R. (2001). Multimedia for learning: Methods and development. 3rd ed., Allyn & Bacon.

Alessi, S.M. and Trollip, S.R. (1991). Computer-based instruction: Methods and development. 2nd ed., Prentice Hall

Alpert, S.R., Singley, M.K. and Fairweather, P.G. (1999). Deploying Intelligent Tutors on the Web: An Architecture and an Example. *International Journal of Artificial Intelligence in Education*, 10(2): 183–197.

Anderson, J.R., Corbett, A.T., Koedinger, K. and Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of Learning Sciences* 4: 167–207.

Anderson, J.R., Boyle, C.F., Corbett, A. and Lewis, M.W. (1990). Cognitive modelling and intelligent tutoring. *Artificial Intelligence*, 42: 7–49.

Barnett, R.A. and Kearns, T.J. (1990). *Intermediate Algebra: Structure and Use*. 4th ed. McGraw-Hill.

Beeson, M. (2002). MathXpert: un logiciel pour aider les élèves à apprendre les mathématiques par l'action. *Sciences et Techniques Educatives*, 9(1–2). English translation 'MathXpert: Learning Mathematics in the 21st Century' available at http://www.mathcs.sjsu.edu/faculty/beeson/Papers/English-ste/English-ste.html. Visited on 24.03.2010.

Beeson, M. (1998). Design Principles of Mathpert: Software to support education in algebra and calculus. In *Computer-Human Interaction in Symbolic Computation*, pp. 89–115, Springer-Verlag.

Beeson, M. (1990). Mathpert: a computerized learning environment for algebra, trigonometry, and calculus. *International Journal of Artificial Intelligence in Education*, 1(4): 65–76.

Blackboard Learning System by Blackboard Inc. Available at http://www.blackboard.com/. Visited on 28.03.2010.

Brown, J.S. (1985). Process versus Product: A perspective on tools for communal and informal electronic learning. *Journal of Educational Computing Research*, 1: 179–201.

Brown, J.S. and Burton, R.R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 2: 155–192.

Burton, R.R. (1982). Diagnosing bugs in a simple procedural skill. In *Intelligent Tutoring Systems*, pp. 157–183, Academic Press.

Cerulli, M. and Mariotti, M.A. (2002). L'Algebrista: un micromonde pour l'enseignement et l'apprentissage de l'algèbre. *Science et techniques éducatives*, 9: 149–170.

Cognitive Tutor by Carnegie Learning, Inc. Available at http://www.carnegielearning.com/. Visited on 02.05.2010.

Equation Wizard by ElasticLogic. Available at http://www.equationwizard.com/. Visited on 29.03.2010.

Hall, R. (2002). An Analysis of Errors Made in the Solution of Simple Linear Equations. *Philosophy of Mathematics Education Journal*, 15, 2002.

Handal, B. and Herrington, A. (2003). Re-examining categories of computer-based learning in mathematics education. In *Contemporary Issues in Technology and Teacher Education* [Online serial], 3(3).

Heffernan, N.T. and Koedinger, K.R. (2000). Intelligent Tutoring Systems are Missing the Tutor: Building a More Strategic Dialog-Based Tutor. In *Proceedings of the*

AAAI Fall Symposium on Building Dialogue Systems for Tutorial Applications, pp. 14–19.

Holland, G. (1994). Intelligent Tutorial Systems. In *Didactics of Mathematics as a Scientific Discipline*, pp. 213–223.

Issakova, M. (2006). Domain Expert Module for Step-By-Step Linear Equation Solving. In *Proceedings of The 11th Asian Technology Conference in Mathematics* (ATCM 2006), pp. 193–202, Hong Kong, China.

Issakova, M. (2005). Possible Mistakes During Linear Equation Solving On Paper And In T-algebra Environment. In *Proceedings of the 7th International Conference on Technology in Mathematics Teaching*, Volume 1, pp. 250–258. Bristol, UK.

Issakova, M. and Lepp, D. (2004). Rule dialogue in problem solving environment T-algebra. In *Proceedings TIME–2004: Montreal International Symposium on Technology and its Integration into Mathematics Education*, 16 p., Montreal, Canada.

Issakova, M., Lepp, D. and Prank, R. (2006). T-algebra: Adding Input Stage To Rule-Based Interface For Expression Manipulation. *International Journal for Technology in Mathematics Education*, 13(2): 89–96.

Issakova, M., Lepp, D. and Prank, R. (2005). Input Design in Interactive Learning Environment T-algebra. In *Proceedings ICALT–2005: The 5th IEEE International Conference on Advanced Learning Technologies*, pp. 489–491, Kaohsiung, Taiwan.

Klai, S., Kolokolnikov, T. and Van den Bergh, N. (2000). Using Maple and the web to grade mathematics tests. In *Proceedings of the International Workshop on Advanced Learning Technologies*.

Kutzler B., (2000): The Algebraic Calculator as a Pedagogical Tool for Teaching Mathematics. In *International Journal of Computer Algebra in Mathematics Education*, v7 n1, pp. 5–23.

Kutzler, B. (1996). Improving Mathematics Teaching with DERIVE. Chartwell-Bratt Publishing & Training Ltd.

Lepik, M., Nurk, E., Telgmaa, A. and Undusk, A. (2000). *Mathematics for VIII grade* (in Estonian). Koolibri.

Lepp, D. (2007a). Study of Student Mistakes in Solving Simplification Problems on Paper and Possibility of these Mistakes in the T-algebra. In *Proceedings of the 8th International Conference on Technology in Mathematics Teaching* (ICTMT8), 6p., Hradec Králové, Czech Republic.

Lepp, D. (2007b). Distribution of student mistakes between three stages of solution steps in case of Action-Object-Input solution scheme. In *Abstracts of First Central- and Eastern European Conference on Computer Algebra- and Dynamic Geometry Systems in Mathematics Education* (CADGME), p. 27, Pecs, Hungary.

Lepp, D. (2006a). Design of polynomial transformation rules in problem solving environment T-algebra. In *Proceedings DES–TIME–2006: Dresden International Symposium on Technology and its Integration into Mathematics Education 2006*, 15p., Dresden, Germany.

Lepp, D. (2006b). Error Diagnosis in Problem Solving Environment Using Action-Object-Input Scheme. In *ITS 2006 Proceedings*, LNCS 4053, pp. 769–771, Springer-Verlag.

