What toolbox is necessary for building exercise environments for algebraic transformations

Rein Prank
e-mail: rein.prank@ut.ee
University of Tartu
Tartu, Estonia

Abstract

The paper analyses functionalities, which are necessary for implementation of different phases of solution steps in exercise environments for school algebra and calculus. The author tries to identify the components that could be borrowed from existing or future universal toolboxes (computer algebra systems or libraries of utilities).

1. Introduction

Algebraic transformations are an important part of mathematics education. Many training tasks for school and university students use algebraic transformations as an instrument for getting the answer. However, students also solve hundreds of purely technical exercises to develop their expression manipulation skills. For many years, the dominating educational environments for algebraic transformations have been computer algebra systems (CAS).

Computer algebra systems contain, for typical exercise topics, a comparatively small set of powerful commands (Expand, Factorize, Simplify, Solve, Differentiate, Integrate, ...), which enables users to solve typical training tasks in a small number of steps, often even in only one step. The role of the student is to choose the appropriate command (sequence of commands) and to specify the operands (some systems choose the operands automatically). The system executes the command(s). Such a division of roles is suitable for word problems, for example, where the emphasis in the class is on composing the equations, while algebraic transformations are used merely as a technical tool for counting the result. The problem types where the object of the work is the transformation itself are much less suitable for the use of CAS.

The first type of tasks, which is unsuitable for CAS, includes purely technical exercises where the students are required to apply conversion rules and perform the necessary numerical calculations. In such exercises, the students should find and input the results of the steps. The second, and probably the most exploited type of exercise, focuses on the task of building an appropriate sequence of conversion steps. For a number of such task types (finding the value of a numerical expression, the solution of linear equations/systems or the derivative of an expression), the students should basically know and apply a 'textbook algorithm.' However, there are also more creative problem types (the solution of more complex equations/systems, factorization of polynomials, integration etc.) where the students themselves are expected to devise the solution path. Although the general working scheme of CASs seems fitting for such tasks, there are many cases where CAS does not enable the construction of the solution in school-style. This is due to the lack of fine-grained commands for developing the solution steps in sufficient detail, as would be expected in school mathematics. In addition, CASs tend to execute the commands iteratively and include several uses of the same conversion rule in one step.

The existence of important and widely used task settings, which do not have didactically acceptable CAS dialogs, means that we also need special educational algebraic transformation environments. The solutions of algebraic transformations are often quite voluminous and contain many details. Teachers are not able to provide feedback about mistakes in time, and the quality of learning could be improved by the use of computer environments. Nevertheless, only a very limited choice of learning environments is available at the moment. On the other hand, the existing

environments contain approaches that allow computerize both purely technical and more creative tasks.

In the abovementioned two cases, the environments for technical exercises could use an input-based step dialog and the environments for building entire solutions could retain the rule-based user interface of CASs (but also include a didactically motivated and task-dependent set of conversion rules).

An input-based dialog is implemented in Aplusix [6]. This program was created for elementary algebra: calculation of values of numerical expressions; simplification of algebraic expressions; solution of equations, inequalities and linear equation systems. For each step, the student copies the previous expression (equation, …) to the next line and then uses the expression editor for transforming it into the result of the step (Figure 1).

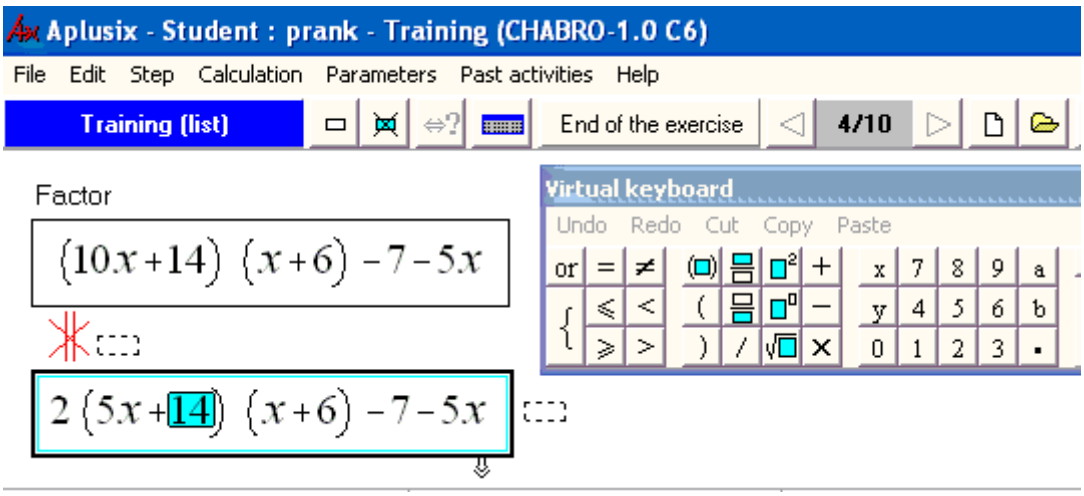

Figure 1. *Solution step in input-based environment Aplusix. The student has copied the expression to the next line and changes now 10x+14 to 2(5x+7). The crossed-out sign of equality indicates that the expressions are not yet equivalent.*

There is also a major exercise environment that uses rule-based dialog – MathXpert Calculus Assistant [2, 3]. In MathXpert, the conversion step consists of two substeps. At the first substep, the student marks a subexpression of the current expression (equation, inequality, system of equations). For the second substep (Figure 2), the program displays a menu with commands and rules that are applicable to the marked subexpression (or to a part of it). The student selects an item from the menu and the program applies the corresponding command/rule.

