JEAN-FRANÇOIS NICAUD, DENIS BOUHINEAU and HAMID CHAACHOUA

MIXING MICROWORLD AND CAS FEATURES IN BUILDING COMPUTER SYSTEMS THAT HELP STUDENTS LEARN ALGEBRA

ABSTRACT. We present the design principles for a new kind of computer system that helps students learn algebra. The fundamental idea is to have a system based on the microworld paradigm that allows students to make their own calculations, as they do with paper and pencil, without being obliged to use commands, and to verify the correctness of these calculations. This requires an advanced editor for algebraic expressions, an editor for algebraic reasoning and an algorithm that calculates the equivalence of two algebraic expressions. A second feature typical of microworlds is the ability to provide students information about the state of the problem in order to help them move toward a solution. A third feature comes from the CAS (Computer Algebra System) paradigm, consisting of providing commands for executing certain algebraic actions; these commands have to be adapted to the current level of understanding of the students in order to only present calculations they can do without difficulty. With this feature, such a computer system can provide an introduction to the proper use of a Computer Algebra System. We have implemented most of these features in a computer system called APLUSIX for a sub-domain of algebra, and we have done several experiments with students (mainly grades 9 and 10). We had good results, with positive feedback from students and teachers. APLUSIX is currently a prototype that can be downloaded from http://aplusix.imag.fr. It will become a commercial product during 2004.

KEY WORDS: algebraic expressions, algebraic reasoning, didactical evaluation, feedback, microworld

1. INTRODUCTION

Since the 1980s, a few ILEs (Interactive Learning Environments) for algebra have been designed and used, at least for experimental purposes (Bundy and Welham, 1981; Foss, 1987; McArthur and Hotta, 1987; Thompson, 1989; Oliver and Zukerman, 1990). Three research groups in particular have carried out long-term research on algebra ILEs. The Carnegie Mellon team (Pittsburgh, USA) developed the ALGEBRA-TUTOR (Anderson et al., 1990) for simple equations, and PAT (Koedinger et al., 1997) for word problems and simple equations. PAT is now a commercial product used in many schools in the USA. The MATHXPERT team created the MATHXPERT system (Beeson, 1990, 1996) for algebra and calculus, now also a commercial product,

International Journal of Computers for Mathematical Learning **9:** 169–211, 2004.

which can deal with a wide range of problems at secondary school and college level. The APLUSIX team realised an ILE[1] (Nicaud et al., 1990) for factorising polynomials. This system has been used experimentally for 10 years (Nguyen-Xuan et al., 1993, 1999), but it remains a prototype. More recently, several other prototypes have appeared, e.g., a web-based tutor for simple equations (Alpert, 1999), and L'ALGEBRISTA (Cerulli and Mariotti, 2000), a microworld in which the operators are theorems.

Since the 1970s, Computer Algebra Systems (CAS) such as MAPLE[2], MATHEMATICA and DERIVE have been developed as systems devoted to formal algebra. These systems are commercial products that solve formal algebraic and numerical problems with powerful methods, and can draw many kinds of 2-D and 3-D graphs. Generally, these systems offer "one-step" solutions – no intermediate calculations are presented. CAS are designed to help professional users of mathematics, and are not directly intended for use in mathematics education. As mathematics educators have found, the use of CAS in education has significant drawbacks and a good understanding of algebra remains essential to use CAS (Ball, 2001). Computer systems for helping students learn algebra can be based on a CAS: the ACTIVEMATH group (Büdenbender et al., 2002) and (Lagrange and Py, 2002) are examples of such work.

In 2000, the APLUSIX group began to investigate a new kind of ILE for algebra. At that time, there were no ILEs for algebra that allowed the student to freely build and transform algebraic expressions, as one can do on paper; the existing ILEs were command-based systems that did not allow the student to proceed without applying a command chosen from a menu. We decided to build such a system, and this paper presents our ideas for the design of such software, and describes the actual APLUSIX system that we built.

For this new kind of ILE, we consider three main characteristics. The first consists of having *advanced editing* to let the student develop his or her own reasoning steps. This includes, on the one hand, an editor for algebraic expressions and, on the other hand, an editor for algebraic reasoning. The second characteristic is *epistemic feedback* through indicators that show the state of expressions and the correctness of the student's calculations. These two characteristics come from the paradigm of microworlds as learning environments, although the second one goes beyond the usual feedback provided in microworlds. (Usually, microworlds are not able to

evaluate the correctness of a piece of reasoning.) The third main characteristic, coming from the CAS paradigm, is the availability of *commands* for having certain calculations performed by the system.

This paper is arranged as follows. Section 2 provides the main goals and principles for the design. Section 3 shows two examples of use of APLUSIX. Section 4 is devoted to the editing of algebraic expressions, a key issue when the goal is to manipulate the same objects as with paper and pencil. Section 5 concerns the editing system for algebraic reasoning, limited to reasoning by equivalence, and the verification of the student's calculations. Section 6 tackles the issue of epistemic feedback in algebra. Section 7 presents the CAS-like commands currently implemented in the APLUSIX system. Sections 8, 9 and 10 describe the tests and experiments we have carried out with APLUSIX.

2. MAIN GOALS AND PRINCIPLES FOR THE DESIGN OF APLUSIX

In this section, we present our main goals and principles for the design of APLUSIX. Before that, we present a model of reasoning by equivalence and we briefly describe three command-based ILEs.

2.1. *A Model of Reasoning by Equivalence*

Reasoning by equivalence is a major reasoning mode in algebra. It consists of searching for the solution of a problem by replacing the algebraic expression of the problem by equivalent expressions until a solved form is reached. The importance of this reasoning mode comes from its capacity to solve many problems and sub-problems in algebra. However, there are other reasoning modes such as necessary conditions, sufficient conditions, recurrence, and so on.

We consider that an *algebraic problem* is given by a *problem type* (e.g., factor a polynomial, solve an equation, calculate a primitive) and an algebraic expression (this term having a wide sense, including equations). So, "Solve $(2x-1)(x+3)+4x^2-1=0$" is an example of an algebraic problem. There is an equivalence relationship between expressions: replacing an expression or a sub-expression in a problem by an equivalent expression provides a new problem having the same solutions. Identities allow transformation of expressions while

maintaining equivalence. For that, the identities are oriented, taking the form of transformation rules (also called rewriting rules). For example, $A^2 - B^2 = (A - B)(A + B)$ may be used in the form of the rule $A^2 - B^2 \rightarrow (A - B)(A + B)$ and the rule $(A - B)(A + B) \rightarrow A^2 - B^2$. There are elementary transformation rules, like $A + 0 \rightarrow 0$, and complex transformation rules, like the use of $b^2 - 4ac$ to factor $ax^2 + bx + c$.

The main inference mechanism in solving an algebraic problem is to apply correct transformation rules that preserve the equivalence of sub-expressions (or the whole expression, which is a particular sub-expression). This mechanism, called *replacement of equals* (Dershowitz and Jouannaud, 1989), can be decomposed into the following steps:

(1) Matching: determine the applicable rules.
(2) Strategy: choose an applicable rule.
(3) Application: apply the chosen rule.

As problems are not always solved in a direct way, there is another strategic aspect that consists of deciding between continuing on the current path or backtracking to a previous step to try another path.

Matching consists of finding a sub-expression *E1*, a transformation rule *R* and a substitution *S* between the variables of the rule and some expressions in such a way that replacing the variables by these expressions in the left hand side of the rule provides an expression equivalent to *E1*. For example, the rule $A^2 - B^2 \rightarrow (A - B)(A + B)$ is applicable to the sub-expression $4x^2 - 1$ of the expression $(2x - 1)(x + 3) + 4x^2 - 1$ because its left hand side $A^2 - B^2$ matches $4x^2 - 1$ with the substitution $[A: 2x, B: 1]$.

Applying an applicable rule *R* with a substitution *S* consists of replacing the right hand side of the rule by the expressions of the substitution and replacing the sub-expression *E1* by the result. In the previous example, one first applies the substitution to $(A - B)(A + B)$ and gets $(2x - 1)(2x + 1)$, then replaces $4x^2 - 1$ by the result and gets the new expression $(2x - 1)(x + 3) + (2x - 1)(2x + 1)$.

Describing in detail the application of a transformation rule *R* consists of providing *R*, *E1*, *S* and the result. At school, usually only partial descriptions of this process are given. Sometimes, instead of indicating the precise rule, one indicates a family (e.g., reduction); often *E1* is not indicated; and *S* is often never shown.

2.2. *Command-Based ILEs for Algebra*

Command-based ILEs are systems in which the student has to choose, at each moment, a command to perform an action. All the existing ILEs for algebra are command-based systems in which the commands are based on transformation rules.

PAT (Koedinger et al., 1997) contains a module devoted to the formal solution of linear equations of one variable. To solve an equation, the student chooses a command corresponding to a transformation rule in a menu. Then she performs a part of the matching by indicating one side of the equation or the whole equation. The rest of the matching is done by the system. The result is provided by the student or the computer, depending on the option setting. This overall mode of interaction is possible because of the narrowness of the domain: (1) there are not many rules, and (2) there are no complex expressions that would require a more precise matching. With PAT, the student can make calculation errors in terms of the "result provided by the student"; matching errors are not directly considered in this system.

MATHXPERT (Beeson, 1990, 1996) is a command-based system that contains a lot of commands corresponding to transformation rules for solving problems from a large domain. In order to avoid the need for selection in very large menus, the author implemented contextual menus. The basic interaction mode is the following: (1) the student selects a sub-expression by drawing a rectangle over it; (2) the system presents a menu composed of all the rules that are applicable to this sub-expression; (3) the student chooses a rule; (4) the system applies the rule. This interaction mode has some drawbacks: some rules are not easy to recognize (due to the difficulty of naming rules that are usually applied without being named in pen and paper calculations); the matching of the system is sometimes limited (see Figure 1) or ambiguous (see Figure 2). With MATHXPERT, the student is prevented from making matching or calculation errors.

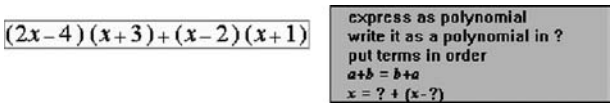

Figure 1. The matching of MATHXPERT does not find the common factor $x - 2$, so the identity $ab + ac = a(b + c)$ is not proposed in the menu. Even advanced students will have first to factor 2 out of $2x - 4$ before factoring $x - 2$ out of the whole expression.