Lepp, D. (2006c). Using Action-Object-Input Scheme for Error Diagnosis in Problem Solving Environment. In *Proceedings of the Student Track ITS 2006*, pp. 18–27, Jhongli, Taiwan.

Lepp, D. (2006d). Error Diagnosis and Categorization in Problem Solving Environment Using Action-Object-Input Scheme. In *Proceedings of The 11th Asian Technology Conference in Mathematics* (ATCM 2006), pp. 215–224, Hong Kong, China.

Lepp, D. (2005). Extended Solution Step Dialogue In Problem Solving Environment T-algebra. In *Proceedings of the 7th International Conference on Technology in Mathematics Teaching* (ICTMT7), volume 1, pp. 267–274, Bristol, UK.

Lepp, D. (2003a). Program for exercises on operations with polynomial. In *Proceedings of the 6th International Conference Technology in Mathematics Teaching* (ICTMT6), pp. 365–369.

Lepp, D. (2003b). Program "Polynom" for exercises on operations with polynomials (in Estonian). *Master thesis*.

Lepp, D., Issakova, M. and Vaiksaar, V. (2005). Expression Editor Features That Simplify Student Work On Manipulating Expressions. In *Proceedings of the 7th International Conference on Technology in Mathematics Teaching* (ICTMT7), volume 1, pp. 259–266, Bristol, UK.

Lewis, M. W., Milson, R., and Anderson, J. R. (1987). The teacher's apprentice: Designing an intelligent authoring system for high school mathematics. In *Artificial Intelligence and Instruction*, pp. 269–301, MA: Addison-Wesley, Reading.

Mavrikis, M. and Maciocia, A. (2003). Wallis: a web-based ILE for science and engineering students studying mathematics. In *Workshop of Advanced Technology for Mathematics Education in the 11th International Conference on Artificial Intelligence in Education*, pp. 505–512.

Maleševic, B. (2009). A way to improve the use of CAS for integration. In *6th CAME Symposium: Structured Abstracts*, pp. 17–18.

Maple by Maplesoft. Available at http://www.maplesoft.com/. Visited on 15.05.2010.

MathAid by MathAid. Available at http://www.mathaid.com/. Visited on 26.03.2010.

MathCAD by Parametric Technology Corporation (PTC). Available at http://www.ptc.com/products/mathcad/. Visited on 15.05.2010.

MathCentre by Mathematics Education Centre of Loughborough University. Available at http://www.mathcentre.ac.uk/. Visited on 25.03.2010.

Mathematics V10 by EptSoft. Available at http://www.eptsoft.com/Maths-HTML/Mathematics%20Index.htm. Visited on 27.03.2010.

Math-Teacher by MATH-KAL. Available at http://www.mathkalusa.com. Visited on 08.05.2010.

McKeague, C.P. (1979). *Intermediate Algebra*. Academic Press.

Nicaud, J.F. and Bouhineau, D. (2008). Natural Editing of Algebraic Expressions. In *Les Cahiers Leibniz*, volume 169, pages 1–15. Longer version of MathUI workshop paper, Linz, Austria, 2007.

Nicaud, J.F., Chaachoua, H. and Bittar, M. (2006). Automatic Calculation of Students' Conceptions in Elementary Algebra from Aplusix Log Files. In *ITS 2006 Proceedings*, LNCS 4053, pp. 433–442, Springer-Verlag.

Nicaud, J.F., Chaachoua, H., Bittar, M. and Bouhineau, D. (2005). Student's modelling with a lattice of conceptions in the domain of linear equations and inequations. In *Proceedings of AIED 05 Workshop 1: Usage Analysis in Learning Systems*, pp. 81–88.

Nicaud, J.F., Bouhineau, D. and Chaachoua, H. (2004). Mixing microworld and CAS features in building computer systems that help students learn algebra. In *International Journal of Computers for Mathematical Learning*, 5, pp. 169–211.

Nicaud, J.F., Bouhineau, D., Varlet, C. and Nguyen-Xuan, A. (1999). Towards a product for teaching formal algebra. In *Proceedings of Artificial Intelligence in Education*, pp. 207–217.

Nurk, E., Telgmaa, A. and Undusk, A. (2006). *Mathematics for VII grade* (in Estonian). Koolibri.

Pais, E. (2001). *Mathematics for VIII grade* (in Estonian). Avita.

Pais, E. (1998). *Mathematics for VII grade* (in Estonian). Avita.

Payne, S. and Squibb, H. (1990). Algebra malrules and cognitive accounts of errors. In *Cognitive Science*, 14, pp. 445–481.

Postel, F. (1999). MuPAD as a Tool, Tutee and Tutor. In *Proceedings of ACDCA summer academy*.

Prank, R. (1991). Using Computerised Exercises on Mathematical Logic. *Informatik-Fachberichte*, 292: 34–38, Springer-Verlag.

Prank, R. and Viira, H. (1991). Algebraic Manipulation Assistant for Propositional Logic. *Computerised Logic Teaching Bulletin*, 4(1): 13–18.

Prank, R. and Lepp, D. (2010). Tools for acquiring data about student work in interactive learning environment T-algebra. In *Proceedings of ITS 2010*, LNCS 6095, pp. 396–398.

Prank, R., Issakova, M., Lepp, D., Tõnisson, E. and Vaiksaar, V. (2007). Integrating rule-based and input-based approaches for better error diagnosis in expression manipulation tasks. In *Symbolic Computation and Education*, pp. 174–191, World Scientific Publishing Co.

Prank, R., Issakova, M., Lepp, D. and Vaiksaar, V. (2006a). Designing Next-Generation Training and Testing Environment for Expression Manipulation. In *International Conference on Computational Science* (ICCS 2006), Part I, LNCS 3991, pp. 928–931, Springer-Verlag.

Prank, R., Issakova, M., Lepp, D., Vaiksaar, V. and Tõnisson, E. (2006b). Problem solving environment T-algebra. In *Proceedings of 7th International Conference Teaching Mathematics: Retrospective and Perspectives*, pp. 190–197, Tartu, Estonia.

Ravaglia, R., Alper, T., Rozenfeld, M. and Suppes, P. (1998). Successful pedagogical applications of symbolic computation. In *Computer-Human Interaction in Symbolic Computation*, pp. 61–88, Springer-Verlag.