The author of this paper was the head of the T-algebra project [7, 12] for basic school algebra where the step dialog combines rule-based and input-based approaches. A combination of the two dialog types was also used in the Mathematics software of Education Program for Gifted Youth at Stanford University [13]. It may be surprising, but there do not seem to be any other relatively large environments that would cover a significant range of topics. Several systems enable solving specific problem types within a smaller range, for example [1, 9]. The main reason for the lack of software is the need to implement a large amount of complex material in such an extensive system. For example, Michael Beeson, at the SCE 2006 Symposium in Beijing (http://www.cc4cm.org/sce2006), said that implementation of MathXpert took seven years. The main part of T-algebra was written by three programmers over the period of four years. The future developers of solution environments should consider using existing utilities for mathematical, syntactical and editing functions.

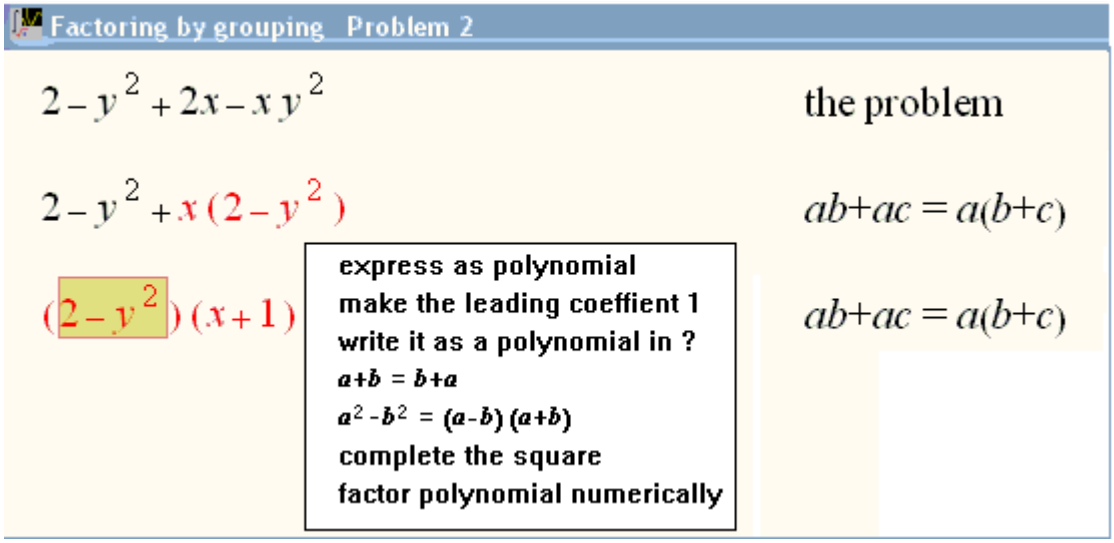

Figure 2. *Second substep in rule-based environment MathXpert. The student has selected subexpression 2-y^2. The program displays a menu with applicable rules.*

This paper analyses the functions that are necessary for building expression manipulation environments for algebra and calculus, and tries to identify the components that could be borrowed from universal toolboxes (underlying CAS or libraries of utilities).

Members of future project groups could find in this paper some information on the functionalities that are necessary for implementating different elements of a solution step dialog. This could be helpful for deciding what kind of dialog can be implemented for the planned topic. The paper also contains suggestions for the creators of computer algebra systems and libraries of algebraic utilities regarding the content that is important for the authors of learning environments.

In section 2, we describe a scheme for a solution step dialog that contains elements of both input-based and rule-based expression manipulation. From this scheme, we extract ten essential kernel functionalities. Section 3 discusses all ten functionalities and tries to identify those that could be implemented in a sufficiently general form and stored in toolboxes for use by several programs. Section 4 summarises the findings.

2. A scheme of detailed dialog and the corresponding kernel functions

Our description of the technical needs of environments is based on the following model of student work. At each solution step, the student should (at least in his/her mind):

1) choose an operation to be applied to certain part(s) of previous expression,
2) choose operand(s) in previous expression,
3) find the result of applying the operation and of assembling the new expression.

In a particular environment or in a particular working mode, some of these activities can be carried out explicitly and others only mentally. Some activities can be left to the computer and the student attends to them passively. For example, in Aplusix the student needs to make all the above listed decisions, but he/she enters only the final result in stage 3. In MathXpert (as well as in CASs), stage 3 is performed by the computer. In some environments (including most settings of CASs), the operands of the selected operation are chosen by the program.

Many mistakes made by pupils are caused by a misunderstanding of the essence of a particular operation or of the structure of an expression (the order of operations, the meaning of parts of fractions etc). Students can also confuse the activities that correspond to different patterns of operands (for example, reduce one member by a factor when the numerator is not a product but a sum of members) or apply operations to incompatible operands (combine terms that are not similar). It would be much easier for a computer environment to diagnose such mistakes if the

dialog contained explicit information on the student's intentions (what operation is applied to what operands). Information about these intentions also enables feedback immediately after the decisions of the first two stages have been entered, without entering the result of meaningless operations. Therefore, we assume that, in the more detailed working modes, the dialog could contain the following actions:

1) selection of the operation (conversion rule) from the menu,
2) precise marking of operand(s) in expression,
3) entering the result of operation.