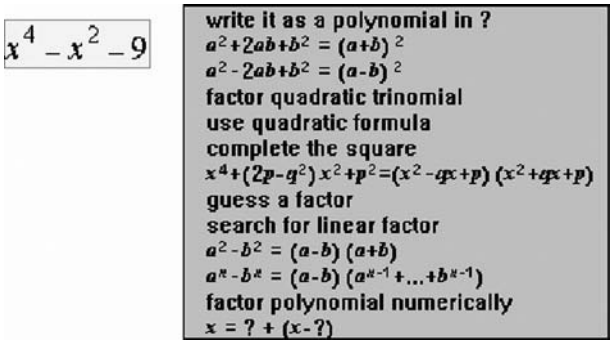

Figure 2. The matching of some rules is done by MATHXPERT on a sub-part of a sum. This example is a case of the rule coming from the identity $a^2 - b^2 = (a - b)(a + b)$. In the above situation, this rule is applicable to $x^4 - x^2$ and to $x^4 - 9$ and the application is made on $x^4 - x^2$; as it is not possible to select $x^4 - 9$, it is difficult to apply the rule on $x^4 - 9$ (one needs first to select $-x^2 - 9$ and to commute it with the identity $a + b = b + a$).

The first ILE we developed (Nicaud et al., 1990) was devoted to factorising polynomial expressions. It was a command-based system in which some of the rules were grouped under the two concepts *reduction* and *expansion* and in which the rules for *factorisation* were detailed rules. The idea was to require the indication of a precise rule for the rules under study by the student and just the indication of a group for the well-known rules. Matching was done by the student in an explicit way, as shown in Figure 3. The calculation was performed by the system. With this system, the student could make matching errors but could not make calculation errors.

Note that the aim of these short descriptions has been to show three different interaction modes based on a common idea

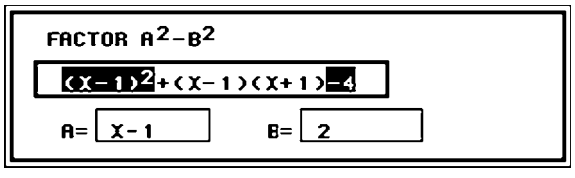

Figure 3. The student chooses Factor $A^2 - B^2$ in the action menu. Then this window is opened and the student has to select a sub-expression and to input the values of the variables of the rule. When the command is incorrect, informative feedback is provided; when the command is correct, it is applied by the system, going forward a step in the calculation.

(commands based on rules) in order to establish the background for a new interaction mode for another kind of system. Our aim here is not to criticise the effectiveness of these other systems, which have empirically proved helpful in learning algebra.

A natural question arises: Why has the general paradigm for building ILEs for algebra been limited to the use of commands associated with transformation rules? We think that the main reasons are: (1) this is a very natural method for ILEs designers, as transformation rules are the fundamental mechanisms for solving algebraic problems; (2) this is the easiest method when the domain is not limited to linear expressions; it avoids the building of complex tools for editing algebraic expressions and for verifying calculations.

2.3. *Our Mains Goals and Principles*

The designers and developers of the APLUSIX system, the two first authors of this article, are researchers in computer science. They began to work on this project in September 2000 in the University of Nantes. In September 2001, they moved to the Leibniz laboratory at Grenoble, joining a team devoted to mathematics education. The third author is a researcher in mathematics education of this team mainly working on experiments of APLUSIX in middle and high schools.

The first main goal of the designers was to develop an ILE for algebra allowing the student to freely build and transform algebraic expressions, and providing epistemic feedback that can help in learning algebra. One classical situation in an algebra classroom is the following: The teacher gives exercises to the students who try to solve the exercises on paper, writing their calculations without constraints. The teacher goes from student to student and acknowledges the correctness of a student's solution or helps correct errors. We wanted to implement a model of activity close to this one, with a system having at least a mode with the following properties: (1) it allows easy inputting of the student's calculational steps; (2) it verifies the student's calculations at each step; (3) it verifies the correct termination of the solution. We had in mind allowing at least the two following situations:

(1) the student is working in a computer class with a teacher: In this case the student can have feedback at each step; the teacher no longer has to verify the student calculations and can better use his or her time;

(2) the student is working alone at school or at home: in this case, s/he is no longer able to receive personal teacher feedback.

Let us call this way of functioning of the system in both these situations the *basic training mode*.

The second main goal was to create an ILE that could be widely distributed and used. As designers of advanced ILEs for algebra, we were not satisfied by the situation in which we found ourselves, building ILEs used only for experiments: This is too much work for too little use. This goal requires building an easy to use and useful system. We considered two main points for usefulness: Encompassing a large mathematical domain and having several modes of functioning (the basic training mode, a test mode without feedback, a mode having a few CAS-like commands, etc.).

2.3.1. *The Microworld Idea*

The first goal led us to enter the mathematical microworld paradigm. According to Laborde (1989), a microworld is a world of objects and relationships between objects. A set of operators allows acting on objects and creating new objects with new relationships – and direct manipulation is a possible way for acting on objects. Balacheff and Sutherland (1994) introduced the idea of epistemic domain of validity, concerning the connections between the representations and their meanings; we apply their ideas to APLUSIX in the Conclusion. According to Thompson (1987), the function of mathematical microworlds is not directly to instruct the student, rather they must facilitate the construction of objects and relationships and allow the student to concentrate on the construction of meanings.

We consider microworlds for algebra to have the following characteristics:

– the concrete objects are algebraic expressions;
– the basic relationship is the structural relation (Kieran, 1991), i.e., the composition of expressions with operators (e.g., 12 is the first argument of the second argument of $\sqrt{x}/(12 + x^2)$);
– a second relationship is the equivalence between expressions;
– operators of the *first type* are linked to the structural relation, they allow the editing of algebraic expressions in a way that takes care of the structure, e.g., $x + 2$ cannot be selected in $3x + 2y$; the replacement of x by $y - 2$ in $3x + 4$ provides $3(y - 2) + 4$;

– operators of the *second type* are linked to the equivalence of expressions, they are mechanisms that produce steps in reasoning.

Command-based ILEs can be seen as microworlds using operators of the second type. This is the view for example of Cerulli and Mariotti for L'ALGEBRISTA (Cerulli and Mariotti, 2000). In this paper, we consider systems having at least operators of the first type and some direct manipulation mechanisms.

2.3.2. *The Educational CAS Idea*

CASs are command-based systems in which the commands can be regarded as types of problems: The user asks the system to expand or factor an expression, to solve an equation, etc. The idea of an educational CAS is to implement such commands in limited areas, for example implementing a command performing numerical calculations only over the integers or a command solving equations only when they are linear. In this context, the student has to do part of the work (at the higher level) whilst the computer accomplishes the remaining work. These commands have important differences from commands based on rules: (1) just a few commands are necessary (see Section 7); (2) the commands are easy to understand because the teacher gives names to the types of problem; (3) they are not obligatory – the student can work without them – and they can be hidden when the teacher wants to avoid scaffolding. The use of an educational CAS can also be seen as a transition between the usual school situation, where the student performs all the calculations, and the use of a CAS at university level, where the computer software may solve the entire problem.

2.3.3. *The Parameterisation Principle*

An ILE may be seen as a computer system providing tools, feedback and constraints to the student. The parameterisation principle consists of having large sets of tools, feedback and constraints that can be active or not, depending on parameters (which can also be called option settings) which are set for some duration by someone (the teacher, a tutor, or the student). Furthermore, lists of exercises may be set-up, associated with particular values of the parameters, which are appropriate for the student to solve in a particular situation. This principle has been applied in the development of the APLUSIX system.

2.3.4. *Our Teaching Ideas and Goals*
The design of APLUSIX was guided by the following ideas:

– the learning of algebra requires an important element of practice exercises;
– technology can help the learning of algebra by producing and assembling good tools; it can help both the student and the teacher.
– In order to have a chance to be widely used, ILEs for algebra must:

• be based on easy-to-use tools,
• be linked clearly to fundamental algebraic properties,
• allow students to make errors, and provide feedback on them,
• apply to a relatively large domain of algebra.

Our main question then was: What are the good generic tools that we can build and which can help the learning of algebra by providing good interaction and feedback?

These ideas are not directly linked to a particular learning theory. The choice of a particular combination of tools (by setting different options), the choice of the exercises and the way a teacher intervenes during the sessions may suit one learning theory or another. For example, the teacher can prepare a situation using the ILE which favours the discovery of some property by students or, in contrast, use the ILE for drill and practice.

3. TWO EXAMPLES OF THE USE OF APLUSIX

In order to help in understanding the rest of the paper, we describe here two examples of the use of APLUSIX. The first example, Figure 4, concerns the basic training mode and is extracted from the experiment described in Section 9. The second one, Figure 5, concerns the use of CAS-like commands.

4. EDITING ALGEBRAIC EXPRESSIONS

Until recently, editing algebraic expressions in computer systems was very awkward. Most of the systems used a 1D (one dimension) editor and a 2D display: The user enters an expression as a string, e.g.,

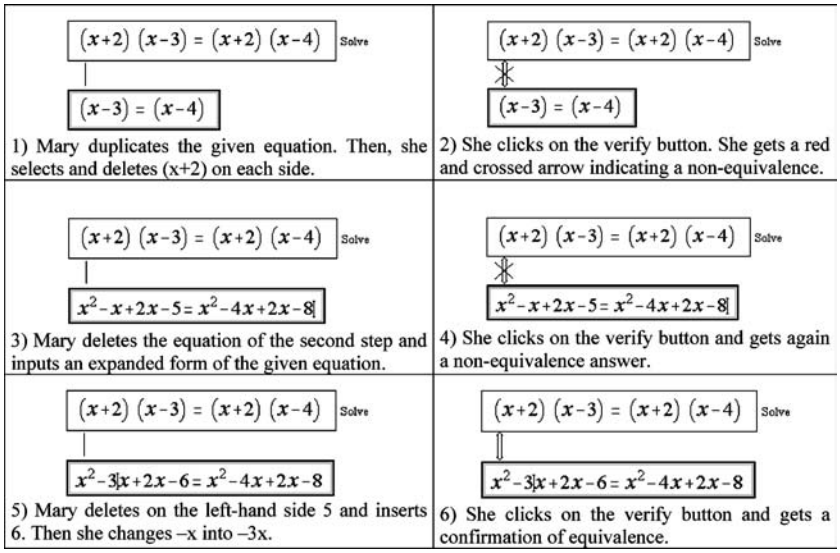

Figure 4. Beginning of the solution of the exercise "Solve the equation $(x+2)(x-3) = (x+2)(x-4)$" by a real student, Mary, grade 10, in the basic training mode. The equivalence between two steps is indicated to Mary when she clicks on the verify button.