Sangwin, C. J. (2007). STACK: making many fine judgements rapidly. In *CAME*.

Sangwin, C.J. (2005). Making Mathematical Distinctions In CAA With Computer Algebra. In *Proceedings of the 7th International Conference on Technology in Mathematics Teaching* (ICTMT7), volume 1, pp. 292–299, Bristol, UK.

Stephens, L. J. and Konvalina, J. (1999). The use of computer algebra software in teaching intermediate and college algebra. In *International Journal of Mathematical Education in Science and Technology*, 30, 4, pp. 483–488.

Strickland, P., and Al-Jumeily, D. (1999). A Computer Algebra System for improving student's manipulation skills in Algebra. *The International Journal of Computer Algebra in Mathematics Education*, 6(1): 17–24.

Tall, D. and Rashidi Razali, M. (1993). Diagnosing Students' Difficulties in Learning Mathematics. In *Int J. Math Ed, Sci & Techn.*, 24 2, pp. 209–202.

Thompson, P. and Thompson, A. (1987). Computer presentations of structure in algebra. In *Proceedings of the Eleventh Annual Meeting of the International Group for the Psychology of Mathematics Education*, volume 1, pp. 248–254.

Trgalova, J., Bouhineau, D. and Nicaud J.F. (2009). An Analysis of Interactive Learning Environments for Arithmetic and Algebra Through an Integrative Perspective. *In International Journal of Computers for Mathematical Learning*, 14 3, pp. 299–331.

Tõnso, T. (2002). *Mathematics for VII grade* (in Estonian). Mathema.

Veelmaa, A. (2004). *Mathematics for VIII grade* (in Estonian). Mathema.

Weitz, R., Heffernan, N, Kodaganallur, V. and Rosenthal, D. (2007). The Distribution of Student Errors Across Schools: An Initial Study. In *Proceedings of the 13th International Conference on Artificial Intelligence in Education*, pp. 671–673.

Xambo, S., Eixarch, R., and Marques, D. (2002). WIRIS: An Internet platform for the teaching of mathematics in large educational communities. *Contributions to Science*, 2 (2): 269–276.

Zuckerman, M.M. (1976). *Intermediate algebra*. W. W. Norton & Company.

SUMMARY IN ESTONIAN

Astmete, üksliikmete ja hulkliikmete valdkonna lihtsustamisülesannete lahendamine interaktiivses õpikeskkonnas T-algebra

Antud väitekiri baseerub tööl, mis on tehtud T-algebra projekti raames. Projekti peamine eesmärk oli luua uut tüüpi interaktiivne teisendusülesannete lahendamise keskkond probleemide lahendamiseks neljas koolimatemaatika ja algebra valdkonnas:

- aritmeetilised operatsioonid ja avaldiste väärtuste arvutamine;
- tehted murdudega;
- lineaarvõrrandite, lineaarvõrratuste ja lineaarsete võrrandisüsteemide lahendamine;
- lihtsustamise ülesanded astmete, üksliikmete ning hulkliikmete teemas.

Süsteemi disaini ja loomise etappidele eelnes olemasolevate sarnaste programmide analüüs, me uurisime kasutajaliidesega võimaldatud sammude tegemise viise olemasolevates süsteemides. Samuti me uurisime erinevate eksperimentide tulemusi, kus uuriti õpilaste poolt tehtud vigu valitud matemaatika valdkondades ning korraldasime oma lahenduskatseid. Lisaks me uurisime koolides kasutatavad õpikud ja õpilaste kontrolltööde lahendusi, et korjata kokku kasutatavad teisendusreeglid ning tüüpilised ülesanded, mida lahendatakse koolis. Uuringute tulemusena me identifitseerisime probleemid olemasolevates süsteemides ja formuleerisime mitu olulist printsiipi, mida me jälgisime süsteemi disainimisel ja loomisel. Loodav süsteem

- võimaldab lahendada ülesandeid sammhaaval sarnaselt paberil lahendamisele;
- lubab õpilasel teha kõik arvutused ja otsused igal lahenduse sammul;
- võimaldab õpilastel teha samu tüüpilisi vigu nagu paberil töötades;
- annab võimaluse õppida ja harjutada nii lahendusalgoritme kui ka üksikute lahendussammude tegemist;
- kasutab sellist dialoogi sammude tegemisel, et süsteem on võimeline aru saama kõikidest õpilase otsustest (kogub infot valitud reegli, operandide ning sisestatud tulemuse kohta);
- omab sisseehitatud valdkonna eksperdi moodulit, mis annab võimaluse mitte ainult anda vastust, vaid genereerida ülesande lahenduskäik, kasutades samu reegleid, mida saab kasutada õpilane;
- oskab diagnoosida õpilase poolt tehtud vigu ning anda arusaadavat tagasisidet ja vajadusel aidata õpilasi soovitustega või automaatselt tehtavate sammude abil.

Analüüsi käigus me ei leidnud süsteeme, mis töötaksid täiel määral vastavalt eespool kirjeldatud printsiipidele. Ühte osa programmidest võib kasutada ainult lahendusalgoritmide õpetamisel, kuna nendega töötades puudub õpilastel võimalus teha vigu. Teised sobivad ainult teadmiste kontrolliks, kuna nad ei diagnoosi täpselt õpilaste vigu või ei paku abi lahendamisel.

Sammude tegemise dialoog on see, mis eristab T-algebrat teistest sarnastest süsteemidest. Tänu sellele õnnestub T-algebral täita põhiprintsiipe, mis on kirjeldatud eespool. Iga samm programmis koosneb kolmest etapist. Dialoogi prototüüp sai läbi proovitud minu magistritöös. T-algebra jaoks me parandasime ning täiendasime esialgset dialoogi: ühtlustasime sisestusetappi erinevate reeglite jaoks kolme erineva sisestusrežiimi väljatöötamisega ning täiendasime skeemi lisaetappidega mõnede reeglite puhul (lisainfo sisestamine, vaheetapi lisamine ning struktuuri laiendamine). Loodud dialoog annab võimaluse õppida ja harjutada ülesannete lahendamise strateegiaid ning samuti üksikute sammude tegemise tehnikat.

