GOOD DIAGNOSTICS = ADEQUATE STEPWISE SOLUTION INTERFACE?

Rein Prank
Tartu University
Institute of Computer Science
prank@cs.ut.ee

About ten years ago we started the computerisation of the exercises of our basic course in Mathematical Logic in Tartu University. As a result, we implemented a package of four programs designed for the following types of exercises:
1) truth-table exercises,
2) formula manipulation,
3) proofs in Propositional and Predicate Calculus,
4) Turing Machines.

A general description of the package and our first experience is given in [1] and some concrete items are discussed in [2] and [3]. Having now exploited the package during six years with the yearly enrolment of 70-100 students, we have made some corrections and additions to our first versions of the programs. Also, some problems for which we do not have good solutions have arisen.

The reason for our project was not "to make science" but to design programs that can be really used in teaching. Our main intention was to give the students a better feedback than by traditional paper-and-pencil technology. In general our programs can be characterised as stepwise problem-solving environments with diagnostic feedback and assessment features. Already at the first stages we based the diagnosis not on prover-like error-reconstruction but on understanding the mistakes at the moment when they are made by the student. This requires some efforts by the design of problem solving dialogues and student interface. Practical orientation also means that the programs do not try to demonstrate their ability to give a very detailed diagnosis. For many situations the exploitation has convinced us that the message *"The expressions are not equivalent"* is sufficient for the students to correct the mistake. Then this minimal message is didactically the best because it makes the student understand himself what caused the incorrect result.

The paper describes the diagnostics-directed features of the interface of two programs: Truth-Table Checker and Algebraic Manipulation Assistant. Both programs have many different working modes for different ways of the use (classroom exercises and homework, tests, self-checking of written homework and even the work of instructors).

Error diagnosis in traditional and computerised environments

Consider the amount of information that the teacher can use when he is diagnosing the errors of the student in different situations. In the most trivial case the teacher knows only the answer given by the student (or looks only at the answer). If the answer is wrong,

then he can make conjectures about the mistake(s) that could cause this answer. If the student has not made some single most typical mistake, the teacher can often guess several quite probable 'stories' of proceeding, and in many cases he has no reasonable assumption.

If the teacher is checking a written homework or test in Mathematics, the solutions are usually presented in a form of some "solution text". It can be a sequence of expressions separated by '=', some kind of a table with numbers, etc. For several types of exercises such "plain solutions" have well-known commonly accepted templates.
The templates often allow to reconstruct the decisions and calculations made by the student and to understand the order of operations. But in some cases this is impossible or the steps of solution presented in the script are not atomic and contain too many possible substeps.

If the student is solving the problem on the blackboard, then the teacher has more information for precise diagnosis. The teacher can look at the order of symbolwise writing of the solution and it can help understand the student's way of thinking. The teacher has the possibility to ask questions when something erroneous appears on the blackboard. In many cases the teacher does not consider it to be necessary to give a direct diagnosis but he advises (in appropriate moment) to check (to compare) some items in the solution. Sometimes he only tells that the script on the blackboard contains mistake(s). The teacher can also require that the student must comment on every step of the solution: what operation will be made next, what part of the expression will be replaced by some equivalent, what formula will be applied, etc. Such a mode of the work allows the other students to understand what namely appears on the blackboard but it also allows to diagnose the errors and misunderstandings better.

Examine now what input data use the mathematical CAL programs for error diagnosis. Most of the drill-and-practice programs do not allow to enter more than only the answer of the task. As in our first case with the teacher it is clear that using such scarce input, the real diagnosis of the reasons for a wrong answer is a very hard or even a hopeless task. It is not surprising that most of such programs have exactly two reactions for the response: *"Correct"* or *"Wrong. Correct answer is …"*. Some papers published in the AI community describe the attempts to reconstruct the (combinations of) errors using serious prover-like algorithms and big computing resources [4]. The author considers it more prospective to develop interfaces for getting the information at the moment when the errors are being made by the student. It also allows to give the feedback when the student has not yet forgotten what caused him to make a particular mistake.