(3*xˆ2)/(y + 1), after that, the system displays the expression in two dimensions. Recently, 2D editors have begun to appear.

In this section, we will analyse the main functions of 2D editors for algebraic expressions, i.e., editors that display the expressions in the usual 2D representation and that allow modification of expressions in this representation. Our basis for this description is twofold: For each action that can be executed by an editor, we consider on the one hand the general principle that is used by most computer systems for this action, in particular by text editors (e.g., the general principle for *delete over a selection is suppress the content of the selection*), and on the other hand the structure of the algebraic expressions that suggests certain behaviours. In the following we often say *The natural idea for … is …* in which the natural aspect is based on our opinion, that is, what we consider to be natural behaviour.

4.1. *The Syntax of Algebraic Expressions*

In Rewriting Rule theory (Dershowitz and Jouannaud, 1989), an algebraic expression is defined recursively as the application of an

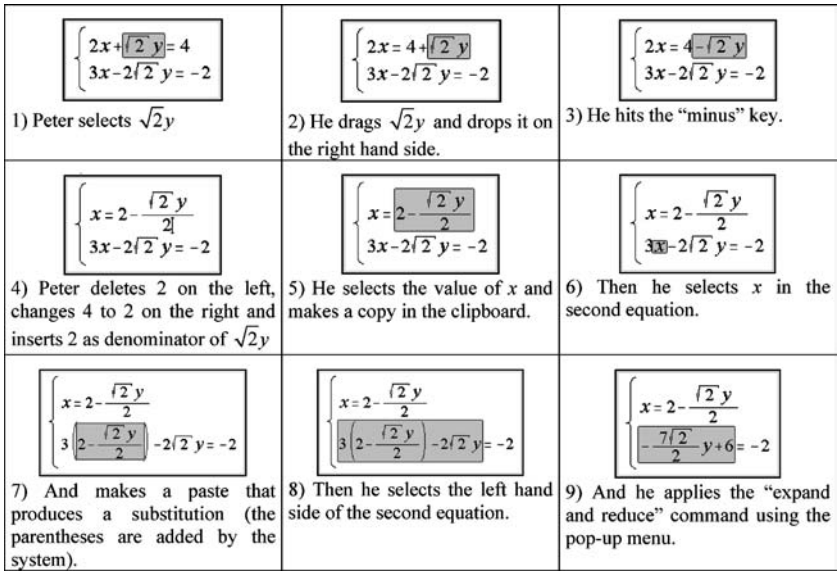

Figure 5. Example of how to solve a system of equations using drag&drop and the command ''Expand and reduce''.

operator (like $+$, *, ^) to arguments, respecting arity (i.e., number of arguments) and expression types. The arguments are expressions. The elementary expressions are digits and symbols (like x, m, π). The operators are mechanisms that build expressions, not functions that calculate something. For example, the $+$ operator for building expressions applied to 2 and 3 generates ''(+ 2 3)'' in a prefixed-parenthesised representation ($2+3$ in the usual representation). Note that this must be distinguished from the addition function that gives 5 when applied to 2 and 3. The representations for algebraic expressions which are close to this definition are the prefixed-parenthesised representation and the tree representation, because there are natural and unambiguous mappings between these two representations, and with the definition. For example, let's take the expression $3x^2 + 2x$ (in the usual representation), this expression is defined as the application of $+$ on two expressions b and c, where b is the product of 3 and d, d is the power of x with exponent 2 and c is the product of 2 and x. The prefixed-parenthesised representation is (+ (* 3 (^ x 2)) (* 2 x)) and the tree representation is shown in Figure 6.

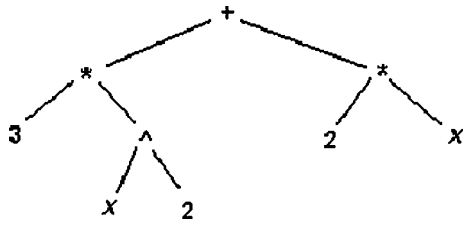

Figure 6. A tree representation.

Expressions are viewed in a broad sense in this paper, in the same way that they are in rewriting rule theory. Thus for example Boolean expressions are included, e.g., $2x = 3$ *or* $3x = 1$ is an expression with the operator *or* applied to the Boolean sub-expressions $2x = 3$ and $3x = 1$.

In this syntax, expressions may be *well-formed* or not. A well-formed expression is the application of an operator to other well-formed expressions respecting the arity and types. An expression is ill-formed when the number of arguments of an operator is incorrect (missing arguments or too many arguments) or the type of an operator is incorrect (e.g., *or* applied to a non-Boolean expression). Some types may have a contextual definition, for example exponents could be restricted to the type "natural number" at some learning level.

4.2. *The Usual Representation of Algebraic Expressions*

The usual representation of algebraic expressions is a 2D representation. It consists of symbols and lines placed in a 2D display. This organisation contains an important 1D sub-organisation: some parts are sequences of symbols that can be seen as text, and some operators can be seen as text operators because they are placed on the left, on the right or between their arguments (e.g., $+$, $-$, sin). Other operators have specific places for their arguments in 2D space (e.g., power, divide, square root). The usual representation also has hidden operators (e.g., *times* in $2x$, *composition of integers* in 23), and implicit priorities between operators and parentheses.

From a 2D view, an expression can be seen as an organisation of text sequences and boxes having invisible borders, cf. Figure 7.

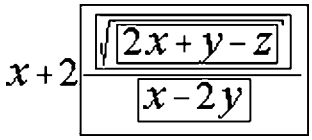

Figure 7. Text and box structure in the 2D view. The borders of the boxes have been drawn.

4.3. *The Text&box Mode and the Structure Mode*

Let us now consider designing a 2D *editor* of algebraic expressions for a computer having a keyboard and a mouse. By 2D *editor*, we mean a system that displays algebraic expressions in the usual 2D representation and that allows modification of expressions in this representation.

We wish to consider the classical actions of an editor: place the cursor (the insertion point), move the cursor, select part of an expression, input or delete at the cursor position or over a selection, copy or cut a selection, paste at the cursor position or over a selection, and *drag&drop*.

In *text&box* mode, expressions are seen as strings of characters or boxes. If $x + 3$ is copied in the clipboard and if the cursor is between 2 and y in $2yz$, a paste in the *text&box* mode provides $2x + 3yz$, and the pasted expression is *not* a sub-expression of the result.

In *structure* mode, the actions are executed with respect to the algebraic structure. For example, if $x + 3$ is copied in the clipboard and y is selected in $2yz$, a paste in the *structure* mode provides $2(x + 3)z$ because the sub-tree y of $2yz$ is replaced by $x + 3$.

There is a third mode which can be envisaged where the expressions are seen as parts of the 2D space. In such a mode, a *drag&drop* of y in xy, a little above the expression, would produce x^y. We will not analyse this mode any further, and just consider below the *text&box* mode and the *structure* mode.

Designers of editors of algebraic expressions have to assign a mode to every action. They can also implement some actions in several modes and leave the choice of the mode to the user.

4.4. *Missing Arguments*

An editor should not only work with well-formed expressions. For example, when we input $a + x$ by typing successively a, $+$, x, we have an intermediate stage of an ill-formed expression $a+$ that has a

missing argument. This is not a problem in a *text&box* mode but it is a problem in a *structure* mode, because of the underlying tree structure. A natural solution for this problem is to add empty arguments represented by question marks or small boxes. This is the choice for example of the editor in FRAMEMAKER which displays $a + ?$ when $a+$ has been typed, and this is also our choice for APLUSIX.

4.5. *Input at the Insertion Point*

In the *text&box* mode, there are two ways to execute an input. The first is to insert a character at the cursor position, and the second is to create a box for a box-operator such as a fraction or a square-root. For example, MATHTYPE works in this mode with a virtual keyboard for box-operators and special characters (see Figure 8). In this mode, some operators may have two representations, e.g., for fractions (Figure 8).

In the *structure* mode, the juxtaposition of arguments has to be interpreted in any situation to get the underlying tree representation. A natural interpretation consists of viewing a juxtaposition as a construction of a number for digits (e.g., the input of 4 after 2 in $2x$ provides $24x$ such that the integer 24 is built), as the construction of a function symbol when this is possible (e.g., the input of *n* after *i* in *six* provides *sin x* so that the function symbol *sin* is built), and otherwise as a product.

In the *structure* mode, there are no unbalanced parentheses. One gets two balanced parentheses around an empty argument when one clicks on the parentheses button or strikes a parenthesis key. This is different from the usual left to right typing and has the following important drawback. Let us consider that we have the expression $(x + 2)(x + 1) + (x + 3)$ and that we want to change this to $(x + 2)((x + 1) + (x + 3))$; we cannot simply insert a "(" at one place and a ")" at the other place.

Figure 8. a part of the virtual keyboard of MATHTYPE. This system allows to input a "unconventional" expression such as $(2 + 3\left(\frac{a}{b} + c/d\right) + 5)$ in which the small parentheses and the "/" are made by key strokes while the large parentheses and the 2D fraction are made using the buttons of the virtual keyboard.

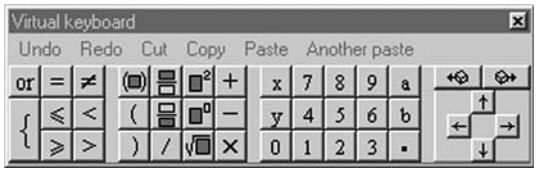

Figure 9. The virtual keyboard of APLUSIX: There are six buttons for box- operators.

In the *structure* mode, we can propose to respect at all times the types of the arguments, refusing to allow $2x = 3$ *or* $3x$ because $3x$ is an argument of *or* and is not a Boolean expression. However, this is a natural intermediate step when we input $2x = 3$ or $3x = 2$ one character after another. If we want to have the constraint on argument types, the above expression would have to be typed in a particular sequence: input "=" after *or*, providing $2x = 3$ *or* $? = ?$, then input the two arguments of "=". So the argument type constraint in *structure* mode can lead to rather complex rules for input.

Our choices for APLUSIX are mainly oriented towards the *structure* mode in order to get more "algebraic" behaviour from an editor. However, we consider that the constraints of the *structure* mode for the input are too strong and too far from the natural writing of algebraic expressions. That is why we based input at the insertion point on the *text&box* mode and used a virtual keyboard with buttons for box-operators as well as for the other operators, digits and letters (see Figure 9).