Ülesannete lahendamine T-algebras on sarnane ülesannete lahendamisega paberil. Iga sammu tegemisel rakendab õpilane ühte konkreetset teisendusreeglit, märgib objektid ning sisestab tulemuse. Ülesannete lahendamise ajal õpilane teeb kõik otsused lahenduskäigu kohta. T-algebra annab võimaluse teha samu vigu nagu paberil, kuid erinevalt paberist on võimeline diagnoosima õpilase vigu ning aitama õpilast üksikute sammude tegemisel. Enne sammu dialoogi ning vigade diagnoosi disainimist me korraldasime eksperimendid õpilastega, et välja selgitada tüüpilised vead, mida nad teevad paberil ning lisaks vaatasime ka teiste analoogsete uurimiste tulemusi. Tänu kolmesammulisele dialoogile T-algebral on olemas kogu eelinfo, mida õpilane valib ja sisestab alametappidel. See annab parema võimaluse diagnoosida vigu võrreldes süsteemidega, kus on olemas ainult info sisestatud sammu tulemuse kohta.

Väitekiri kirjeldab loodud süsteemi erinevaid aspekte, erinevate komponentide disainiotsuseid ning realisatsiooni detaile. Väitekirja autori panuse ja tulemused võib liigitada kolme kategooriasse:

- T-algebra üldiste funktsionaalsete komponentide disain ja realisatsioon;
- valdkonna spetsiifiliste ülesannete tüüpide ja teisendusreeglite disain ja realisatsioon (lihtsustamise ülesanded astmete, üksliikmete ning hulkliikmete teemas);
- osalemine eksperimentides ning nende korraldamine selleks, et hinnata nii T-algebra üldise lahenduse erinevaid aspekte kui ka valdkonna spetsiifilise disaini ja realisatsiooni otsuseid.

Väitekirja eraldi osades on kirjeldatud need süsteemi osad, mille loomise või disainimise eest vastutas väitekirja autor:

- osalemine sammu dialoogi (action-object-input) disainis ja realisatsioonis (projekti seminarid õpetajatega ja õpikute autoritega) (esitatud peatükis 2.5);
- avaldiste parsimine ja kuvamine redaktoris (esitatud peatükkides 2.7.1 ja 2.7.2);
- avaldiste redaktori ja sammu dialoogiga seotud funktsionaalsuse disain ja realisatsioon (esitatud peatükis 2.7);
- sammu tegemise dialoogi ja selle laienduste disain ja realisatsioon (esitatud peatükis 2.6);

- vigade diagnoosimise ja kategoriseerimise printsiip (esitatud peatükkides 2.8.6 ja 2.9);
- reeglite ja ülesannete tüüpide klasside sisemine disain ja realisatsioon ja nende rakendused automaatses lahendamise algoritmis, vigade diagnoosis jne. (esitatud peatükkides 3.2.2 ja 3.3.1).

Olulise osa tööst moodustavad teisendusreeglite ja ülesannete tüüpide üldise disaini ja realisatsiooni detailid ning minu teema (astmete, üksliikmete ja hulkliikmete lihtsustamisülesanded) reeglite ja tüüpide detailsed kirjeldused:

- valitud valdkonna erinevate tüüpülesannete väljaselgitamine ning nende disainimine ülesande tüüpidena T-algebras (esitatud peatükkides 3.1 ja 3.3);
- õpikute ja õpilaste lahenduste analüüs eesmärgiga koguda infot kasutatavate teisendusreeglite kohta antud valdkonna ülesannetes, reeglite disain T-algebras ja disaini otsuste arutamine õpetajatega (esitatud peatükkides 3.1 ja 4.1);
- valitud valdkonna tüüpiliste vigade kogumine ning nende arvestamine reeglite ja vastavate vigade diagnoosi disainis (esitatud peatükis 4.1);
- eelnevalt loetletud ülesannete tüüpide realiseerimine, sealhulgas vigade diagnoosi realisatsioon, algavaldise ja lõppavaldise tingimused, lahendamise algoritm jne. (esitatud peatükis 3.3);
- valitud valdkonna spetsiifiliste teisendusreeglite realiseerimine, sealhulgas vigade diagnoosi realisatsioon, valdkonna eksperti osa nende reeglite automaatse rakendamise kohta (esitatud peatükis 3.2).

Väitekirja autor korraldas mõned eksperimendid ja katsed T-algebraga ning osales kogu meeskonna poolt organiseeritud eksperimentides, selleks et hinnata T-algebra erinevaid aspekte (tulemused on esitatud väitekirja 4. osas):

- sammu dialoogi valideerimine õpilastega ja õpetajatega (esitatud peatükkides 4.2 ja 4.3);
- T-algebra hindamine valitud valdkonnas (tehted üksliikmetega), katsed õpilastega (esitatud peatükkides 4.2 ja 4.3);
- õpilaste ülesannete lahenduste ja tehtud vigade uuring ülesannete lahendamisel T-algebra-s (lihtsustamise ülesanded astmete, üksliikmete ning hulkliikmete teemas) ning tulemuste võrdlus paberitesti analüüsi tulemustega (esitatud peatükis 4.4).

Eksperimentide peamine tulemus on see, et loodud sammu tegemise skeem on kergesti omandatav õpilaste poolt, vigade diagnoos ja abiteated on arusaadavad ja aitavad õpilasi parandada vigu ja lahendada ülesandeid. Viimases eksperimendis me uurisime vigu, mida õpilased teevad T-algebras ja saime positiivseid tulemusi. Juba esimesel lahendamise sessioonil T-algebraga õpilased õppisid oma vigadest tänu kohesele diagnoosile ja programmi reaktsioonile. Me nägime mõnede vigade tüüpide korral, et õpilased tegid selliseid vigu üks või kaks korda ja hilisemates ülesannetes enam ei teinud sama tüüpi vigu. Loomulikult me ei saa hinnata T-algebra kasutatavust ja efektiivsust nende lühikeste

eksperimentide tulemusena. Õpilastele meeldis lahendada ülesanded keskkonnas, kuid T-algebra oli nende jaoks uus kogemus ja see võis mõjutada eksperimentide tulemusi. Osade õpetajate jaoks on T-algebra kasutamine õpetamisel samuti uus kogemus, paremate tulemuste saavutamiseks on vaja pikemat aega eksperimenteerida erinevate õpetamisviisidega ning luua piisavalt õppematerjale.