'Precise marking' in this context means an interface where the student marks only the actual operands (for example, only those like terms in a sum that the student intends to collect).

The exercise environment should enable presentation of expressions, equations and systems on the screen, support input at all substeps of solution steps, check correctness of the input, diagnose errors, give hints, and also demonstrate execution of steps and entire solutions. This requires several kernel functionalities. The following is a list of the most important functionalities, which are discussed in the next section. The first two items on the list have quite a universal character; others are required for particular activities in the dialog.

1. Sufficient spectrum of syntactical constructions for numbers, operations, functions, expressions, equations, systems, etc.
2. 2D layout of expressions and corresponding WYSIWYG expression editor.
3. Editor functionality for marking arbitrary parts of expressions.
4. Checking of syntactical correctness of expressions and marking.
5. Checking of suitability of marked operands for selected operation.
6. Testing of equivalence of expressions/equations/…
7. Checking of consistency of the entered result with the operation and operands.
8. Execution of all menu operations.
9. Execution of 'textbook algorithms' for all task types.
10. Checking the final 'solved' form in all task types.

Note that no item in this list corresponds to the first substep in the above description of the step – selection of operation from the menu. This action is computationally simple and can be implemented using standard tools of programming environments.

3. Discussion of kernel functionalities

3.1 Syntactical constructions

School mathematics and textbooks contain a wide spectrum of mathematical objects (different kinds of numbers, unary and binary operations, functions, fractions, equations, systems, etc.) and syntactical constructions for presenting them in expressions. In this context, 'different syntactical constructions' mean, for example, the opportunity to write $\sin x$ instead of $sin(x)$ or $sin^2(x)$ instead of $(sin(x))^2$. One of the objections of teachers, who do not use computers, has always been that the software does not implement certain important objects or constructions or the objects and constructions look differently on the screen than they do in the textbook. The sets of objects and constructions in computer algebra systems are clearly smaller than in school algebra. For the purposes of teaching/learning each topic, it would be easier to use software where all constructions are available and have the same form as in the textbook. Is it possible to solve this contradiction?

The issue of existence of necessary primitive objects and classes of expressions is quite clear: they should be implemented in the software. If the textbook chapter contains mixed numbers then mixed numbers should be implemented in the software as well, and it would be desirable to obtain them from an underlying toolbox. If the textbook writes about unary plus then the software should not eliminate it automatically from expressions. More generally, it should be possible to

write and manipulate expressions in any correct form, without forced conversion into a specific standard form of a given system.

In case of some topics, the learning environments should implement more constructions than required in CAS. Learning environments for learning of elementary skills should enable presentation of results of rather basic transformation steps. Sometimes, this requires addition of syntactic categories that are not needed in computer algebra systems. A CAS can implement mixed numbers and use a syntax definition where the numerator should be a nonnegative integer and the denominator should be a positive integer. This is sufficient for input of reasonable tasks and output of results of CAS commands. However, a learning environment should also accept mixed numbers in intermediate results of operations, where at least the numerator contains arithmetical operations. This means that mixed numbers with negative fractional part appear and should be understood by the system.

When we consider the forms of presentation of expressions, then CAS or a library of utilities cannot implement everything that exists in textbooks. Some constructions are redundant because they duplicate other, more general constructions. Some constructions can cause contradictions. Indeed, there can be contradictions between different textbooks and sometimes even within the same textbook (or in a series of textbooks written for consequent grades).

A very common construction that is often prohibited in CASs is writing the products without the multiplication sign. Some systems allow entering *2ab* but convert it automatically to $2 \cdot a \cdot b$. Furthermore, the curriculum of mathematics also includes repeating decimals and, in some countries, they are written in a form where the repeating part is closed in brackets. If the system uses the same time simplified presentation of the product, it could result in the following steps:

$$\frac{7}{30} = \frac{6}{30} + \frac{1}{30} = 0.2 + 0.0(3) = 0.2(3) = 0.6$$

However, in most settings it is desirable for learning environments to allow the simplified form. If the underlying CAS or toolbox requires expressions with multiplication signs then they can be easily inserted before call and removed after that from the returned result.

When we think about $sin^2(x)$ and about the possible general form of such expressions, we should keep in mind that in some settings $f^2(x)$ means squared application of *f*, i.e. $f(f(x))$.

The next suspicious construction is using the space as special separator in expressions. Usually, the syntax and semantics of such separator is not explained in the textbooks. In the case of trigonometric functions, many textbooks use space before an argument in expressions like *sin 2x* or *tan 30°*. They do not use such notation for arbitrary arguments, but usually do not explain which cases are allowed and which cases are not. In our T-algebra project, we encountered another use of space – the example of division of monomials in the textbook looked as follows:

$$28x^3y^2 : 7xy = 4x^2y$$

(the division sign ':' was between two spaces). This was written despite earlier instruction that operations of the same level of priority (multiplication and division) should be performed from left to right. The author is of opinion that it would be better not to use the space as separator.