The most natural way to design acceptable computerised problem-solving environments is to put the traditional solution templates on the screen of the computer. The most natural way to design good diagnostic features is to follow the traditional noncomputerised error diagnosis environments.

Formula Manipulation Assistant

The Formula Manipulation Assistant was the first of our two programs for dealing with expressions. Expression manipulation is one of the subjects where the teachers are not satisfied with the level of skills of the students. But at the same time it is not very clear for the mathematicians where namely the students go wrong. In our work we have got some picture about the errors with propositional formulas. Here the situation of the student is slightly different than, for example, with polynomials in secondary school. On the one hand, the propositional exercises are easier(!). The definitions of all the propositional operations together contain less information than the multiplication table, and the members of the logical expressions do not have coefficients. On the other hand, the students have not yet memorised these definitions and the ideas behind propositional operations are new for them when they start the exercises.

Starting the computerisation project, we had quite clear ideas about the work of Turing Machine and Proof Editor programs. We also wanted to design a program for the formula manipulation exercises (expressibility in terms of $\{\&,\neg\}$, $\{\vee,\neg\}$, $\{\supset,\neg\}$, normal forms). But we realised that without some experience we are not able to design good interface for entering the solutions. Therefore, we first implemented quite a simple environment computerising mainly the "written homework" paradigm of error diagnosis.

Working with our first version, the student entered the solution row by row and signalled with '=' about the end of the formula (with ';' if he supposed the solution to be finished). The program had special editor that did not allow to enter meaningless symbols, ignored the difference between the upper and lower case, enabled entering the propositional connectives by functional keys and had a mechanism for symbolwise copying from preceding line.
After entering the symbol ';' or '=' , the program first checked the syntax of the obtained formula and required the correction if necessary. Further the program checked the equivalence with the preceding formula, then some simple conditions of profitability of the step and reaching the goal. In case of the mistakes the program gave a corresponding error message, the counters of mistakes were increased and the student had to correct the mistake.
The first version was not able to diagnose the errors deeper when the entered formula was not equivalent to the preceding. The exploitation demonstrated that the insufficiency of the diagnosis really caused the problems with understanding of the messages. A typical example is the situation where the student has transformed the formula
$X\&Y\sim Z\vee U$ to
$X\&Y\&Z\vee\neg Y\&\neg Z\vee U$,
i. e. applied the conversion rule $A\sim B = A\&B\vee\neg A\&\neg B$ to the substring $Y\sim Z$, that is not a subformula. In such a case the program gave simply the message that the formula was not equivalent with preceding line. The students often checked the equivalence in their lecture notes and protested that the program had not accepted their correct operation. Exploiting the first version of Assistant we formulated following law:

If the student had violated a conversion rule then he is able to correct the mistake himself using his textbook or lecture notes. But many students violate the order of operations and are then not able to correct the mistake.
Moreover, it seems that there are students who establish a new order of operations in the formula each time trying to make the next step as easy as possible. We decided to overcome the difficulties with the order of operations improving the problem solving interface and providing a better feedback.

Working with the second version of Assistant every formula conversion step consists of two parts. In the first part the student fixes the subformula that will be changed and in the second part he defines the string to be put instead of this subformula. The remaining left and right sides of the formula will be copied automatically to the next line.

Our program implements two modes for the localisation of subformula:
a) TREE - the highlighted active part of the formula on the screen is moved by arrow keys up, down, left and right on the syntax analyse tree of the formula,
b) LINEAR - the ends of active part are moved by arrow keys.

Mode a) is used in many interactive computer algebra programs. It is clear that in this mode the computer itself makes the work with the order of operations and the student has no possibility to make mistakes and there is almost no need to learn. Only sometimes the subformula locator does not jump to the expected part of the formula and it can make the student think what is wrong.

Mode b) gives the student the possibility to make arbitrary mistakes. The error message will be given if the fixed substring is not a syntactically correct formula (for example, the brackets are not balanced) or if it is not a subformula (misunderstanding of the order of operations).