2009 aastal me lõpetasime jooksva T-algebra versiooni arenduse ning see on nüüd kättesaadav kõikides eesti koolides. Paljud õpetajad läbisid ka T-algebra alased koolitused ja kasutavad programmi õpetamisel. Me loodame saada õpetajatelt väärtuslikku tagasisidet, mis võiks anda infot võimalike vigade kohta ning ideed T-algebra edasiarendamiseks.

Siin võib tuua mõned võimalikud suunad süsteemi edasiseks arendamiseks:

- esiteks võiks disainida ja realiseerida veelgi täpsema vigade diagnoosi osade reeglite puhul;
- teiseks me võiksime realiseerida T-algebras mingisuguse variandi õpilase mudelist, mida saaks kasutada näiteks teadmiste hindamisel või ülesannete genereerimisel, selleks et pakkuda õpilasele lahendamiseks just selliseid probleeme, millega tal tekib raskusi;
- lisada T-algebrasse tuutori moodul, mis sisaldaks kõikide realiseeritud reeglite seletusi näidistega ning erinevate ülesannete tüüpide puhul lahendusalgoritmide kirjeldusi näidistega;
- T-algebrasse võiks lisada statistilise komponendi, mis lubaks korjata terve klassi õpilaste lahenduste ja vigade statistika andmed ja kasutada neid võimalikes uurimistes (Prank and Lepp, 2010).

ACKNOWLEDGEMENTS

I would like to thank my parents, my brother and especially my wife Marina Lepp, who gave me the opportunity to concentrate on my PhD thesis. Separate thanks to my son Kristjan, who encouraged me to finish the thesis. I would also like to thank all my friends for their support.

I wish to thank my supervisor Rein Prank for pleasant cooperation. I would like to thank mathematics teachers Mart Oja and Maire Oja for their co-operation in discussing important design decisions for T-algebra. I would separately like to thank all students for solving tests and participating in different experiments.

I also wish to thank Alar Helstein for correcting my English in this thesis.

During my PhD studies I was supported by the 'Tiger Leap' computerization programme for Estonian schools as they financed the T-algebra project. I was partially supported by the Estonian Science Foundation under grants 5272, 7180 and by the targeted financing project SF0182712s06 "The methods, environments, and applications for solving large and complex computational problems". I was also supported by the Estonian Information Technology Foundation and Estonian Doctoral School in Information and Communication Technologies.

APPENDIX A

Tests for 7th and 8th grades

The test for 7th grade included calculation problems as well as some easiest simplification problems. Simplification problems of variant A were the following (the subset that I used for analysis in my field):

- open parentheses: 1) $7(2x-y)$; 2) $(3m-4n+1)\cdot 5$; 3) $-(5m-3n)$;
  4) $-4(2x-3y+1)$; 5) $(3u-2v)\cdot(-3)$; 6) $a(2b-3c)$;
  7) $-m(3x+2y-4)$; 8) $0{,}2(-5x+4y-3)$;
- combine like terms: 1) $9c-7c$; 2) $7a+a$; 3) $3m-4m+2m$;
  4) $9-2x+4+3x-12+2x$; 5) $5n-3-6n+2-2n+3$;
  6) $3a-7b+a-3a+2b+4a-2b$;
  7) $4x-3y+9-5x+3y-3+x-6$.

The problems of variant B were the following:

- open parentheses: 1) $5(3m+n)$; 2) $(2x+5n-1)\cdot 7$; 3) $-(-3u+4n)$;
  4) $-2(5a+2y-1)$; 5) $(3s-4t)\cdot(-2)$; 6) $t(7u-3v)$;
  7) $-a(2b-4b+5)$; 8) $0{,}4(-5m-10n+7)$;
- combine like terms: 1) $6b-10b$; 2) $5k-k$; 3) $-5m+12m-3m$;
  4) $9y-2+4y+3-12y+2$; 5) $4n+7-9n-2+5n-8$;
  6) $7-3b+7a-3+3b-4a-2$;
  7) $4m+3n-9-5m-3n+3+m+6$.

The test for 8th grade included different types of problems. The problems of variant A were the following:

- combine like terms: 1) $9ab^2-7a^2b+2ab-5ab^2+3a^2b-2ab$;
  2) $12x^2yz+5xy^2z-8x^2yz-3xyz^3-5xy^2z-2xyz^3$;
  3) $9m-(-3n)+(-2m)-5n-m+8n$; 4) $2xyx-3x^2y+xxy$;
- perform operations: 1) $(2x+5y)+(4x-2y)$;
  2) $(3m-2n+7)-(5m-2n+9)$;
  3) $(7u-9v+3)-(2u+3v-5)+(5u+12v-3)$; 4) $3x(2x-3y)$;
  5) $-2m^2n(3mn^2-4m^2n+mn)$; 6) $(2uv^3+3u^2v-uv)\cdot u^3v^4$;
  7) $(18u^7v^2-9u^5v^4+12u^4v^3):(3u^3v^2)$;
  8) $(24m^8n^6p^7-18m^7n^5p^8):(-6m^6np^5)$; 9) $(x-y)(2x+3)$;
  10) $(-2x+y)(-3x-2y)$; 11) $(9x-3)(2x+1)$;
  12) $(m+2)(m^2-3m+1)$; 13) $(-2x+y)(x^2-xy+2)$;

14) $(2a-b+c)(a+2b-c)$; 15) $(2x-3)(2x+3)$;

16) $(m+4)(m-4)$; 17) $(x+5)^2$; 18) $(2y-3)^2$;

19) $(x+3)(x^2+3x+9)$; 20) $(x+2y)^3$;

- simplify: 1) $(2x+y)^2+(x-2y)(x+2y)-4x(x+y)$;

  2) $(2m-3)(8m+1)-(4m-3)^2-12$;
- simplify and then evaluate for specific values of variables:

  1) $2x(3xy^2-x^2y)-(12x^3y^4-9x^4y^3):(3xy^2)$, if $x=-2$ and $y=3$;

  2) $(2m-3)^2+(m+2n)(m-2n)-(5m-1)(m-2)$, if $m=-5$ and $n=2$.