The problems of adequate presentation of expressions can be solved only through compromise. Developers of kernel functionalities should study educational needs and add the necessary cases. Developers of learning environments should use those sources of utilities that are suitable for actual topics. The teachers should speak clearly about their needs but should also understand that school mathematics contains elements that cannot be presented in software, because they are introduced in textbooks for some primitive cases which may not have acceptable general definitions.

3.2 WYSIWYG expression editor

Our students know that computer software should be user-friendly and state-of-the-art. This is doubly important if the software is used in teaching of a subject that is not particularly loved by

some students. In case of expression manipulation, a user-friendly and state-of-the-art environment should enable 2D presentation of expressions on the screen and include a WYSIWYG expression editor. This would also reduce the cognitive load on students because, for lower grades, conversion of expressions from linear to 2D form and back can be as difficult as the exercises themselves.

Most computer algebra systems and virtually all new learning environments today have a 2D editor. There are also some early attempts at standardizing the virtual keyboards used for mathematical symbols and syntactical constructions. However, when we try to mark a subexpression with the mouse in the editor of MathXpert or T-algebra, we realize quickly that the editor is not a professional product. The colored rectangle follows the movements of our hands and although we manage to mark the desired subexpression we do not get a feeling of assured control.

Enthusiasts of mathematical didactics and computer algebra usually implement learning environments for working with expressions. Their work would be more fruitful if there were some good expression editors available for inclusion in the learning environments. In the next subsection, we discuss an additional feature of editors that would be required for building learning environments but is not commonly found in existing editors.

3.3 Marking of arbitrary parts of an expression

We discuss here certain marking options that should be central to building didactically adequate environments and, therefore, should be included in underlying software.

Expression editors and expression manipulation environments usually allow marking a certain part of the expression for the purpose of copying, deleting or applying a mathematical operation. In both rule-based and input-based environments, marking of the active part allows one to copy the passive part(s) of the expression to the next line and to insert the result of the operation (or insert input box(es) for the result). Separation of the active and passive parts of the expression simplifies diagnostics of mistakes and formulation of feedback.

In many cases, different operands of operation are located in separate locations in the expression. A typical example of such a situation is the collection of like terms. The most natural manner of marking would be to mark only those members of the expression that the student intends to collect. Students often make errors (occasional slips but also misunderstandings) in recognition of similarity and, as a result, add the coefficients of members that are not similar. The use of precise marking would enable the generation of an error message and the formulation of an adequate diagnosis even before entering the result.

Precise marking of separate operands requires the editor to offer the possibility of marking more than one piece of the expression. The need to mark separate operands also occurs in case of reducing fractions (with products in numerator and denominator), moving the members to other side of equation, applying auxiliary rules of multiplication of polynomials, separating groups of members for factorization, and other operations. Marking of actual operands also enables to improve the readability of completed solution steps. Figure 3 presents some solution steps in the T-algebra environment where the editor enables precise marking.

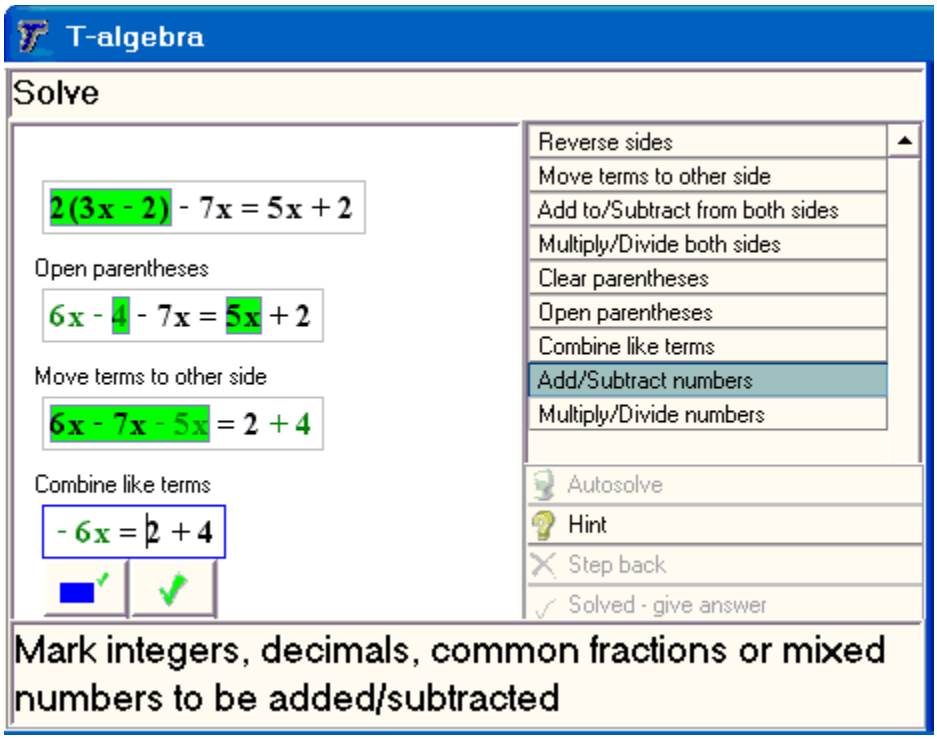

Figure 3. *Solution window of T-algebra. Operands of previous steps are marked by green background. The student has selected the operation* Add/Subtract numbers *for the next step but not yet marked the operands.*

Many editors try to help the user. They use information about the structure of expressions and the order of operations to ensure that only the proper subexpressions are marked. For example, it is impossible to mark *x+1* or *3x* as subexpressions in *43x+1*. This feature is very useful for engineers and scientists, who use CAS for the work with expressions that sometimes do not fit on the screen. However, at least the environments for learning elementary skills and the software for assessment should enable the student to select the operands that correspond to his/her correct or incorrect understanding of the subject matter. In case of mistakes, the environment should give corrective feedback. Such environments need editors that allow marking both correct and incorrect parts of expressions.