4.6 *Selection*

In the *text&box* mode, any succession of strings or boxes can be selected. So a selection in a well-formed expression is not necessarily a well-formed expression. And even when a selection is a well-formed expression, it is not necessarily a sub-expression (see Figure 10).

In the *structure* mode, the natural way to implement the selection mechanism is to allow the selection of one or more sub-expressions

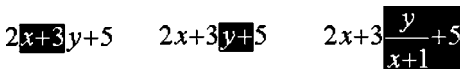

Figure 10. Three examples of selection in the *text&box* mode from MATHTYPE.

Figure 11. Two examples of selection in the APLUSIX structure mode. In the second example, after having selected $2x$, a control-click anywhere over $-5x^3$ extends the selection.

and nothing else (see Figure 11). This selection is then, a true algebraic selection.

We believe that selection in the *structure* mode is the best selection mechanism both for advanced users, who generally work on sub-expressions, and for beginners who may be learning about the structure of expressions through selection. We therefore implemented this structured selection mechanism in APLUSIX as follows: When two parts of an expression are covered by a mouse movement, the smallest sub-expression containing these two parts is included in the selection. For example, in $(2x + y)(z + 1) - 2$, let us drag left to right over x, x is selected; when we continue over $+$, the selection is $2x + y$, when we continue over "$)$", the selection is $(2x + y)$, when we continue over "$($" the selection is $(2x + y)(z + 1)$.

4.7. *Input Over a Selection*

The general principle for implementing an input over a selection consists of replacing the selection by the input, whatever it is. In the *text&box* mode, this is quite natural.

In *structure* mode, we can separate the case of the input of a digit or a letter (which are elementary well-formed expressions) from the case of an operator; we can use the general principle for the first case, and another principle for operators which is *apply the operator to the selection*. For unary operators, like structured parentheses, minus and square root, this is very natural, e.g., $2x$ being selected, a click on the square root button provides $\sqrt{2x}$. For operators having more than one argument, an argument must be chosen as the "main" argument. For example, $2x$ being selected, a click on the fraction button provides $2x/?$, the main argument of a fraction taken to be its numerator. Note that this action may introduce parentheses, e.g., the application of "–" to $x + 2$ produces $-(x + 2)$.

MATHTYPE implements the *text&box* mode for key strokes and *application of the operator* for most of the operators of the virtual keyboard. So, $2x$ being selected, a key strike on "(" provides "(" while a click on the parentheses button provides$(2x)$.

For APLUSIX, we implemented the *structure* mode with the following features:

- when the selection begins by a minus sign, the input of a minus sign cancels this minus sign, e.g., applying "–" on $-2x$ provides $2x$ instead of $-(-2x)$; we think that this corresponds to the expectation of users in most situations;
- we implemented two fraction buttons, one for putting the selection as a numerator, the other for putting the selection as a denominator (see Figure 9);
- when the operator is a relational operator ($= \ < \ \leq \ > \ \geq \ \neq$) or an operator having a variable arity ($+$, $\times$, *and*, *or*), we decided for the program to do nothing, in view of the fact that there is no natural way to do such input over a selection.

4.8. *Delete*

The general principle for implementing *delete* on the right or the left of the insertion point consists of deleting the object (character or box) at this place and, for implementing *delete* over a selection, of suppressing the content of the selection. In the *text&box* mode, this is natural, but it is sometimes awkward. For example, if we want to *delete* the denominator of a fraction, keeping just the numerator without the fraction line, we must select the numerator, perform a copy, select the fraction, hit the *delete* key, then paste the clipboard. MATHTYPE, for example, implements this behaviour using *text&box* mode.

In *structure* mode, the natural method consists of deleting an argument of an operator and either replacing it by an empty argument or deleting the operator. For example, the cursor being before y in $x+y$, delete can provide $x+?$ or x. When the cursor is over a question mark, it is natural to suppress the operator, e.g., with $x/\sqrt{?}$, the question mark being the cursor point, a first *delete* provides $x/?$ and a second *delete* provides x. This is a better way to solve the problem mentioned in the previous paragraph, and we implemented the *structure* mode in APLUSIX.

4.9. *Copy, Cut, Paste, Drag&Drop*

The general principle for *copy* and *cut* is:

– *Copy* consists of placing a copy of the selection in the clipboard,
– *Cut* is *copy* plus *delete*.

This is a natural method for algebraic expressions. The difference between *text&box* mode and *structure* mode comes from the selection. The consequence is the following: After *copy* or *cut*, when the entire expression is well-formed, *structure* mode always put a well-formed expression in the clipboard while *text&box* mode can enter an ill-formed expression.

The general principle for *paste* at the cursor position is to insert the content of the clipboard at this position. In the *text&box* mode, this is natural, but it may have strange results. For example, *paste* $x + 1$ after 2 in $2y$ provides $2x + 1y$. *Paste* at the cursor position in *structure* mode needs an operator. For example, *paste* $x + 1$ after 2 in $2y$ with the operator $\times$ provides $2(x + 1)y$ and with the operator $+$ provides $2 + x + 1 + y$. There are two main ways to choose the operator: the first consists of presenting a menu to let the user choose the operator; the second consists of choosing an operator using a principle such as considering the operator at the cursor position.

The general principle for *paste* over a selection is to delete the selection and to insert the content of the clipboard at this position. In *text&box* mode this is natural, while in structure mode the natural method is to replace the selected sub-expression by the clipboard. For example, a *paste* of $3y + 1$ over x in $2x + 5$ provides $2(3y + 1) + 5$ performing an algebraic substitution of x by $3y + 1$.

The general principle for *drag&drop* is *cut a selection* and *a paste* at the cursor position. This is a natural idea for both *text&box* mode and structure mode. The behaviour is different between the two modes because *paste* at the cursor position is different.

MATHTYPE implements the *text&box* mode for *copy*, *cut*, *paste*, and *drag&drop*.

We implemented *structure* mode in APLUSIX for these functions. Concerning the choice of the operator for *paste*, we decided to only consider operators of variable arity (composition of numbers, $+$, $\times$) in non-Boolean expressions and (*and*, *or*) in Boolean expressions, and to choose the most appropriate, taking into account the operator at

the cursor position and the main operator of the expression to be pasted. We implemented a menu item called *another paste* allowing changing the operator.

4.10. *Discussion*

Although APLUSIX is a computer system for education, our thinking process and our choices were not oriented toward education for the editor because we consider that the writing of algebraic expressions to be universal and that it is better to design a universal editor. Of course, the editor of APLUSIX is not universal because many standard operators are missing (trigonometry, exponential, integral, matrix, etc.), but the implemented mechanisms are universal.

In the development of ideas for an editor and its implementation in APLUSIX, we made many choices. We did not perform experiments beforehand in order to compare different possible choices for the following reasons: (1) we were guided by the idea of an algebraic microworld with operators of a first type linked to the structural relation of the expression (cf. Section 2.3.1); (2) our professional background in mathematics and school teaching gave us some expertise in making choices; (3) experiments done *before* development may provide misleading results because they are not carried out in a real situation, as the system is not available; (4) the editors of existing systems provide examples of behaviour that can be adopted, and improved particularly from an algebraic point of view.

5. ALGEBRAIC REASONING BY EQUIVALENCE

We have chosen to limit the scope of APLUSIX to reasoning by equivalence, as presented in Section 2.1. This is also the case of the other ILEs for algebra with which we are familiar. Within this model, the inference mode is the application of transformation rules that preserve the equivalence.

5.1. *The Denotation of Algebraic Expressions*

Algebra is a domain with fundamental semantics, usually called denotation (Arzarello et al., 2001; Nicaud and Bouhineau, 2001). It is based on a set of numbers K (in education, this set is successively expanded during the learners' progress: integer, decimal, rational,

real, complex numbers). The denotation of a non-Boolean expression E with variables $x_1, \ldots, x_n$ is the function from K^n to K that associates E to $x_1, \ldots, x_n$. For a polynomial expression, an alternative denotation of the expression E is a polynomial of n variables. The denotation of a Boolean expression E with variables $x_1, \ldots, x_n$ is the function from K^n to $\{false, true\}$ that associates E to $x_1, \ldots, x_n$. An alternative and equivalent denotation of the expression E is the set of solutions of E in K. The denotation defines the equivalence between expressions: Two expressions are equivalent if and only if they have the same denotation.

5.2. *Presentation of Reasoning*

Algebraic reasoning can be presented by mixing natural language and algebraic expressions. At school, students are invited to place the expressions of their calculations on successive lines, generally with commentaries that indicate which transformation rules are applied. The notion of reasoning by equivalence is more or less explicit, depending of the curriculum. Sometimes, the implicit framework is reasoning by equivalence, and the presentation does not explain it. This is the case when students solve systems of equations, making calculations first on one equation then on another.

The concept of backtracking is connected to reasoning by equivalence: when a sequence of calculations is not promising, one can go back to a previous step and try another direction with another transformation rule. This is a strategic issue. It is not necessary to delete steps 3 and 4 of a reasoning to execute a backtrack to step 2 if steps 3 and 4 are correct: The new direction taken in step 2 may be a dead-end so that finally one has to continue from step 4. This concept of backtracking is mostly absent from the school curriculum. Furthermore, it is difficult for students in a paper-pencil context, because when they find themselves at a non-promising step, the reason may be that their calculations are incorrect. So students generally go back to a previous step and cross out all the following steps.