The problems of variant B were the following:

- combine like terms: 1) $-3mn^2-5m^2n+7mn^2-6mn+5m^2n+4mn$;

  2) $2ab^3c^2-5a^2bc^3+3ab^3c^2+3a^2bc^3-5ab^3c^2-4a^2b^2c^2$;

  3) $5x+(-2y)-(-3x)-6x+4y-8y$; 4) $3u^2v^3-2uuv^2v+u^2vvv$;
- perform operations: 1) $(4m-3n)-(2m+n)$;

  2) $(2u+5v-7)+(-2u-5v+9)$;

  3) $(2x^2-3x+1)+(3x^2+5x-4)-(7x^2+2x-3)$; 4) $-5m(2m-1)$;

  5) $(3u^2v+2uv^2-uv)\cdot 2u^2v^3$; 6) $a^2b(-2ab^2+3a^3b-ab)$;

  7) $(20x^3y^2+12x^2y^3-4xy^2):(-4xy^2)$;

  8) $(28s^5t^3u^5-21s^2t^2u^3):(7s^2tu^3)$; 9) $(3m+n)(-m+2)$;

  10) $(-3a-2b)(2a+5b)$; 11) $(5y+3)(3y-2)$;

  12) $(u-3)(2u^2+3u-1)$; 13) $(2m-n)(m^2-2mn+1)$;

  14) $(k+2m-n)(k-2m+n)$; 15) $(4+3y)(4-3y)$;

  16) $(u-3)(3+u)$; 17) $(n-5)^2$; 18) $(2a+3b)^2$;

  19) $(x-2)(x^2+2x+4)$; 20) $(2m-n)^3$;
- simplify: 1) $2u(u-2v)-(2u-v)^2+(3u+v)(3u-v)-6u^2$;

  2) $(2n+3)^2-(2n+1)(2n-5)-20n$;
- simplify and then evaluate for specific values of variables:

  1) $3xy-(3x+2y)^2+(2x-y)(2x+y)+(x+2y)(5x-y)+3y$, if $x=-5$ and $y=-2$;

  2) $(18u^4v^3-24u^3v^4):(6uv^2)-2u(2u^2v-3uv^2)$, if $u=-1$ and $v=2$.

APPENDIX B

Problem file for trial with 11th grade students in T-algebra

1. Multiply powers
   $a^{5} \cdot a$
2. Multiply powers
   $x^{5} \cdot x^{-3} \cdot x^{4}$
3. Divide powers
   $y^{12} : y^{7}$
4. Divide powers
   $a^{-2} : a^{-7} : a^{3}$
5. Divide powers
   $b^{3} \cdot b^{5} : b^{7}$
6. Divide powers
   $s^{9} : s \cdot s^{-5}$
7. Raise to a power
   $(x^{3})^{2}$
8. Raise to a power
   $(u \cdot v)^{4}$
9. Raise to a power
   $(3abc)^{4}$
10. Raise a quotient to a power
    $\left(\frac{x}{y}\right)^{4}$
11. Raise a quotient to a power
    $\left(\frac{2}{m}\right)^{3}$
12. Raise a quotient to a power
    $\left(\frac{3a}{2c}\right)^{3}$
13. Raise a quotient to a power
    $\left(\frac{6xy}{z}\right)^{2}$
14. Raise to a power
    $(-t)^{7}$
15. Raise to a power
    $(-v)^{4}$
16. Raise to a power
    $(ab)^{0}$
17. Raise to a power
    $(-s)^{0}$
18. Simplify
    $m^{9} \cdot m^{2} : m^{11}$
19. Raise to a power
    $(-3)^{4}$
20. Raise to a power
    $(-10stuv)^{3}$
21. Raise to a power
    $(-5mnp)^{0}$
22. Multiply monomials and simplify if possible
    $-0,3xxxxyyy \cdot (-5yyyy) \cdot (-4xxx)$
23. Multiply monomials and simplify if possible
    $-0,2mnnn \cdot 5mmm \cdot (-nnnn)$
24. Multiply monomials and simplify if possible
    $0,25ababa \cdot bbbb \cdot (-4aaab)$
25. Multiply monomials and simplify if possible
    $5mn^{5} \cdot 3m^{2}n^{2}$
26. Multiply monomials and simplify if possible
    $-5a^{2}bc^{4} \cdot \left(-2a^{3}b^{5}c^{2}\right)$
27. Multiply monomials and simplify if possible
    $0,2x^{5}y^{2}z \cdot 10xz^{3} \cdot 0,5y^{4}z^{2}$

28. Multiply and divide monomials and simplify if possible
$27u^8v^5 : (9u^3v^5)$
29. Multiply and divide monomials and simplify if possible
$42x^9y^3z^4 : (-7x^6yz^4)$
30. Raise to a power and simplify if possible
$(ab^4c^3)^5$
31. Raise to a power and simplify if possible
$(-s^2t^3)^7$
32. Raise to a power and simplify if possible
$(-3x^2y^5)^4$
33. Raise to a power and simplify
$(v^5)^2 \cdot (v^3)^4$
34. Raise to a power and simplify
$(x^7)^3 : (x^4)^5$
35. Multiply and divide monomials and simplify if possible
$a^9b^{12} : (a^2b^6 \cdot a^5b)$
36. Multiply and divide monomials and simplify if possible
$s^5t^3 \cdot (-12s^2t^4) : (-4s^5t^7)$
37. Raise monomials to a power and simplify if possible
$(2a^5b^4)^4 : (a^2b^3)^2 : (a^3b)^4$
38. Raise monomials to a power and simplify if possible
$(-2x^2y^3)^4 \cdot (-3x^4y)^3 : 18x^{15}y^{13}$
39. Raise monomials to a power and simplify if possible
$(9m^{-3}n^4)^{-2} \cdot (3m^2n^5)^3$
40. Simplify
$(5t^{-4}v^2)^{-2} : (50t^6v^{-1})^{-1}$
41. Calculate the value of expression if values of variables are a=–2 b=-1
$-5a^4b^7$
42. Simplify and calculate the value of expression if values of variables are x=3 y=27
$x^{-2}y$
43. Simplify and calculate the value of expression if values of variables are x=-6 y=-2
$72x^{-1}y^5$
44. Raise a quotient to a power
$\left(\frac{1}{3}\right)^{-3}$
45. Calculate
$6^{-2} + 6^{-1}$
46. Calculate
$8^{-1} - 4^{-2}$