Note here that the technical possibility of invalid marking is not the same as 'paper and pencil mode' where the student can continue even when the result of the previous step is meaningless or changes the task completely. According to our understanding, the program should require correction of invalid marking before work is continued.

3.4 Checking of syntactical correctness of expressions/marking

After the student has entered the result of a step, the program should check its syntax. The same should be done in the teacher's program after the teacher has entered initial expressions for exercises. After marking the operands, the program should check whether the marked parts are correct expressions and whether they are proper subexpressions of the whole expression (i.e., the student has correctly understood the order of operations, the structure of numbers and fractions, etc.).

The precise meaning of syntactical correctness depends on the particular set of allowed primitive objects and syntactical constructions. A checker of a fixed syntax can only have limited use in a specific environment. Nevertheless, the habitual objects and constructions belong to a small and finite number of types. The toolbox software can contain syntax engine(s) where the concrete

syntax can be defined by lists of members of each type. For example, in a concrete syntax the decimal delimiter can be a comma and allowed trigonometric functions can be *sin*, *cos* and *tan*.

Syntax definitions of a particular environment can contain quantitative and qualitative restrictions. For example, an environment for younger grades can enable only one level of exponents, prohibit nested fractions and prohibit the use of variables in the numerator and denominator of fractions. It would probably be better to leave the checking of such restrictions to programmers of concrete environments. A universal checker could return either an error message and error location (if possible) when the structure of expression is incorrect or a positive answer together with the result of parsing if the structure is correct.

It is also important to have a choice of languages so that the messages are reported in the native language of learners.

3.5 Checking the suitability of marked operands for a selected operation

Before execution of a selected operation of marked operands (in rule-based working mode) or display of input box(es) for the result (in input-based mode), the environment should check the suitability of operands for the selected operation. For different operations, the notion of suitability can include requirements for individual operands themselves, for their location in the expression, for consistency of the set of operands, for the number of operands, etc. For example, in elementary exercises on combining like terms, the operands should be monomials. They should belong to the same sum (polynomial) in the expression and have equal variable parts. There should be at least two operands.

In many cases, however, the precise conditions of acceptability of operands for different operations depend on traditions, textbooks, and individual teachers. In our normal mathematical practice, we do not apply formal rewrite rules. We name conversion operations after their 'main' content but our written conversion steps often contain some extent of pre- and postprocessing. The kind of preprocessing allowed before a given operation broadens the allowed forms of operands.

Let us consider the expression

$$3aba+4a^2b-ba^2+2a\cdot 5ab-(3a^2b-ba^2).$$

Most teachers would probably accept combining the first three members without requiring any preceding conversion of variable parts to an identical form. However, some teachers require normalization of monomials before operations. Then again, other teachers would accept the inclusion of all members, even though the fourth member is formally not a monomial and the last two members form a separate sum. Such variations lead the authors of different environments to different decisions about the limitations of allowed preprocessing. Our opinion is that checking the suitability of operands is not a function that could be included in universal toolboxes in a finalized form. The toolboxes should contain a sufficient choice of utilities for checking syntactical properties and for conversion of expressions to different forms. The authors of the environments can then use them to assemble different checking procedures.

3.6 Testing of equivalence

In rule-based environments, the program executes the step after successful testing of the suitability of operands for the selected operation. In input-based and in combined environments, the result of conversion should be entered by the student. The first action after that is usually testing the syntax of the result. We discussed the corresponding kernel features in subsection 3.4. If the result is a syntactically correct expression, the next obligatory stage in processing the input is the testing of equivalence with the previous line.

Consider first the tasks where the initial object and the final answer are expressions (also including numbers). Most expression conversion operations preserve equivalence with the previous line at each step . A break in equivalence means that the student has made an error. The program should detect the error and issue an error message. Some operations (finding of reciprocal value,

rounding) do not maintain equivalence with the previous line. Nevertheless, in such cases the program should check the equivalence of the entered result with a 'right' value or expression. For a better indication of the location of errors, it is sometimes necessary to additionally test the equivalence between smaller components of the entered result and the corresponding parts of the previous expression. This means that, to enable implementation of the input-based working mode, the environment should be capable of testing the equivalence of any expressions that can occur in a particular task type.