In APLUSIX, we chose to have a strong reification of reasoning. Expressions are placed in boxes and lines are drawn between boxes, whose first meaning is *this box is obtained from that box*. A second meaning is given by these lines. As the calculations are made by the student, they may be correct or not. When they are correct, the two expressions are equivalent and the line is drawn as an equivalence sign for Boolean expressions and an equal sign for non-Boolean

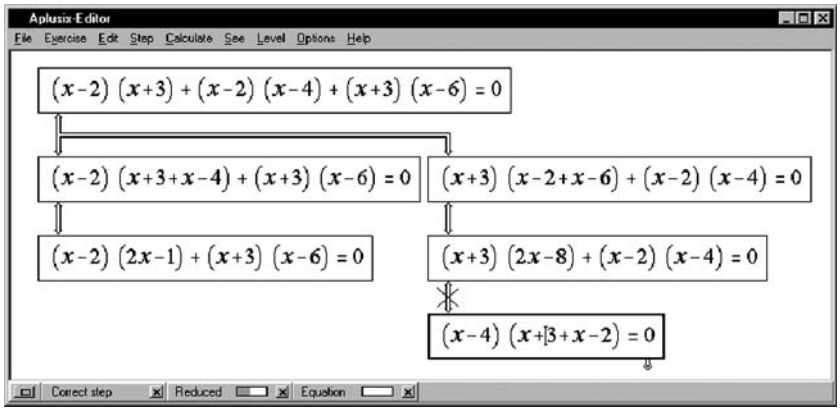

Figure 12. Peter is at a level where efficient methods for factoring $ax^2 + bx + c$ forms are not known to him. At this stage, a good strategy for factoring polynomial expressions consists of applying factorisation rules. So Peter used first the partial common factor $x - 2$ expecting a common factor afterwards. As his expectation failed, he went back to use the other partial common factor $x + 3$. His calculations are correct, except the last one. The error is indicated by a red cross over the equivalence sign.

expressions. When they are not correct, the two expressions are generally not equivalent and the line is drawn as a red equivalence sign, or equal sign, with a red cross. Figure 12 show a piece of reasoning containing a backtrack which was made by a student with APLUSIX.

This form of presentation of reasoning, with links between steps and a tree when a backtrack occurs, is also present in ALGEBRALAND and was used in our first ILE. In MATHXPERT no links are presented and, although the system treats more difficult problems, it does not include backtracking.

5.3. *Verification of Equivalence*

Currently, APLUSIX calculates equivalence of expressions over the real numbers, by calculating the denotations of the expressions to be compared. For rational expressions, we calculate the rational fraction as a pair of polynomials (for any number of variables and any degree). For polynomial equations and inequalities of one variable, we calculate the solution sets (currently only for degree less than or equal to 4 with the discriminant, Cardan's and Ferrari's formulae). We transform rational equations in polynomial equations. For systems of

linear equations, we calculate the solution sets (maximum 10 equations and 10 variables).

As far as we know, this is a novel feature of APLUSIX compared with other educational software for learning algebra. Other systems are action-driven, with the action being performed by the computer. In such conditions, since the student does not input his/her own results then the system does not need to verify the calculations. Note that the indication of the equivalence is linked to an option setting in APLUSIX. The teacher can choose to activate continuous verification (that is, recalculated at each modification), or verification on demand (when the student clicks on the *verify* button), or to deactivate verification altogether.

There is a difference between correct calculations and equivalent expressions. The correct use of a rule, coming from an identity, always produces an equivalent expression, but an incorrect rule sometimes produces an equivalent expression. For example, if a student "moves" $3(x-2)$ from the right to the left of $6x-12=3(x-2)$ without changing the sign, s/he makes an incorrect application of a transformation rule and gets $6x-12+3(x-2)=0$. However, in this particular situation, the result is equivalent because both equations have the unique solution 2. Fortunately, such a situation is very rare (for such moves in polynomial equations, the moved expression must be equal to zero for any solution of the equation).

6. EPISTEMIC FEEDBACK

Feedback is a major feature of microworlds. The first level of feedback consists of displaying objects after they have been modified. The second level consists of providing conceptual or semantic information (Hollan et al., 1984).

Concerning formal algebra, important semantic feedback concerns the *correctness* of the calculations. Because it is linked to denotation, we discussed this issue in the previous section.

Other interesting elements of feedback are indications about general properties of expressions having an important role for many problem types. For example, *reduced* is a basic concept, and the *sort* concept is interesting because there are many situations where there is a preferred order for commutable expressions.

A very important element of feedback is the *solved* property: Telling the student whether the problem is solved or not. In fact, this

concept depends on the problem type so a *solved* feedback can be implemented for certain problem types, but a general feedback mechanism for *solved* cannot be implemented.

When we go into specifics, we do find a major problem concerning these properties. Are they epistemic (pure algebraic properties) or didactic, in the sense that constructions change from one teacher to another, or from one level of learning to another?

In the following we describe some concepts we studied and implemented in APLUSIX.

6.1. *The Factored Property*

Let us analyse in some detail the concept of an expression being *factored*. This concept is often introduced this way: *An expression is factored if it is expressed as a product*. At some level, problems given to students can be arithmetic factorisations (e.g., $3 \times 4 + 3 \times 7$) or algebraic (such as $3a + 6b$ or $ab + 2a$). At a higher stage, students may have to factor $x(x+2) + 2(x+2)$, where $(x+2)(x+2)$ is a result and $(x+2)^2$ is a better result, because it is reduced. But this last expression is not a product. It is also natural to consider that $-(x+2)^2$ is *factored* although it is not a product and that $x+1$ is *factored* because one cannot produce a more factored form. Is $3(x^2+4x+4)$ factored? Yes, in a sense, but it can be further factored to $3(x+2)^2$ and in many cases non-constant factors are more important than constant factors. Thus, the *factored* property is basically didactic because it changes from one level of learning to another.

Implementing didactic concepts in a generic ILE is a problem. One solution consists of choosing among the different possibilities during the program design. But it is possible that some teachers will not like the choices that are made. Another solution consists of implementing several forms of a concept and options for choosing the form to be applied by the program. We believe it would probably be difficult for the teacher to choose the options and even more difficult for the student. Hence for APLUSIX we chose the first solution, considering that the second is contrary to some of our ideas presented in Section 2.3.4 (to have a system that is based on easy-to-use tools, and linked to strong algebraic properties). This led us to define two concepts for *factored*: *P-factored* and *N-factored*.

P-factored (P standing for polynomial) is limited to polynomial expressions of one variable and this property indicates whether a

polynomial expression has the form of a product of prime polynomials. This property depends on the set of numbers used (prime polynomials being first degree polynomials over the complex numbers and first degree polynomials or second degree polynomials having a discriminant less than 0 over the real numbers). Syntactically, a *P-factored* expression is a composition of prime polynomials with the operators *times*, *power*, and *minus*. This is a Boolean definition of the concept. *P-factored* can also be defined as a sort of degree indicating the number of prime polynomials that are factored (e.g., value 0 for x^3–x, 1 for $x(x^2$–1), 2 for $x(x$–1)$(x+1)$). We implemented this form of *P-factored* in APLUSIX with a gauge to indicate the ratio between the number of prime polynomials that are factored over the number of prime polynomials of the totally factored form.

N-factored (*N* stands for numerical) is limited to expanded polynomial expressions of one variable and indicates whether numbers have been factored or not. There is not a unique way to define *N-factored*, for example:

- Is $2x^2 + x/2$ *N-factored* or do we need to factor it into $\frac{1}{2}(4x^2 + x)$ or into $2(x^2 + x/4)$?
- Is $2x + \sqrt{2}$ *N-factored* or do we need to factor it into $\sqrt{2}(\sqrt{2}x + 1)$?

Numerical factors are clear at a syntactic level when the same numerical expression is a common factor, e.g., in $(2+\sqrt{2})$ $x^2 + (2+\sqrt{2})x$. Numerical factors are also clear at a semantic level over the integers, e.g., in $15x^2 + 6x - 9$. We chose to combine these two features to implement the N-factored property in APLUSIX.

6.2. *Some Other Properties*

A *reduced* property can be based on a set *R* of rewriting rules having a reduction status (in a broad sense). Examples of reduction rules are:

$$\mathrm{A} + 0 \to \mathrm{A}, \qquad -(-\mathrm{A}) \to \mathrm{A}, \qquad \mathrm{A} \to \mathrm{A}^2, \qquad \frac{A}{AB} \to \frac{1}{B} \ .$$

In order to create a strong concept of reduction, arithmetical calculations (e.g., replacing $3+4$ by 7) and arithmetical

simplifications of fractions (e.g., replacing 12/15 by 4/5) must be seen as reductions. So must the grouping of like terms in a sum (e.g., replacing $2x^2 + 3x^2$ by $5x^2$). In this framework, an expression e is *reduced* if there is no rule of R applicable to a sub-expression of e.

Such a concept has been implemented in APLUSIX with a gauge depending on the number of reduction rules that are applicable.

We implemented a *sort* property in the same way, using a set of sort rules. Among these are: commute A and B in a product if A is a number and B is not; commute two monomials A and B in a sum if the degree of A is less that the degree of B.

We also implemented an *expanded* property based on the number of parentheses in the expression.

We placed an indication of the state of the current step in the status bar of the system. This state may be "unachieved" (e.g., when there are question marks), "undefined" (e.g., when a denominator is 0), "out of domain" (e.g., for an equation of fifth degree) or "well-formed"

6.3. *The Solved Property*

The solved forms of some problem types can be expressed in terms of the above properties. For example, *Factor a polynomial expression* may be seen as getting an equivalent form that is *P-factored* and *Reduced*. However, the solved forms of equations cannot be expressed with these concepts, so we implemented another property called *Equation* that indicates a degree of progression towards a solved form for equations, inequalities and systems of linear equations. The degree for equations is based on the fraction of decomposed forms and expressions of the form $x = a$ (e.g., the degree increases when $(x + 3)(x - 2) = 0$ is transformed into $x + 3 = 0$ or $x - 2 = 0$ and when $x - 2 = 0$ is transformed into $x = 2$).

7. CAS-LIKE COMMANDS IN APLUSIX

7.1. *Commands*

Our idea for APLUSIX was to have commands that look like the commands of a CAS, corresponding to the types of problem we wanted to present to students. We also wanted to have more powerful commands for advanced students and less powerful commands for

less advanced students. In the current version, we have implemented four commands: *calculate, expand-and-reduce, factor*, and *solve*. The *calculate* command means calculate the canonical form of a numerical expression. An option setting allows for activation or not of this command, and when it is activated, the choice of domain among integer, decimal, rational and irrational expressions. The *expand-and-reduce* command applies to polynomial expressions of several variables; it provides the canonical form of the polynomial. It can be deactivated. The *factor* command applies to polynomial expressions of one variable; it can be deactivated, active for polynomial expressions of degree 1, or for polynomial expressions of degree less than or equal to 2. The *solve* command applies to equations of one variable; it can be deactivated, active for equations of degree 1, or for equations of degree less than or equal to 2. In the future, we will extend the *solve* command to some inequalities and to some systems of linear equations.

To apply a command, the student has first to select a subexpression, then to choose the command in the menu. When the command is applicable, the system makes the calculations and replaces the selected expression by the result (this does not generate a new step in the calculation). See the example in Figure 13.