APPENDIX C

Categorization of errors in T-algebra

1. Unclassified errors
2. Impossible rule selected
3. Rule does not correspond to the solution algorithm
4. Selected syntactically incorrect object
5. Selected objects are of unsuitable form for applying the rule
6. Selected objects are not compatible
7. Selected objects belong to different subexpressions or to subexpressions of wrong form
8. Too few objects selected
9. Too many objects selected
10. Input is incomplete (some boxes are empty)
11. Syntactically incorrect expression entered
12. The form of the result is incorrect (does not correspond to rule and objects)
13. Calculation errors
14. Sign errors
15. Entered terms are not equivalent to selected objects
16. The whole expression is not equivalent to the previous
17. Did not recognise the answer
18. Unfinished solution offered as an answer
19. Error in final answer
20. Messages concerning the program's special requirements

In this list, errors 2-3 are usually diagnosed on rule / object selection, 4-9 and 20 are diagnosed on object selection, 10-16 and 20 are diagnosed on input of result (or intermediate result), 17-19 are diagnosed on reporting the answer to a problem.

APPENDIX D

Backus-Naur Form full description of expressions

Backus-Naur Form full description of expressions in T-algebra is as follows:

<digit> ::=	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<non-zero digit> ::=	1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<integer> ::=	<non-zero digit> | <digit> | <non-zero digit> <integer>
<power> ::=	^ <integer> | ^ + <integer> | ^ - <integer>
<decimal separator> ::=	, | .
<zero> ::=	0 | 0 <zero>
<decimal> ::=	<integer> <decimal separator> <integer> | <integer> <decimal separator> <zero> <integer>
<number> ::=	<integer> | <integer> <power> | <decimal> | <decimal> <power> | <numerical fraction> | <mixed number>
<numerical atom> ::=	<numerical parentheses> | <numerical parentheses> <numerical atom> | <numerical parentheses> <number> | <numerical parentheses> <number> <numerical atom>
<numerical term> ::=	<number> | <numerical atom> | <number> <numerical atom>
<numerical mul div> ::=	<numerical term> | <numerical term> * <numerical mul div> | <numerical term> : <numerical mul div>
<numerical sign mul div> :: =	+ <numerical mul div> | - <numerical mul div>
<numerical non-sign sum sub> ::=	<numerical mul div> | <numerical mul div> + <numerical non-sign sum sub> | <numerical mul div> - <numerical non-sign sum sub>
<numerical sum sub> ::=	<numerical non-sign sum sub> | <numerical sign mul div> | <numerical sign mul div> + <numerical non-sign sum sub> | <numerical sign mul div> - <numerical non-sign sum sub>
<numerical parentheses> ::=	[<numerical sum sub>] | (<numerical sum sub>) | [<numerical sum sub>] <power> | (<numerical sum sub>) <power>
<numerical fraction> ::=	<numerical sum sub> / <numerical sum sub>
<mixed number> ::=	<integer> <numerical fraction>
<letter> ::=	a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
<variable> ::=	<letter> | <letter> <power>
<atom> ::=	<variable> | <parentheses> | <variable> <atom> | <parentheses> <atom> | <parentheses> <number> | <parentheses> <number> <atom>
<term> ::=	<number> | <atom> | <number> <atom>
<mul div> ::=	<term> | <term> * <mul div> | <term> : <mul div>
<sign mul div> :: =	+ <mul div> | - <mul div>
<non-sign sum sub> ::=	<mul div> | <mul div> + <non-sign sum sub> | <mul div> - <non-sign sum sub>
<sum sub> ::=	<non-sign sum sub> | <sign mul div> | <sign mul div> + <non-sign sum sub> | <sign mul div> - <non-sign sum sub>

<parentheses> ::=	[<sum sub>] | (<sum sub>) | [<sum sub>] <power> | (<sum sub>) <power>
<fraction> ::=	<sum sub> / <sum sub>
<equation inequality signs> ::=	= | < | > | <= | >=
<equation inequality> ::=	<sum sub> <equation inequality signs> <sum sub>
<system> ::=	<equation inequality> & <equation inequality> | <equation inequality> & <system>
<expression> ::=	<system> | <equation inequality> | <sum sub>

CURRICULUM VITAE

Dmitri Lepp

Citizenship:	Republic of Estonia
Born:	May 2, 1981, Võru, Estonia
Marital status:	married
Address:	Loopealse 20, Ülenurme alevik, Tartumaa, Estonia
Contacts:	phone: +372 55 85 309
	e-mail: dmitri@ut.ee; dmitri.lepp@ee.fujitsu.com

Education

1987–1991	Tartu Secondary School No. 4
1991–1998	Tartu Annelinn Upper Secondary School
1998–2001	University of Tartu, Bachelor in Computer Science (*cum laude*)
2001–2003	University of Tartu, MSc in Computer Science
2003–…	University of Tartu, PhD studies in Computer Science

Professional employment

2001–…	AS Fujitsu Services, project manager

Scientific work

The main fields of interest are interactive learning environments and the use of computers in mathematics education.

CURRICULUM VITAE

Dmitri Lepp

Kodakondsus:	Eesti Vabariik
Sünniaeg ja -koht:	2. mai 1981, Võru, Eesti
Perekonnaseis:	abielus
Aadress:	Loopealse 20, Ülenurme alevik, Tartumaa, Eesti
Kontakt:	tel.: +372 55 85 309
	e-post: dmitri@ut.ee; dmitri.lepp@ee.fujitsu.com

Haridus

1987–1991	Tartu 4. keskkool
1991–1998	Tartu Annelinna gümnaasium
1998–2001	Tartu Ülikool, informaatika bakalaureus (*cum laude*)
2001–2003	Tartu Ülikool, informaatika magister (MSc)
2003–…	Tartu Ülikool, doktoriõpe informaatika erialal

Erialane teenistuskäik

2001–…	AS Fujitsu Services, projektijuht

Teadustegevus

Peamine uurimisvaldkond on interaktiivsed õpisüsteemid ja arvutid matemaatikahariduses.