Complexity of the equivalence testing depends on the class of expressions. Some classes enable defining a canonical form of expressions. Every expression can be converted to the canonical form, and two expressions are equivalent if, and only if, the canonical forms are identical (as strings of symbols). For example, the canonical form of polynomials is an expanded form where the members are in a fixed order (for example, alphabetic order). If, instead of canonical forms of two polynomials, we compare the canonical form of the difference of polynomials with zero, then the need for ordering disappears. This means that if the learning environment contains solution algorithms for addition and multiplication of polynomials then the solution algorithms can be used for verification of equivalence in the area of polynomials. For example, T-algebra verifies equivalence in all kinds of expressions (including polynomials, division and fractions) mainly by using solution algorithms of appropriate problem types.

The problem is much more complex in the case of arbitrary expressions. Theoretical results indicate that it is not possible to construct a general algorithm for the testing of equivalence of arbitrary expressions, even for school mathematics. The most important negative result was proved by D. Richardson in 1968 [14]. In 1970, Y. Matiyasevich removed in his solution of Hilbert's tenth problem the exponential function from the earlier result about Diofantine equations, which resulted in the contemporary formulation of the Richardson's theorem [10, 11]:

Let F denote the class of functions in one real variable that can be defined by expressions constructed from the variable x, the integers and the number π, combined through addition, subtraction, multiplication, sin and abs (absolute value).

There is no algorithm for deciding for an arbitrary given expression E(x) from the class F whether the equality

$$E(x) = 0$$

holds identically for all values of x.

Note that $|x| = sqrt(x^2)$ and when we replace the absolute value in the theorem by square root, we get a set of primitives where each primitive is indispensable for expressions in school trigonometry. Consequently, the input-based environment for trigonometry cannot have a testing module that is always successful in checking the equivalence.

Positive results [15, 8] state that the problem of equivalence is decidable when exponentiation and division are added to polynomial operations. The main argument in the proofs relies on the estimation of the upper bound of the number of roots in the difference of the two expressions. This allows drawing a positive conclusion about equivalence if the difference is zero in a sufficient number of points.

But even for the expressions from undecidable classes, it is quite natural to compute the values of two functions in some sample of points and to compare them in order to test for equivalence. If the values are different in some point (taking into consideration the precision of calculations), then the program can decide that the expressions are not equivalent. If the values are (approximately) equal in all points, then the program should accept that the expressions are equivalent (with high probability). This approach is described in [5] and used, for example, in the *testeq* procedure in Maple.

In most practical cases, student conversion mistakes cause large differences in values, and nonequivalence is often discovered already, at the first point of evaluation. However, there is also a case where expressions differ only within a zero-measure set. For example, we know that the

expressions x^2/x and x are not equivalent, but in order to discover this nonequivalence, the random evaluator should choose *0* for the value of *x*. In [4] T.Fisher also warns about expressions like *abs(1000-x*) and *1000–x* where the sample values can be too small for discovering the difference.

Testing the equivalence of expressions is a complex task that requires very specific knowledge. Therefore, it would be desirable to borrow this functionality from a CAS or a library of utilities that contains sufficiently general and computationally efficient solutions.

The tasks on solution of equations, inequalities and systems of equations contain conversions of objects that are composed of several expressions. Most of the conversion operations should again preserve the equivalence of the respective objects. Equivalence of equations, inequalities or systems of equations means equality of sets of solutions. As long as the cases remain sufficiently simple, it is possible to get precise solutions and to compare them. The problem types of T-algebra only include linear equations, inequalities and systems with rational coefficients. Their precise solutions also only contain rational numbers. Therefore, we could implement testing of equivalence of equations, inequalities and systems through solution algorithms of corresponding task types as well.

Generally, testing of equivalence of such objects is not implemented in CASs. In addition, algorithms for the solution of equations use operations that do not preserve equivalence of equations (for examples, raising to power 2, removing of some functional operator, etc.). Most textbooks deliver sample solutions but they do not describe what conversion steps should be treated as being correct. This means that even for implementation of rule-based environments, we need some additional theoretical knowledge.

3.7 Checking of consistency of the entered result with the operation and operands

If the environment asks the student what operation he/she will apply at the current step, then, after the result has been entered, the program should ascertain that this operation was really executed. Unfortunately, this check is quite complicated. We cannot assume that the result of application of the selected operation to marked operands is uniquely defined.

Execution of some operations depends on additional parameters. For example, the fraction *12/18* can be reduced by factor *2*, *3* or *6*. However, the foregoing dialog step for separate entering of the factor would be reasonable only during the very first exercises on reducing.

The domain of suitability of operands could be expanded by taking into account the possibility of preprocessing. Analogously, we can accept the result of the step where the student has performed some postprocessing after the main operation. The acceptable forms of allowed postprocessing normally include numerical calculations and any standard conversions that are supposed to be routine for the target group of the exercises in question. For example, the program can accept the input *2x=-11* when the student has applied operation Move members to other side to the members *5* and *2x* of equation *4x+5 = 2x-6*.

In some cases, it would be reasonable to accept the result of the step in a situation where the student has not performed the selected operation but only some crucial part of it. For example, the teachers can, in cases of operations with fractions, recommend that weaker students not perform the entire operation, but focus on what should be done in the numerator and denominator:

$$\frac{5}{7}:\frac{3}{4}=\frac{5\cdot 4}{7\cdot 3} \quad \text{or} \quad \frac{5}{7}+\frac{3}{4}=\frac{5\cdot 4+7\cdot 3}{7\cdot 4}$$

Admissibility of such reduction to integer operations can also be implemented in the checking procedures of the rules Divide fractions and Add fractions.