Note that these CAS-like commands do not make APLUSIX behave like a CAS. A CAS solves a given problem. With APLUSIX, the teacher chooses the commands that are made available to students (corresponding to actions that are well known to the students) and the

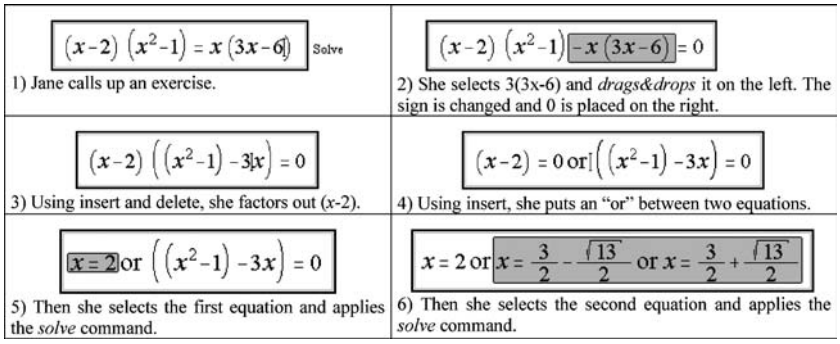

Figure 13. example of use of the equivalent *drag&drop* and of the solve command. The exercise is a third degree equation. The solve command allows solving of first and second degree equations, so it cannot directly solve the exercise.

exercises set for the students; whilst the commands can help to solve the exercises they should not be able to solve them directly.

7.2. *Direct Manipulation Preserving Equivalence*

In a microworld, the use of direct manipulation is typically to allow the movement of objects whilst respecting the relationships between them. In algebra, this takes the form of manipulating sub-expressions in a way that respects their structural relationships. This leads to the structural *drag&drop* functionality which we presented in Section 4.9. Sub-expressions can also be manipulated using equivalence relationships, which leads to having an "equivalent *drag&drop*": moving a sub-expression inside a global expression and preserving the equivalence of the global expression.

Such functionality is implemented in the software called GRAPHING CALCULATOR, allowing one to manipulate expressions algebraically with the mouse and is described as "the calculator preserves equality during manipulations".

Moving arguments of a commutative operator is the first natural form of equivalent *drag&drop* (e.g., moving $3x^4$ to the left in $2x^3 + x^2 + 3x^4$ provides $2x^3 + 3x^4 + x^2$ if the drop is before $3x^4$ and $3x^4 + 2x^3 + x^2$ if the drop is before $2x^3$). In this particular situation, the structural *drag&drop* does the same thing if the drop is made with the operator "+".

A second natural form of equivalent *drag&drop* consists of "moving" an additive sub-expression from one side of an equation (or inequality) to the other side, changing its sign (e.g., in $x^2 + 5x = -6$ moving -6 to the left produces $x^2 + 5x + 6 = 0$).

A third natural form of equivalent *drag&drop* consists of moving a multiplicative sub-expression from one side of an equation (or inequality) to the other side (e.g., in $5x = -6$ moving 5 to the right produces $x = -6/5$).

Other forms are proposed in Table I. They are based on factorisations or reductions.

The equivalent *drag&drop* functionality is not yet implemented in APLUSIX. We will first implement it for the case of moving additive and multiplicative sub-expressions. This will be useful for advanced students (see Figure 13), but there will be an option to allow the teacher to deactivate it since it is too powerful for beginners. It is not yet clear whether other sorts of equivalent *drag&drop* actions may be useful for the APLUSIX system.

TABLE I

Examples of equivalent *drag&drop* with basic operators

Expression	Selected sub-expression	Place of the drop	Result	Type of action
$3x^2 - 1 + x^2$	x^2	Over $3x^2$	$4x^2 - 1$	Reduction
$(y-1)(x+x^2)$	First occurrence of x	Between)(	$(y-1)x(1+x)$	Factorisation
$(xy)^2$	x	Before (	x^2y^2	Factorisation
$\sqrt{4+x}$	4	Before the root	$2\sqrt{1+\frac{x}{4}}$	Factorisation

8. SOME TESTS OF APLUSIX

From the end of 2001 to the end of 2003, we regularly organised tests of the system with teachers of middle schools and high schools, mainly for grades 9–11. The first goal was to test the software in order to assess its utility and its usability. The second goal was to collect data for a project aiming to model students algebraic thinking with mal-rules (Payne and Squibb, 1990) and students' algebra conceptions (Balacheff and Gaudin, 2002). This work is still continuing and will not be presented in this paper. The first goal led us to organise experiments from September 2002 to December 2003 to evaluate the usability of the system.

Apart from a few special cases, all the tests of APLUSIX were made by teachers of participating classes during regular school time. Usually, there was one student per computer in computer rooms having from 12 to 30 computers.

After these tests, the usability of the software was apparent to us, at least for the basic functions (input, delete, make new steps). Classes of 30 students were able to engage in a productive activity from the beginning with very little help of the teacher. However, most of the students did not naturally use the *cut*, *copy*, *paste* and *drag&drop* functions. The use of these functions needed to be suggested. We also noted that the difficulties faced by the students were mainly mathematical in nature, as opposed to interface or usability problems with the software.

8.1. *Regular Use of APLUSIX in a Grade 9 Class*

In December 2001, 18 students of a grade 9 (middle school) class used the system several times a week (in Montfermeil in France). The class

was a special one, with many students having deep difficulties with mathematics. The students started learning expansion, simplification and factorisation of simple expressions, and solution of simple equations with APLUSIX. Some students worked alone, others worked in groups of two. Most of them needed just a few minutes to become familiar with the software, even those who did not have significant experience with computers. Some of them acquired a good mastery of the *drag&drop* functionality.

The teacher noticed an improvement of the students' interest for algebra. All the students enjoyed going to the computer room. Some of them, who generally were unengaged during lessons, began to ask questions. They solved more exercises than usual and more difficult exercises. The teacher noticed also that his relation with the students shifted from the position of a judge (who decides what is wrong) to the position of an interpreter (who answers questions and explains errors).

8.2. *A Test with Paper-Pencil Pre-test and Post-test*

A test was organised in January 2002, with a small group of 8 volunteers coming from different classes of the middle school of Montfermeil. They worked outside their normal classes. The students had a 30 min paper and pencil pre-test followed by a 90 min session with the system. One week later, they had another 90 min session with the system followed by a 30 min paper and pencil post-test. The exercises all concerned linear equations and inequalities. Verification of calculations was activated during the sessions and deactivated during the tests. There was no classroom lesson on algebra between the two sessions.

From the pre-test to the post-test, the average of the group increased from 4.2 out of 10 to 7.9 while the standard deviation decreased from 3.4 to 2.8. Besides this progress, we noticed, for some students, an evolution in the presentation of reasoning, as shown for example in Figure 14.

During the sessions, the students had the possibility of asking questions of the teacher. Most of the questions were *Is it finished?* (at this time, the system indicator for the end of the exercise were not available) and *Why is it not working?* The latter question generally arose from difficulties with fractions and negative integers. The algebraic techniques used by the students in the post-test satisfied the expectations of the teacher.

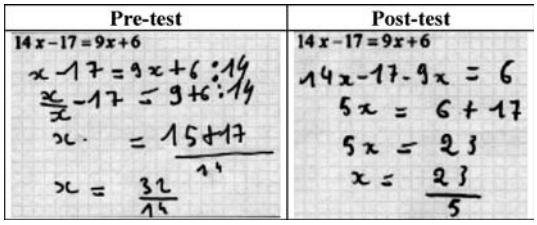

Figure 14. The evolution of the presentation of reasoning for one student.

9. A ONE-YEAR-LONG USE: APLUSIX AS A MILIEU FOR LEARNING

A long experiment of APLUSIX was conducted in 2002–2003 at grade 10, with 33 students of a high school in Annemasse in France. At the beginning of the year, before any teaching of algebraic notions, we prepared a pre-test, using APLUSIX, concerning different types of algebraic problems which had already been presented in grade 9. The analysis of the test pointed out some student difficulties concerning algebraic notions studied in grade 8. As a consequence, we organized lessons on factorisation and equation solving, and prepared activities (2 – 3 hours on each topic) with APLUSIX during the class time.

During the rest of the year, the teacher was invited to use APLUSIX every time he thought it was relevant. We observed that he used APLUSIX every time he worked on algebra, especially for inequalities and systems of equations, in preference to the paper-and-pencil environment.

We employed a constructivist approach to learning where students learn by adapting to a *milieu* that provides contradictions, difficulties and disturbing situations, and so on (Brousseau, 1997). In this approach, knowledge construction is the result of the interaction of the student with a particular environment. The environment should be organized by the teacher with adequate problems, adequate sorts of actions available to students in the environment, and adequate types of feedback provided by the environment. We viewed APLUSIX as a *milieu* for learning in which the actions concerned mainly the manipulation of algebraic expressions, and providing three categories of feedback: feedback about the equivalence of expressions; feedback on the state of the current step provided by indicators in the software; and feedback provided by textual messages.

9.1. *Solving Equations in the French class "Seconde" (Grade 10)*

From grades 8 to 10 in France, the curriculum contains equations that can be classified into the following types, where x is the unknown, a–d are numbers, and the coefficients of x are not equal to 0:

T_0: $ax+b=0$, T_1: $ax+b=cx+d$, T_2: $(ax+b)\times(cx+d)=0$.
T_3: $(ax+b)\times(cx+d)=(ex+f)\times(gx+h)$.
- T_{3a}: After expansion, terms in x^2 disappear, factorisation does not apply, expansion is necessary.
- T_{3b}: After expansion, terms in x^2 disappear, but factorisation applies too.
- T_{3c}: After expansion, terms in x^2 do not disappear, factorisation is necessary.

T_4: $A=B$, where A and B are expanded and reduced forms of degree less than or equal to 2, one form at least being of degree 2. After reduction and collection of all the terms on the left, one gets $ax^2+bx+c=0$ where a factorisation is possible with a second degree identity.
T_5: $x^2=a$, with a positive or null. T_6: $(ax+b)^2=0$.
T_7: Other types of equations that can be transformed and lead to a previously described type.

Types T_0, T_1, T_2, T_{3a} and T_{3b} are well studied in the curriculum before grade 10. Types T_{3c}, T_4 and T_5 are studied at grade 10. Given this curriculum, we chose to conduct an experiment that focused on the need for factorisation in the equation-solving process.

The methodology used in this experiment was the following. Taking into account our analysis of the problems of types T_1, T_2 and T_3 in the pre-test, we prepared a teaching sequence organized in three phases. During these phases, the students worked individually and exclusively with APLUSIX.