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

1. **Mati Heinloo.** The design of nonhomogeneous spherical vessels, cylindrical tubes and circular discs. Tartu, 1991, 23 p.
2. **Boris Komrakov.** Primitive actions and the Sophus Lie problem. Tartu, 1991, 14 p.
3. **Jaak Heinloo.** Phenomenological (continuum) theory of turbulence. Tartu, 1992, 47 p.
4. **Ants Tauts.** Infinite formulae in intuitionistic logic of higher order. Tartu, 1992, 15 p.
5. **Tarmo Soomere.** Kinetic theory of Rossby waves. Tartu, 1992, 32 p.
6. **Jüri Majak.** Optimization of plastic axisymmetric plates and shells in the case of Von Mises yield condition. Tartu, 1992, 32 p.
7. **Ants Aasma.** Matrix transformations of summability and absolute summability fields of matrix methods. Tartu, 1993, 32 p.
8. **Helle Hein.** Optimization of plastic axisymmetric plates and shells with piece-wise constant thickness. Tartu, 1993, 28 p.
9. **Toomas Kiho.** Study of optimality of iterated Lavrentiev method and its generalizations. Tartu, 1994, 23 p.
10. **Arne Kokk.** Joint spectral theory and extension of non-trivial multiplicative linear functionals. Tartu, 1995, 165 p.
11. **Toomas Lepikult.** Automated calculation of dynamically loaded rigid-plastic structures. Tartu, 1995, 93 p, (in Russian).
12. **Sander Hannus.** Parametrical optimization of the plastic cylindrical shells by taking into account geometrical and physical nonlinearities. Tartu, 1995, 74 p, (in Russian).
13. **Sergei Tupailo.** Hilbert's epsilon-symbol in predicative subsystems of analysis. Tartu, 1996, 134 p.
14. **Enno Saks.** Analysis and optimization of elastic-plastic shafts in torsion. Tartu, 1996, 96 p.
15. **Valdis Laan.** Pullbacks and flatness properties of acts. Tartu, 1999, 90 p.
16. **Märt Põldvere.** Subspaces of Banach spaces having Phelps' uniqueness property. Tartu, 1999, 74 p.
17. **Jelena Ausekle.** Compactness of operators in Lorentz and Orlicz sequence spaces. Tartu, 1999, 72 p.
18. **Krista Fischer.** Structural mean models for analyzing the effect of compliance in clinical trials. Tartu, 1999, 124 p.

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
20. **Jüri Lember.** Consistency of empirical k-centres. Tartu, 1999, 148 p.
21. **Ella Puman.** Optimization of plastic conical shells. Tartu, 2000, 102 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
25. **Maria Zeltser.** Investigation of double sequence spaces by soft and hard analitical methods. Tartu, 2001, 154 p.
26. **Ernst Tungel.** Optimization of plastic spherical shells. Tartu, 2001, 90 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 p.
28. **Rainis Haller.** *M(r,s)*-inequalities. Tartu, 2002, 78 p.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
30. **Eno Tõnisson.** Solving of expession manipulation exercises in computer algebra systems. Tartu, 2002, 92 p.
31. **Mart Abel.** Structure of Gelfand-Mazur algebras. Tartu, 2003. 94 p.
32. **Vladimir Kuchmei.** Affine completeness of some ockham algebras. Tartu, 2003. 100 p.
33. **Olga Dunajeva.** Asymptotic matrix methods in statistical inference problems. Tartu 2003. 78 p.
34. **Mare Tarang.** Stability of the spline collocation method for volterra integro-differential equations. Tartu 2004. 90 p.
35. **Tatjana Nahtman.** Permutation invariance and reparameterizations in linear models. Tartu 2004. 91 p.
36. **Märt Möls.** Linear mixed models with equivalent predictors. Tartu 2004. 70 p.
37. **Kristiina Hakk.** Approximation methods for weakly singular integral equations with discontinuous coefficients. Tartu 2004, 137 p.
38. **Meelis Käärik.** Fitting sets to probability distributions. Tartu 2005, 90 p.
39. **Inga Parts.** Piecewise polynomial collocation methods for solving weakly singular integro-differential equations. Tartu 2005, 140 p.
40. **Natalia Saealle.** Convergence and summability with speed of functional series. Tartu 2005, 91 p.
41. **Tanel Kaart.** The reliability of linear mixed models in genetic studies. Tartu 2006, 124 p.
42. **Kadre Torn.** Shear and bending response of inelastic structures to dynamic load. Tartu 2006, 142 p.

43. **Kristel Mikkor.** Uniform factorisation for compact subsets of Banach spaces of operators. Tartu 2006, 72 p.
44. **Darja Saveljeva.** Quadratic and cubic spline collocation for Volterra integral equations. Tartu 2006, 117 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
46. **Annely Mürk.** Optimization of inelastic plates with cracks. Tartu 2006. 137 p.
47. **Annemai Raidjõe.** Sequence spaces defined by modulus functions and superposition operators. Tartu 2006, 97 p.
48. **Olga Panova.** Real Gelfand-Mazur algebras. Tartu 2006, 82 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
50. **Margus Pihlak.** Approximation of multivariate distribution functions. Tartu 2007, 82 p.
51. **Ene Käärik.** Handling dropouts in repeated measurements using copulas. Tartu 2007, 99 p.
52. **Artur Sepp.** Affine models in mathematical finance: an analytical approach. Tartu 2007, 147 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
54. **Kaja Sõstra.** Restriction estimator for domains. Tartu 2007, 104 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
57. **Evely Leetma.** Solution of smoothing problems with obstacles. Tartu 2009, 81 p.
58. **Ants Kaasik.** Estimating ruin probabilities in the Cramér-Lundberg model with heavy-tailed claims. Tartu 2009, 139 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
60. **Indrek Zolk.** The commuting bounded approximation property of Banach spaces. Tartu 2010, 107 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
63. **Marek Kolk.** Piecewise Polynomial Collocation for Volterra Integral Equations with Singularities. Tartu 2010, 134 p.

64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
65. **Larissa Roots.** Free vibrations of stepped cylindrical shells containing cracks. Tartu 2010, 94 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
68. **Olga Liivapuu.** Graded q-differential algebras and algebraic models in noncommutative geometry. Tartu 2011, 112 p.
69. **Aleksei Lissitsin.** Convex approximation properties of Banach spaces. Tartu 2011, 107 p.
70. **Lauri Tart.** Morita equivalence of partially ordered semigroups. Tartu 2011, 101 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.
72. **Margus Treumuth.** A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects. Tartu 2011, 95 p.