Our conclusion about testing of consistency is similar to the conclusion in the case of suitability of operands. For most operations, we do not currently have a common understanding of what forms of results should be accepted or refused. This requires further experimentation with different variants. The general toolboxes could contain detailed utilities for implementation of different variants.

3.8 Execution of all menu operations

In rule-based environments, this feature is explicitly exploited at every step. For example, in MathXpert the program proposes, after marking a subexpression, a menu of applicable operations and then executes the operation selected by the student. The working modes of T-algebra assume that steps are executed by the student, but the program is also able to automatically fill the input box by the result of the operation that was specified by the student.

Purely input-based environments do not have menus of operations, and the possible steps are not prescribed by the program. However, if the authors of such an environment would like the program to dynamically demonstrate the solutions (of user-given exercises) or the next recommended step, then the program should be able to follow the textbook algorithm and execute all operations in it. In cases of non-algorithmic tasks, the environment should be able to execute a set of operations that enables students to solve every task.

Therefore, we can conclude that a solid learning environment should be able to execute the same steps that the students are learning. Implementations of commands of computer algebra systems do not constitute a sufficient basis for menu operations. Learning environments require tens and sometimes hundreds of different low-level rules for producing small steps.

It would be nice to have a toolbox where at least the most standard operations are already implemented. In terms of the possibility of universal collections of utilities, the task of execution of operations is less sensitive to local traditions and didactical tenets than checking whether the student-entered result is consistent with the operation. Traditional operations in textbooks have a commonly accepted main content, and it is possible to implement this content together with reasonable preprocessing and with minimal amount of postprocessing. For example, the choices made in MathXpert seem to be quite acceptable. Indeed, the programmers of new environments can also use toolbox procedures that contain excessive preprocessing. Good examples of such situations are the utilities that do not require the precise marking of operands, but choose an appropriate subexpression for transformation by themselves. If the operand contains unsuitable parts, then the main program can refuse such operand even before calling the utility and the superfluous skills of the utility are not visible in the dialog.

Collections of utilities, which correspond to operations from the algorithms in mathematics textbooks, could be very useful for building training environments.

3.9 Execution of ‘textbook algorithms’ for all task types

Note first that we are not referring here to algorithms that receive the task and return the answer. Such black-box algorithms are implemented in computer algebra systems. However, learning environments need algorithms that return the entire solution. If we want students to learn how to apply a ‘textbook algorithm’ to the tasks of a specific problem type, then our learning environment should be able to build demo solutions and generate step hints according to this algorithm. Consequently, the kernel of our environment should contain solution-building algorithms for all problem types. In cases of rule-based environments, this algorithm should build the solutions consisting of applications of the rules. Instead of an algorithm that returns the whole solution, it is possible to have an algorithm that returns the result of the next step. In a rule-based environment, the steps should be annotated by rules and markings of operands.

Programming of such a collection of algorithms would be a very laborious task, even when the problem types themselves are only limited to basic algebra. For example, T-algebra implements about 60 problem types with fractions, linear equations, systems, exponents, monomials, and polynomials. In order to build environments of comparable size, it would be nice to have a repository of implemented algorithms. What should we take into account when we want to make a contribution to such repository or use the algorithms implemented by other authors?

Algorithms for some problem types can differ between different countries and textbooks. Consider, for instance, some examples with fractions. Some countries/textbooks prefer to work with decimal fractions, while others claim that this is allowed only if no rounding is used. In some countries/textbooks, mixed numbers are converted to improper fractions before addition or subtraction. Other textbooks claim that the reason for using mixed numbers is simply doing the addition and subtraction separately with whole and fractional parts. Some textbooks almost obligatorily rewrite division of common fractions as multiplication with reciprocal value and then apply multiplication. Nevertheless, this plurality is not infinite and implementation of any reasonable variant has some value.

Some topics in high school and university mathematics (factorization of polynomials, integration) do not have a 'textbook algorithm.' However, computer algebra specialists have developed non-elementary algorithms for solving such problems. They are implemented in computer algebra systems, and, in many cases, they produce the answer within a reasonable amount of time. Both factorization and integration are cases where the inverse transformation (expanding or differentiation) is algorithmic. Therefore, the authors of a learning environment can try to produce a solution by backward engineering if the answer is known. The answer-finding procedure can also be used in the teacher component of the environment for composition of tasks. School mathematics tasks on factorization are usually supposed to have solutions with integer coefficients, and the teacher needs a tool to test this feature.

3.10 Checking of the final 'solved' form for all task types

Learning environments can behave differently at the end of exercises. In most environments (including Aplusix, MathXpert and T-algebra), the student is required to press a specific button (for example 'Solved') when he/she believes that the solution is completed. The program then verifies the final state and agrees or requires continuation. Some environments finish the work on the task automatically when the expression is in the final form. In both cases, the program should be able to verify whether the suggested expression is in the final form for the current task type.

The authors of computerized environments should be careful with final requirements. Many teachers have only limited experience with using computerized environments, and they continue thinking in the same manner as when working with paper and pencil. For them the requirements for the final answer are much more important than restrictions on making the steps.