An important item by implementation of subformula location modes is the treatment of conjunction and disjunction as n-ary operations (the same problems arise with arithmetical operations in school algebra). The exercise program must enable to carry out the conversions in the same way as they are taught to be made on paper. In LINEAR mode the program allows the student to choose an arbitrary segment of the n-ary junction as subformula. We wanted to do the same in TREE but we did not find a natural way to implement this by some combination of keys. After some experiments we have, in fact, stopped to use TREE in our exercises.

For the second part of the step the program has three modes
1) IMMEDIATE - immediate entering of the string replacing the subformula,
2) RULE - choice of conversion rule from the menu,
3) INPUTRULE - entering of conversion rule.

IMMEDIATE is the working mode of the first version applied to subformula. First the program checks the syntax of the entered formula. If the syntax is correct, the equivalence with highlighted subformula is checked (and the corresponding error

message given if necessary). If the subformulas are equivalent, then the program checks the correctness of replacing (again with a possible message). For instance,
$\neg(\neg X\&\neg Y) = X\vee Y$ is a valid equivalence but in conjunction $\neg(\neg X\&\neg Y)\&Z$ the left operand cannot be replaced by $X\vee Y$ but only by $(X\vee Y)$. It means that in IMMEDIATE mode we also require from the some checking of the order of operations student in second part of the step and we diagnose corresponding errors.

In RULE mode the student has to choose some rule from the menu for the conversion of highlighted subformula. For example, for DNF exercises the menu looks like the following:

1. $X\sim Y = X\&Y\vee\neg X\&\neg Y$
2. $X\supset Y = \neg X\vee Y$
3. $\neg(X\&Y) = \neg X\vee\neg Y$
4. $\neg(X\vee Y) = \neg X\&\neg Y$
5. $\neg\neg X = X$
6. $X\&(Y\vee Z) = X\&Y\vee X\&Z$
7. $X\vee f = X$
8. $X\vee X\&Y = X$
9. $X = X\&Y\vee X\&\neg Y$
10. $(X) = X$
11. $X*X = X$
12. $X*Y = Y*X$
13. $X*(Y*Z) = (X*Y)*Z$

Rules 11-13 can be applied to conjunction or disjunction. Issues 1-6 in the list also present more than one rewrite rule. The program allows to modify the "activated" rule by functional keys. The F1 key applies the negation to the both sides (for example, the first rule is changed to $\neg(X\sim Y) = X\&\neg Y\vee\neg X\&Y)$. The F2 key exchanges the sides of the rule (the rules are supposed to be applied from left to right). After the selection of the rule the program checks whether the application of this rule to highlighted subformula is possible and then applies the rule. The brackets are put automatically if needed.

We have implemented the rules concerning disjunction and conjunction so that they may be applied to n-ary junctions. In case of the rules from the right column we also permit the multi-use of the rule in one step (deleting more than one member by rules 7, 8 and 11, adding more than one variable by rule 9, enabling arbitrary reordering by rule 12 and arbitrary changes of the brackets by rule 13). If the result of the multi-application of the rule is not uniquely defined, then the student is asked to enter additional information. So, in case of rule 7, the student is asked in case of each member of disjunction, whether to delete it or not. Then the program verifies that the members to be deleted are really false, and that not all the members are deleted. Using rules like 7, 8 and 11, the program allows to delete the members only in accordance with the meaning of the rule. For instance, by rule 11 we can convert
$X\&Y\vee X\&Y\vee X\vee X\&Y\vee X$ to $X\&Y\vee X$ but not to X.
The last step can be made by rule 8. It means that in RULE mode the students must clarify their operations to certain extent and this argumentation is checked.

The RULE mode helps the student perform the details of the conversion automatically and concentrate on the general algorithm of solving the task. We use this mode especially in DNF exercises. The argumentation of the steps also gives the possibility to diagnose

better the steps which do not correspond to the conversion algorithm. But those items are not yet sufficiently well implemented in our program.