Phase 1: Acquisition of knowledge :

Verification of calculations by APLUSIX was on demand and the indicators were shown to the student. As the goal of the experiment was to show the use of factorisation in the solution of problems of type T_{3c}, we had three steps in this phase:

Step 1: (1 hour): Solution of linear equations and second degree equations without factorisation (problems of types T_1, T_2, T_{3a} and T_{3b}).

Step 2: (1 hour): Solution of second degree equations with factorisation (problems of type T_{3c}). After steps 1 and 2, the teacher presented a synthesis of the procedures for solving equations.

Step 3: (1 hour): Training with equations of type T_1, T_2 and T_3. For problems of type T_3, the students had to decide which procedure should be used.

Phase 2: Post-test:

A post-test was organized in order to measure the student's progress in comparison with the pre-test on equations of type T_1, T_2 and T_{3c}. During this phase, the verification of calculations was disabled and the indicators were hidden.

Phase 3: Individualized help:

The results obtained from the post-test showed a group of five students in great difficulties. So we set up individualized help for those who were volunteered (four out of five). This work was done outside of the class; the students worked at home, or in a self-service computer room. During this phase, verification of calculations was available and the indicators were shown.

9.2. *Results*

Table II shows the evolution of the scores of the students for the problems of types T_1, T_2 and T_{3c}. The amount of progress depends on the type of the problem, and also, for each type of problem, on subcategories of problems. For example, for type T_1, the success for exercises with integer coefficients was 100% in the post-test whereas it was only 30% for exercises with non-integer coefficients.

As students only worked with APLUSIX during this period, we make the hypothesis that the evolution of the results was due to the *milieu* provided by APLUSIX and to the choice of the situations.

TABLE II

Evolution of the scores between the pre-test and the post-test (percentages of well-solved exercises)

Type of problem	Pre-test%	Post-test%
T_1	46	74
T_2	3	63
T_{3c}	27	71
Total	18	68

9.3. *Different Behaviors of Students with APLUSIX*

During phase 1, verification of the calculations was available. The students had to decide to ask for the display of equivalence, and to use the resulting information. This introduces APLUSIX as an environment for doing experiments. While performing actions and observing feedback, students developed their abilities, their control over the software, their strategies, and they were able to correct their errors. An example of such behaviour is shown in Figure 4 (Section 3).

During phase 2, the students tried to expand equations of type T_{3c}. They reached an equation of the form $ax^2 + bx + c = 0$ that they were not able to solve. At this time, the teacher made an intervention to explain the factorisation strategy. Then the students applied this strategy to equations having apparent factors. After a while, some students again used the expand strategy for second degree equations. Two hypotheses can explain this behaviour: (1) the factorisation method was not yet well established; (2), the common factor was not apparent enough, e.g. $(2x - 4)(x + 1) - (x - 2)(x + 3) = 0$. After they got the $ax^2 + bx + c = 0$ form, most of these students back-tracked to a previous step and used the factorisation method.

The systems of linear equations confronted the students with the issue of equivalent systems. The teacher recalled the two methods (linear combination and substitution) and the students solved a few systems of equations using paper and pencil, working on one equation, then on the other, but not working through equivalent systems. Then, the students had a session on systems of equations with APLUSIX set to offer continuous feedback about equivalence. In the beginning, most of the students did not work with equivalence. But the sanction of the *milieu* was immediate, refusing to proceed to a new step from a non-equivalent step. The students thought first that it was a bug in the system, and after an explanation given by the teacher, they began to worked by using equivalence. Later, when they worked with paper and pencil environment, a majority of students solved the systems of equations by equivalence.

Globally, the teacher observed that the students changed their answers more easily using APLUSIX than in the paper and pencil environment, and did not hesitate to test new strategies.

9.4. *The Teacher's Point of View About the use of APLUSIX*

Regularly during the year we discussed the use of APLUSIX with the teacher. We present below the main points evoked during these discussions:

- The use of the system on the school computer network was appreciated because it permits individual and collective use. The possibility to set the options of the system according to the current learning goals was appreciated too.
- The initial learning of the system did not present any difficulty and was very quick.
- Students in difficulty showed motivation to solve exercises, on their own, outside the lesson. They chose exercises in their text book and solved them with APLUSIX. This may be explained by the fact that the students without computer software do not usually have enough self-control to validate their answer and that the system provides such validation.
- The students needed less help from the teacher when they were working with APLUSIX (this point was emphasised by the teacher). They gained autonomy in the solving of problems. This confirms the aspect of APLUSIX as an environment for exploring and experimenting.
- The teacher observed better results in comparison to those of previous years, and students seemed to become more rigorous about algebra syntax in the paper and pencil environment.

10. AN AUTOMATIC TEST IN A GRADE 10 CLASS

In November and December 2003, a teacher prepared a test for her grade 10 class in the high school of Seyssinet in France, devoted to equations and inequalities. The general organisation is given in Table III.

The main goal was to evaluate the progress of the students after the 50 min training session (phase 3) with APLUSIX and just a little help from the teacher. 28 students participated in the 5 phases. The training phase consisted of using the system with verification of calculations available on demand by the student. For evaluating the students' progress, a pre-test and a post-test were prepared with the computer in phases 2 and 4, without verification of calculations.

TABLE III

Organisation of the test in the high school of Seyssinet

Phase	Date	Duration	Activity	Verification of calculations
1	November 14	20 min	Discovery	Continuous
2	November 14	20 min	Pre-test	Off
3	December 5	50 min	Training	On demand
4	December 19	30 min	Post-test	Off
5	December 19	20 min	Post-training	Continuous

Phase 1 was devoted to "discovery" learning of the software with continuous verification of calculations. At the end, phase 5 was devoted to a free use of the software, again with continuous verification of calculations. All the actions of the students were recorded in interaction files.

The interaction files were analysed by ANAÏS, a computer program developed in our team which calculates summary statistics from the students' interaction files. Table IV shows the results of the pre-test and Table V the result of the post-test.

In the post-test (phase 4), the students engaged in many more exercises (a total of 282 exercises in 30 min against a total of 132 exercises in 20 min in phase 2) and succeeded in solving many more exercises (a total of 96 exercises compared with a total of 56 exercises). When we compare the results for similar exercises, we also see significant progress – see Table VI.

However, the ratio of solved to attempted exercises decreased from 42% to 34%, and the percentage of correct steps decreased from 72% to 66%. The explanations we propose for this phenomenon are: (1) some difficulties remain (e.g., with fractions); (2) phase 4 contains more difficult exercises, so new difficulties may have introduced new errors (e.g. complex expansion).

The exercises of the training phase (phase 3) that were attempted by the students are given in Table VII. Notice that the students in the training phase solved during 50 min many fewer exercises than in the post-test (30 min). This partly comes from the use of the system feedback during the training phase, which indicated the correctness of the calculation steps and led the students to redo incorrect calculations.

TABLE IV

Results of the 20 min pre-test. "Attempted the exercise" means that the student spent time trying to solve the exercise. "Solved the exercise" means that the student succeeded in this activity by obtaining a solved form using correct steps

No.	Exercise	No. Students who attempted the exercise	No. Students who solved the exercise	Who solved (%)	Number of steps	Correct steps	Correct steps(%)
1	$2x+4=6$	28	24	85	87	80	91
2	$5-3x=9$	27	9	33	86	58	67
3	$3-2x=3x-6$	24	9	37	70	52	74
4	$\frac{3}{2}+x=\frac{1}{3}-\frac{1}{2}x$	20	2	10	93	57	61
5	$3x+2<0$	8	4	50	17	11	64
6	$2-x>0$	6	1	16	9	2	22
7	$-4x-1>0$	6	1	16	11	5	45
8	$3x-8<2-2x$	6	2	33	18	13	72
9	$4(x-2)-2(x+1)=0$	4	3	75	15	14	93
10	$(2x+1)(x-3)+(-x+3)(2x-3)=0$	3	1	33	9	7	77
	Total	132	56	42	415	299	72

TABLE V

Results of the 30 min post-test

No.	Exercise	No. students who attempted the exercise	No. students who solved the exercise	Who solved (%)	Number of steps	Correct steps	Correct steps (%)
1	$2x+4=7+5x$	28	18	64	96	73	76
2	$\frac{3}{2}+\frac{1}{2}x=\frac{1}{3}-\frac{1}{2}x$	28	9	32	116	60	51
3	$3x+2<6x+11$	28	11	39	94	70	74
4	$-4x-1>2x$	28	12	42	84	59	70
5	$3x-8<2-2x$	27	18	66	87	75	86
6	$5(x-2)-(1-x)=0$	25	10	40	90	65	72
7	$(2x+1)(x-4)=(-x+4)(2x-3)$	25	1	4	107	55	51
8	$(x-2)(3x+1)-(x+1)(3x-1)=0$	21	2	9	72	37	51
9	$(x+1)(x-1)=(2x+2)(x-3)$	18	1	5	69	40	57
10	$3(x-1)-(x+1)<0$	13	6	46	36	28	77
11	$(x+1)(x+2)<(x+3)(x+4)$	11	2	18	40	29	72
12	$(x-3)(x+1)=-x^2+1$	8	0	0	20	9	45
13	$10x-5(x-2)=6x+4$	7	3	42	15	14	93
14	$(x+2)(x-3)=(x+2)(x-4)$	5	1	20	10	5	50
15	$(9x-5)(-6x+2)=0$	4	1	25	5	4	80
16	$8\left(\frac{1}{5}+x\right)=\frac{1}{7}\left(7x-\frac{1}{5}\right)$	3	0	0	2	0	0
17	$5x-(2-3x)<=2x+4$	3	1	33	3	3	100
	Total	282	96	34	946	626	66

TABLE VI

Comparison between the two tests for isomorphic exercises

Exercise, phase 2	Similar exercise, phase 4	Score, phase 2(%)	Score, phase 4(%)
$3-2x=3x-6$	$2x+4=7+5x$	37	64
$\frac{3}{2}+x=\frac{1}{3}-\frac{1}{2}x$	$\frac{3}{2}+\frac{1}{2}x=\frac{1}{3}-\frac{1}{2}x$	10	32
$-4x-1>0$	$-4x-1>2x$	16	42
$3x-8<2-2x$	$3x-8<2-2x$	33	66

11. CONCLUSION AND FUTURE WORK

We have presented design principles for computer systems for helping students learn algebra. The principles mix features associated with microworlds and CAS, and we have described their implementation in APLUSIX. In this conclusion, we first discuss the *domain of validity* as emphasised by Balacheff and Sutherland (1994), who defined four dimensions in mathematics for the *epistemological domain of validity* of a microworld.