Requirements for the final form very much depending on teacherss local traditions and habits. For example, some textbooks recommend decimal fractions for solutions of linear equations, while others prefer common fractions. Some textbooks require that improper fractions should be converted to mixed numbers, while others almost do not use them at all. In many problem types, there can be different requirements for the order of members in the answer if the final expression is a polynomial. Many teachers impose their own additional conditions and sometimes change them during exercises.

On the other hand, typical final requirements are usually quite simple combinations of well-known syntactical properties. Their verification procedures can be assembled from corresponding checkers. Universal toolboxes could contain checkers of necessary elementary properties and assembled implementations of minimal sets of requirements for most common problem types. The policy of minimalism can also be useful in the programming of particular environments. We can implement commonly accepted requirements and leave the checking of more changeable features to teachers. For example, an environment for solving linear equations can accept both decimal and common fractions or mixed numbers and improper fractions as answers. However, such decisions should be clearly documented in teacher manuals.

4. Conclusions and final remarks

We extracted ten necessary kernel functionalities from the description of a 'maximally detailed dialog.' The discussion of these functionalities resulted in the following conclusion: a universal toolbox could contain implementations of features 1-4 and 6 together with features 7, 8 and partially 10. Checking of 5, 9 and some cases of 10 is more sensitive to the particular understanding of operations and the toolboxes could contain syntactical utilities for assembling variable testing procedures.

The author would like to expresses here two further dreams. The first is the availability of a universal transformation engine. We can imagine an environment where the teacher can specify not only the task, but also the set of available rules for building the solution. The transformation engine could then enable teachers to specify the solution algorithm, using an ordered list of rules and possibly some other means. A program like MathXpert probably uses a mechanism of this kind. The same is done in T-algebra but with the addition of some ad hoc cases. The second dream is a universal matching engine that would allow for the finding of matching parts and differences in the previous expression and in the entered result of a step.

Acknowledgements

The author is financed by the grant SF0182712s06 of the Estonian Ministry of Education and the Estonian Science Foundation grant ETF7180.

References

1. Alpert, S.R., Singley, M.K. and Fairweathrer, P.G., Deploying Intelligent Tutors in the Web: An Architecture and an Example. International Journal of Artificial Intelligence in Education. 10 (2), 183-197, 1999.
2. Beeson, M., *Design Principles of Mathpert: Software to support education in algebra and calculus.* In Kajler, N. (ed.) Computer-Human Interaction in Symbolic Computation, Springer, 89-115, 1998. http://www.mathxpert.com/
3. Beeson, M., *MathXpert : un logiciel pour aider les élèves à apprendre les mathématiques par l'action*, Sciences et Techniques Educatives, 9(1-2), 37-62, 2002. English translation: *MathXpert: Learning Mathematics in the 21st Century*. http://www.michaelbeeson.com/research/papers/pubs.html
4. Fisher, T., *Probabilistic Checks for the Equivalence of Mathematical.* A Senior Thesis by. Travis Fisher, 1999 www.cse.unl.edu/~sscott/students/tfisher.pdf
5. Gonnet, G.H., *Determining Equivalence of Expressions in Random Polynomial Time.* Proceedings of the 16th ACM Symposium on the Theory of Computing, 334-341, 1984.
6. Nicaud, J., Bouhineau, D. and Chaachoua, H., *Mixing microworld and cas features in building computer systems that help students learn algebra.* International Journal of Computers for Mathematical Learning, 5(2), 169-211 2004.
7. Issakova, M., Lepp, D. and Prank, R.; *T-algebra: Adding Input Stage To Rule-Based Interface For Expression Manipulation.* International Journal for Technology in Mathematics Education, 13(2), 89-96, 2006.
8. Macintyre, A.,Wilkie, A.J., *On the decidability of the real exponential field.* P. Odifreddi (ed.) Kreiseliana: about and around Georg Kreisel. A.K.Peters, 441-467, 1996.
9. MATH-TEACHER. http://www.mathkalusa.com/index.html
10. Matiyasevich, Yu., *Hilbert's Tenth Problem.* The MIT Press, Cambridge, London, 1993.
11. Matiyasevich, Yu., *On Hilbert's Tenth Problem.* Pacific Institute for the Mathematical Sciences Distinguished Lecturer Series, 2000.

12. Prank, R., Issakova, M., Lepp, D., Tõnisson, E. and Vaiksaar, V., *Integrating Rule-based and Input-based Approaches for Better Error Diagnosis in Expression Manipulation Tasks.* In Shangzhi Li, Dongming Wang, Jing-Zhong Zhang (ed.). *Symbolic Computation and Education*. Singapore: World Scientific, 174-191, 2007.
13. Ravaglia, R., Alper, T., Rozenfeld, M. and Suppes, P., *Successful pedagogical applications of symbolic computation.* In Kajler, N. (ed.) Computer-Human Interaction in Symbolic Computation, Springer, 61-88, 1998.
14. Richardson, D., *Some unsolvable problems involving elementary functions of a real variable*. J.Symbolic Logic 33, 514-520, 1968.
15. Richardson, D., *Solution of the Identity Problem for Integral Exponential Functions.* Zeitschrift für mathematical Logik und Grundlagen der Mathematik. Vol. 15, 333-340, 1969.