Feedback in exercise assistants

Johan Jeuring

Department of Information and Computing Sciences, Utrecht University

Technical Report UU-CS-2007-036

www.cs.uu.nl

ISSN: 0924-3275

Feedback in exercise assistants

Johan Jeuring,
Utrecht University and Open University, the Netherlands
johanj@cs.uu.nl

1 Introduction

There exist many exercise-assistants, in particular in the domain of mathematics and logic, that support students to interactively solve exercises. Examples of exercises that students solve interactively are: simplifying a fraction, determining the value of an expression, solving a system of linear equations, etc. Tools that support solving such exercises are ActiveMath, Aplusix, and Mathpert; a complete list would contain hundreds of tools. The tools vary along different dimensions (user-interface, user-input: text, or transformation-step selection, student-level: high-school, university), but, unfortunately, all of them provide very little feedback.

Giving feedback in an interactive exercise assistant is non-trivial. To give feedback, the following questions have to be addressed:

- when do you give feedback?
- about what do you give feedback?
- how do you construct feedback?

In this abstract, I will discuss how these questions are addressed in existing tools, what the literature says about these questions, and how we think the current situation can be improved.

2 When do you give feedback?

Feedback may be given immediately after an error, after a certain number of steps after an error has been made, when the student is finished, or only when the student asks for it.

The literature is not clear about when it is best to give feedback. Some research shows that giving immediate feedback is preferred, whereas other work claims that allowing a student to proceed on an erroneous path for a while, and giving delayed feedback is better.

Most interactive tools give limited feedback at intermediate steps, if they give feedback at all. The most common kind of feedback is correct/incorrect. A number of tools give more elaborate feedback at the end of an exercise. For example, if an answer to an exercise is incorrect, some tools analyse the difference between the students answer and the correct answer, and base their feedback on that difference.

3 Feedback about what?

At the end of an exercise, existing tools tell a student at least whether or not the constructed answer is correct. In case of an error, a number of tools analyse the difference between the students answer and the correct answer, and base their feedback on that difference. But most often the feedback message is correct/incorrect, possibly together with an explanation of the most frequently occurring mistakes, leaving it to the student to find out exactly what kind of mistake has been made.

Some tools give feedback on the level of rewrite steps ('you have not performed the rule that distributes multiplication over addition correctly'). These feedback messages sometimes take the expression supplied by the student into account, but often the message is generic, and only depends on the student performing an incorrect step. We have encountered no tools that provide feedback on the level of the strategy (procedure, plan, ...) for solving an exercise ('first simplify an expression, before you start on ...'). Tools that give hints about which step to take next do exist, but they do not give feedback when the student strays away from the strategy for solving an exercise.

4 How do you construct feedback?

Is the feedback the same for all exercises, do you have to specify the feedback separately for each exercise, or is the feedback derived from the exercise, and probably some more information?

In most tools, the feedback is the same for all exercises: correct or incorrect. Some tools provide some information about common mistakes made for the class of exercises to which the exercise belongs at the end of an exercise, and very few tools compare the students answer with the correct answer to calculate feedback for a particular exercise. The latter kind of tools expect the feedback to be specified with the exercise.

5 What is the cause of the lack of feedback?

The lack of feedback in exercise assistants is unfortunate. We think giving semantically rich and immediate feedback is a very important aspect of exercise assistants, which may be critical for their adaptation in teaching. There are a couple of reasons why we think current tools give rather poor feedback:

- specifying feedback per exercise usually more than doubles the size of an exercise, and the amount of work needed to add exercises together with feedback to an exercise assistant is large, probably too large for many teachers.
- automatically generating feedback for exercises, depending on the exercise, and the strategy for solving the exercise, requires knowledge about term-rewriting, strategies, error-correcting parsers, generic programming, and probably more advanced concepts from computer science. Without knowledge of these concepts, it is almost impossible to formulate the problem of automatically calculating feedback, let alone implement solutions.

In conclusion: specifying feedback is either very laborious, or it is a complicated problem that requires intimate knowledge of advanced and sometimes recent computer science concepts.

Improving feedback in exercise assistants

We have started several research projects about automatically calculating feedback when incrementally solving an exercise according to a strategy. To calculate the feedback, we need:

- knowledge about the domain of the exercise (logical expressions, sets of linear equations, matrices, ...).
- knowledge about how expressions can be manipulated (rewrite rules for the domain, such as multiplication distributes over addition), and possibly
- knowledge about common mistakes in the domain.
- knowledge about strategies for solving an exercise.

For example, consider an exercise assistant that supports solving exercises in the domain of linear algebra. A typical exercise students have to solve is to bring a matrix into echelon form, usually as part of another, more advanced, exercise. The strategy for solving this exercise is standard:

- Find the leftmost column of the matrix that has a non-zero entry.
- Exchange the topmost row that has a non-zero entry in that column with the first row in the matrix.
- Scale the first row in the matrix using scalar multiplication such that the leftmost entry is 1.
- Add the first row a scalar multiple of times to each row below it to make the entries in those rows in the column found in the first step equal to 0.
- Repeat the preceding steps for the submatrix obtained by ignoring the first row of this matrix.

This strategy should not only work for a 3x3 matrix, but it should work for a matrix of any size.

To implement a tool that supports bringing a matrix into echelon form, we have to implement the domain of matrices (probably as a list of rows), together with rules for manipulating matrices (exchange two rows, multiply a row with a scalar, add a multiple of one row to another row). Furthermore, we have to turn the description of the strategy given above into a form which the tool can understand.

We have developed a strategy language, in which we can implement the above strategy, but also other strategies for the same domain, and even for other domains, such as logical expressions. Furthermore, we have implemented an 'interpreter' for our strategy language, which takes a strategy (such as the above strategy for bringing a matrix into echelon form) and an expression from the domain (such as a particular matrix that has to be brought into echelon form), and then monitors the steps taken by a student. The interpreter gives feedback whenever a student strays away from the specified strategy. For example, if a student would start with scaling a row that is not the row with the leftmost non-zero element, the interpreter would say so. We have yet to extensively experiment with our tools, but until now the results are promising. Building an exercise assistant for a particular tool that automatically gives semantically rich feedback becomes a matter of implementing the domain, the rules on the domain, and the strategies for the exercises.

We believe our approach to building exercise assistants is very promising. We would like to emphasize that the specification of rules and strategies for a particular domain is necessary to automatically calculate feedback. This does not imply that a student can only apply one of the specified rules: we try to recognize the rule(s) applied by a student. This neither implies that every exercise has to be solved by means of one particular strategy, which would force a student

on a particular path: the strategy language is sufficiently rich to specify arbitrary strategies. Furthermore, we might allow a student to stray away from a strategy, giving feedback when a correct path is left, and resuming feedback when the interpreter recognizes the strategy again.

More information about our work can be obtained via our website:

`http://ideas.cs.uu.nl/wiki/index.php/IDEAS:Projects`

Acknowledgements: Some of this work was made possible by the support of the SURF Foundation, the higher education and research partnership organisation for Information and Communications Technology (ICT). For more information about SURF, please visit www.surf.nl. Josje Lodder commented on a previous version of this abstract. Eric Bouwers, Bastiaan Heeren, Arthur van Leeuwen, Josje Lodder, Harrie Passier, and Sylvia Stuurman contributed to the research.