The first dimension is the set of problems that the microworld allows to be presented. Concerning the calculation of equivalence between expressions, the set of expressions of the domain has been indicated in Section 5.3. Concerning the types of problem, these are *reduce*, *expand and reduce*, *factor* polynomials expressions on the one hand, and *solve* equations, inequalities and systems of equations on

TABLE VII

The 14 exercises of the training phase (phase 3)

No.	Exercise	No. students attempting the exercise
1	$7x(3x+5)=0$	28
2	$8x-4=3x+2+5x$	26
3	$(x+2)(x-3)=(x+2)(x-4)$	25
4	$(9x-5)(-6x+2)=0$	20
5	$\frac{-7}{2}x-\frac{6}{2}x+\frac{5}{7}=0$	18
6	$3x-1<2x+4$	14
7	$8x-4>11x-2$	13
8	$(x+1)(x-5)>(x-3)(x+7)$	11
9	$3x+1/2<x+1$	6
10	$(x+1)(x-2)-(x+3)(x-4)>0$	6
11	$-7x-5(3x+2)+3=-2(x+9)$	6
12	$(5x+2)(4x+3)=(2+5x)(7x-5)$	5
13	$12x^2-7x=0$	2
14	$(5x-2)(4x+3)+(5x-2)(7x-5)=0$	2

the other hand. Problem types not in this list can be used with APLUSIX, but the system is not able to indicate whether the problem is solved or not. (In this case, the student indicates *terminated* instead of *solved*, and gets no feedback).

The second dimension is the nature of the tools and the objects, which is provided by the formal structure of the microworld. The objects of APLUSIX are algebraic expressions and algebraic reasoning by equivalence. The nature of the tools is to allow construction and solution of algebraic problems.

The third dimension is the nature of the phenomenology over the formal structure. The phenomenology at the interface of APLUSIX is building algebraic reasoning by equivalence with two sorts of actions: building a new step freely with the editor, and applying a command to a sub-expression. The representation of expressions has a high level of fidelity, and the reasoning is effectively reified.

The fourth dimension is the sort of control the microworld makes available to users and the feedback that it provides. For the *building expression* activity, the student may execute any editing action. The major difference compared with editors in other software is that ill-formed expressions are highlighted, and moreover editing actions that lead to incorrect expressions are completed in order to get a well-formed expression, instead of just refusing the action. This is a form of feedback: when the student disagrees with the interpretation of the system, his/her can undo the action or modify the result. Indicators provide feedback concerning the expressions and the problem. For the *reasoning* activity, the student may produce a new step at any time and receives an indication of equivalence as the critical feedback.

The APLUSIX system has been use experimentally for two years in schools, mainly for grades 9 to 11. Results are positive in terms of usability as well as in pedagogical usefulness of the system. These results confirm our didactical analysis which has shown the adequacy of the system as a *milieu* for learning, particularly in terms of the verification of equivalence which the system can perform. APLUSIX can also be viewed as a *milieu* for validation, in the sense given by Brousseau (1997), because the student can know if his/her answer is correct or not without the intervention of the teacher. This can reduce the effect of the ''didactical contract'' where the student tries to guess the result expected by the teacher when she is asked for validation. Finally, we consider APLUSIX to be an environment for experimentation as the interaction between

the subject and the *milieu* allows for exploration and the evolution of strategies.

The APLUSIX system is currently a prototype that can be downloaded from: http://aplusix.imag.fr for testing and classroom use. It runs in French, English, Portuguese, Italian, Japanese, and can be easily translated into other languages. It will become a commercial product in 2004.

Our future research on the development of APLUSIX will have four main points.

First, we will add an automatic solver, capable of solving problems in different ways depending on the level of the learner. This solver will have a behaviour close to that of a good student of the given level. It will be used to show to the student what next steps can be executed or how a problem can be solved. Such a solver is currently under development for the middle school level.

Second, we will add a graphical module capable of displaying the graphs of functions associated with certain algebraic expressions.

Third, we will study in depth the work of students with the system by analysing the recorded interactions. This work is currently taking place in a project funded by the French Ministry of Research and involving several teams.

Fourth, we will add several "tutor modules" to the system. Each tutor will be devoted to the monitoring of the student on a particular point in order to teach the point or correct a misconception.

Besides these developments of the student software, we will develop the software for teachers, providing tools for the administration of the students' files on the server, and for calculating statistics like those presented in Section 10.

We will also continue to experiment with APLUSIX, in particular investigating its role as a tool integrated in the curriculum for classes in a range of grades. And we will further test the use of the CAS-like commands with grade 12 students.

NOTES

[1] The current ILE that we describe in this paper is called APLUSIX like its predecessor. To avoid confusion, we will name it "our first ILE" in this paper.

[2] For the computer systems mentioned without references details are given in the References at the end.

REFERENCES

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), pp. 183–197.

Anderson, J.R., Boyle, C.F., Corbett ,A.T. and Lewis, M.W. (1990). Cognitive modeling and intelligent tutoring. *Artificial Intelligence* 42 (1).

Arzarello, F., Bazzini, L. and Chiappini, G. (2001). *A Model for Analysing Algebraic Process of Thinking*, in Sutherland, et al (Eds.), *Perspectives on School Algebra*. Dordrecht: Kluwer Academic Publishers.

Balacheff, N. and Gaudin, N. (2002). Students conceptions: An introduction to a formal characterization. Cahier du laboratoire Leibniz # 65, http://www-leibniz.imag.fr/Les-Cahiers/Cahiers2002.html.

Balacheff, N. and Sutherland, R. (1994). Epistemological domain of validity of micro-worlds, the case of Logo and Cabri-géomètre. In : R. Lewis and P. Mendelshon (eds.), *Proceedings of the IFIP TC3/WG3.3*, *Lessons from learning* (pp.137–150). North-Holland.

Ball, L. (2001). Solving equations: Will a more general approach be possible with CAS? Proceedings of the 12th ICMI Study Conference. The University of Melbourne.

Beeson, M. (1990). Mathpert, a computerized learning environment for Algebra, Trigonometry and Calculus. *Journal of Artificial Intelligence in Education* pp. 65–76.

Beeson, M. (1996). Design principles of mathpert: Software to support education in algebra and calculus. In N. Kajler, (ed.), *Human Interfaces to Symbolic Computation*, Springer-Verlag.

Brousseau, G. (1997). *Theory of didactical situations in mathematics*. Kluwer Academic Publishers, Dordrecht.

Büdenbender, J., Frischauf, A., Goguadze, G., Melis, E., Libbrecht, P. and Carsten U. (2002). Using Computer Algebra Systems as Cognitive Tools. *Proceedings of ITS2002*. *LNCS 2363* (pp. 802–810). Springer.

Bundy, A. and Welham B. (1981). Using meta-level inference for selective application of multiple rewriting rule sets in algebraic manipulation. *Artificial Intelligence*, Vol 16, no. 2.

Cerulli, M. and Mariotti, M.A. (2000). A symbolic manipulator to introduce pupils to algebra theory. *Proceedings of the workshop Learning Algebra with the Computer, a Transdisciplinary Workshop*. ITS'2000. Montreal.

Dershowitz, N and Jouannaud, J.P. (1989). Rewrite Systems. In *Handbook of Theoretical Computer Science*, Vol B, Chap 15. North-Holland.

Foss, C.L. (1987). Learning from errors in ALGEBRALAND. IRL report No IRL87–0003.

Hollan, J.D., Hutchins, E.L. and Weitzman, L. (1984). STEAMER: An interactive inspectable simulation-based training system. *AI Magazine*, vol 5, n° 2.

Kieran, C. (1991). A procedural-structural perspective on algebra research. *proceedings of Psychology of Mathematics Education*. Furinghetti, F (Ed.), Assisi, Italy.

Koedinger, K.R., Anderson, J.R., Hadley, W.H. and Mark, M.A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education* 8, (pp. 30–43).

Laborde, J.M. (1989). Designing intelligent tutorial systems: The case of geometry and cabri-géomètre, *IFIP WG 3.1 Working Conference on Educational Software at the Secondary Education Level,* Reykjavik.

Lagrange J.B. and Py D. (2002). Développer un environnement d'apprentissage utilisant le calcul formel. In *Sciences et techniques éducatives*. V. 9 n° 1–2, p. 91–120. Hermès-Lavoisier, Paris.

McArthur, D. and Hotta Y. (1987). Learning problem-solving skills in algebra. *Journal of education technology systems* 15.

Nguyen-Xuan, A., Nicaud, J.F., Gélis, J.M. and Joly, F. (1993). An experiment in learning algebra with an intelligent learning environment. *Proceedings of PEG'93,* Edinburgh.

Nguyen-Xuan, A., Bastide, A. and Nicaud J.F. (1999). Learning to match algebraic rules by solving problems and by studying examples with an intelligent learning environment. *Proceedings of Artificial Intelligence in Education*, Le Mans.

Nicaud, J.F., Aubertin, C., Nguyen-Xuan, A., Sa, M. and Wach P. (1990). APLUSIX: A learning environment for student acquisition of strategic knowledge in algebra. *Proceedings of PRICAI'90*. Nagoya.

Nicaud, J.F. and Bouhineau, D. (2001). Syntax and semantics in algebra. *Proceedings of the 12th ICMI Study Conference*. The University of Melbourne.

Nicaud, J.F., Bouhineau, D. and Huguet T. (2002). The Aplusix-Editor: A new kind of software for the learning of algebra. Proceedings of ITS2002. LNCS 2363. (pp. 802–810). Springer.

Payne, S.J. and Squibb, H.R. (1990). Algebra mal-rules and cognitive accounts of errors. *Cognitive Sciences* 14.

Oliver, J. and Zukerman I. (1990). dissolve: An algebra expert for an intelligent tutoring system. *Proceeding of ARCE,* Tokyo.

Thompson, P.W. (1989). Artificial intelligence, advanced technology, and learning and teaching algebra. In S. Wagner and C. Kieran: *Research issues in the learning and Teaching of Algebra*. Lawrence Erlbaum.

Jean-François Nicaud IMAG-Leibniz, Université de Grenoble
46, rue Félix Viallet, 38000 Grenoble cedex, France
E-mail: Jean-Francois.Nicaud@imag.fr