The INPUTRULE mode is an intermediate one between IMMEDIATE and RULE modes. The student manipulates the formula as in RULE Mode using the rules from menu. From the very start some rules (like 10-13) are on the screen. The student can add new rules and delete them. The program accepts only the rules with equivalent left and right sides. If the entered rule is identical to some rule implemented in the program, then the program will understand it in an equal way (multi-use etc.). Otherwise the rule will be applied purely syntactically. The length of the added rules is restricted. Entering the rules allows the program to diagnose incorrectly memorised rules.

The exploitation of the program during several years has proved that the two-stage structure of the step has essentially improved the interface of Formula Manipulation Assistant and the diagnostic messages are almost always understandable to the students.

Truth-Table Checker

Truth-Table Checker is designed for the exercises of filling the table, checking of tautologicity, checking of equivalence etc. The work of the student at the task consists normally of two parts: filling the table and the answer motivation dialog. The student can switch over to the answer when he supposes that the answer can be motivated using the entered values in the table. In this paper we consider only the table-filling.

The program for truth-table exercises was designed as the last in our package. But the student meets it first. In our course the truth-table exercises are the first real mathematical tasks following the exercises on the formalization of the sentences of natural language. They do not require ingenuity because the solution consists usually of filling some part of the truth-table and straightforward verifying of conditions of some definition. The student must know the definitions of logical operations, the order of operations and the definition of the considered notion. Nevertheless, we have experienced that some participants of the test at the end of chapter are totally incompetent. We decided to computerise also the truth-table exercises in order to give the students a possibility to exercise with feedback and to mechanise the work of the instructors.

The mistakes of some students in truth-table are caused by insufficient carefulness. Some other students start the labs without studying the content of lectures and this can cause all types of mistakes. In both cases the message "*The row contains mistake(s)*" could be sufficient. But when designing our truth-table program we knew already from the experiments that there exists a more serious ground - for some students the idea that they must follow some order of operations is not sufficiently clear. And we knew that such students need a clear diagnosis of the mistakes.

The truth-table is a clear template for the visual organisation of the work on the task. But the symbols *'T'* and *'F'* in the filled table are not sufficient for diagnosing the errors

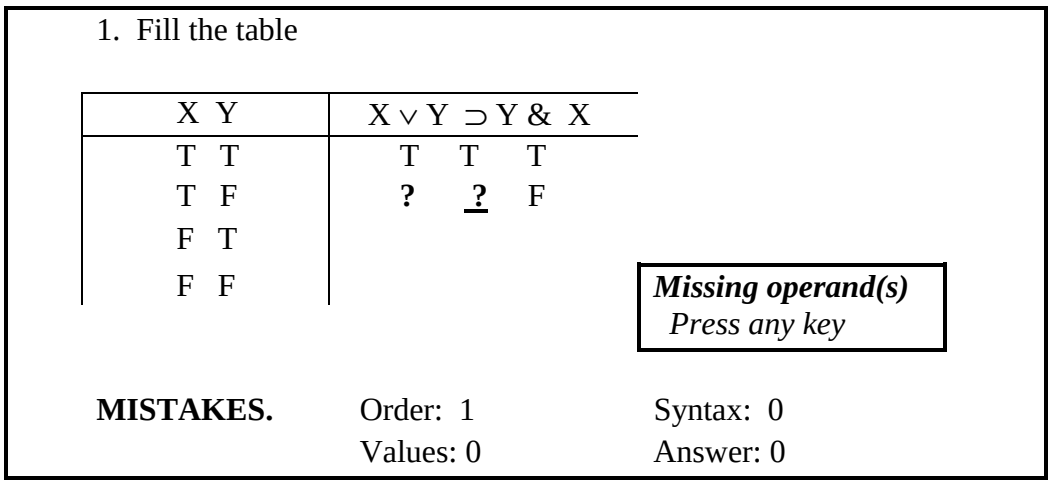
caused by misunderstanding the order. In many cases it is impossible to understand in what order the row was filled and whether the wrong value at some position is caused by a wrong order of operations or by a mistake of performing the operation. As an example of this kind we can try to diagnose the following situation:

(T,F,T) $X \supset Y \vee Z$
$F \quad T$

The comparison of different rows does not necessarily give more information because if the student does not accept the order of operations then he can fill different rows in different order. The only natural way to get the correct diagnosis is to track the student during filling the table and to react to the wrong order of operations immediately.

Now let us describe the work of the program. For filling the table of Truth-Table Checker the student has to perform all the logical operations moving the cursor to the necessary position in the table and entering *T* or *F*. The program checks the work of the student and can do (in some working modes) part of the work. The working mode depends on the values of the following options:
ORDER (CHECKED / FREE / AUTOMAT),
OPERATIONS (STUDENT / AUTOMAT),
VALUES-CHECKING (RAPID / BEFORE_ANSWER / AT_END),
LINEFILL (FULL / PARTIAL).
The student can control the direction of filling the table (*HORISONTAL / VERTICAL*).

In *VERTICAL* case the cursor is moved down in the column automatically. *PARTIAL* and *FREE* were implemented for possible self-checking of written homework but we have not used them. *AUTOMAT* has been used in the exercises on finding the formula for the given column of the values and in the work of the teachers.
Normally the order is *CHECKED*. It means that if the student tries to enter the result of some operation but the operands are not yet entered, then the program gives the message "*Missing operand(s)*", the counter of order mistakes is increased and the question marks appear in the table at the position of cursor and missing operand(s):



Our students solve the first exercises for learning the propositional connections by *VALUES-CHECKING=RAPID* signalling about the wrong truth-values immediately by the input. But the students do not like this mode and ask for the possibility to revise the table before checking the mistakes. In real work they correct the values very seldom. In *BEFORE_ANSWER* the values in the table are checked when the student chooses the item Answer in main menu. If the table contains wrong value(s), the corresponding message is given, the counter of mistakes is increased by 1 and the cursor points to some mistake. The student can correct one or more mistakes and choose Answer again. The program switches to the answer motivation dialogue only after correcting all the mistakes in the table. We have also designed a mode *VALUES-CHECKING =AT_END* where the students answer the question of the task and motivate it using the actually entered (not certainly correct) values. Note that this mode is not a sole computerisation of the paper-and-pencil procedure. If the order of operations is *CHECKED*, then we know that the operations are made in the right order and we are able to point out and count the mistakes. Main (default) mode of Truth-Table Checker is
CHECKED + STUDENT + BEFORE_ANSWER + FULL. Different combinations of the values of the task options produce the modes which are suitable for different stages of the work and for different types of the tasks.

Conclusions

1. The work with two expression-oriented programs has convinced us that the error diagnosis can be successfully improved using appropriate stepwise solution interfaces. Fortunately we must not invent new ways of conversation between the student and the computer for this purpose. They could be hard to understand for the students. The most natural way for getting useful information is to keep the usual templates of written solutions on the screen and to follow the dialogues developed during centuries by human teachers. If we do it adequately, then the student understands the solution interface and the program can give the feedback in understandable terms. It also guarantees sound integration of computerised and traditional methods.

2. Understanding of the order of operations plays some key role in the exercises with expressions. Understanding of right order is critical for the students and understanding of student's order is critical for the diagnostic procedure. The programs that are successful with the order of operations seem to be successful.

3. The exercises of our computerised labs have direct analogs in school mathematics. We suppose that the same is true of the methods of computerisation.

References

[1] R Prank. Using Computerised Exercises on Mathematical Logic. *Informatik und Schule 1991*. *Informatik-Fachberichte*, 292, 34-38. Springer-Verlag 1991.
[2] R. Prank, H. Viira. Algebraic Manipulation Assistant for Propositional Logic. *Computerised Logic Teaching Bulletin* 4, Nr.1, 13-18. University of St Andrews 1991.
[3] R.Prank. Towards Flexible Programs For Exercises In Mathematics. *Hypermedia in Tallinn'96. Proceedings.* 188-192. Tallinn Technical University 1996.
[4] H.U.Hoppe. Deductive error diagnosis and inductive error generalisation for intelligent tutoring systems. *Journal of Artificial Intelligence in Education*, 5, 27-49.