Supporting Self-Explanation in a Data Normalization Tutor

Antonija MITROVIC
Intelligent Computer Tutoring Group
Computer Science Department, University of Canterbury
Private Bag 4800, Christchurch, New Zealand
tanja@cosc.canterbury.ac.nz

Abstract: Self-explanation is one of the most effective learning strategies, resulting in deep knowledge. In this paper, we discuss how self-explanation is scaffolded in NORMIT, a data normalization tutor. Data normalization is the process of refining a relational database schema in order to ensure that all relations are of high quality. We present the system first, and then discuss how it supports self-explanation. We hypothesized the self-explanation support in NORMIT would improve students' problem solving skills, and also result in better conceptual knowledge. A preliminary evaluation study of the system was performed in October 2002, the results of which show that both problem-solving performance and the understanding of the domain of students who self-explained increased. We also discuss our plans for future research.

1. Introduction

The goal of intelligent educational systems is to support students' learning, and yet evaluations show that even in the most effective systems, some students acquire shallow knowledge. Examples include situations when the student can guess the correct answer, instead of using the domain theory to derive the solution. Aleven et al. [1] illustrate situations when students guess the sizes of angles based on their appearance. On the other hand, we want students to acquire deep, robust knowledge, which they can use to solve different kinds of problems, and to develop effective meta-cognitive skills.

One of the approaches to acquiring deep knowledge is to self-explain. Psychological studies [5,6] show that self-explanation is one of the most effective learning strategies. In self-explanation, the student solves a problem (or explains a solved problem) by specifying why a particular action is needed, and how it contributes toward the solution of the problem. Self-explanation has been supported in several existing intelligent tutoring systems with extremely good results [1,2,3,7].

This paper presents the support for self-explanation in NORMIT, a data normalization tutor. Data normalization is the process of refining a relational database schema in order to ensure that all relations are of high quality [8]. Normalization is usually taught in introductory database courses in a series of lectures that define all the necessary concepts, and later practised on paper by looking at specific databases and applying the definitions.

Section 2 discusses the support for self-explanation in several existing tutors. Section 3 overviews the learning task supported by NORMIT, while the architecture of the system is given in Section 4. Support for self-explanation is discussed in Section 5. The results of a preliminary study of NORMIT are presented in Section 6. Finally, the conclusions and avenues for future research are given in the final section.

2. Related Work

Metacognition includes processes involved with awareness of, reasoning and reflecting about, and controlling one's cognitive skills and processes. Metacognitive skills can be taught [4], and result in improved problem solving and better learning [1,7]. Of all metacognitive skills, self-explanation has attracted most interest within the ITS community. By explaining to themselves, students integrate new knowledge with existing knowledge. Furthermore, psychological studies show that self-explanation helps students to correct their misconceptions [6]. Although many students do not spontaneously self-explain, most will do so when prompted [5] and can learn to do it effectively [4].

SE-Coach [7] is a physics tutor that supports students while they study solved examples. The authors claim that self-explanation is better supported this way, than asking for explanation while solving problems, as the latter may put too big a burden on the student. In this system, students are prompted to explain a given solution for a problem. Different parts of the solution are covered with boxes, which disappear when the mouse is positioned over them. This masking mechanism allows the system to track how much time the student spends on each part of the solution. The system controls the process by modelling the self-explanation skills using a Bayesian network. If there is evidence that the student has not self-explained a particular part of the example, the system will require the student to specify why a certain step is correct and why it is useful for solving the current problem. Empirical studies performed show that this structured support is beneficial in early learning stages.

On the other hand, Aleven and Koedinger [1] explore how students explain their own solutions. In the PACT Geometry tutor, as students solve problems, they specify the reason for each action taken, by selecting a relevant theorem or a definition from a glossary. The performed evaluation study shows that such explanations improve students problem-solving and self-explanation skills and also result in transferable knowledge. In Geometry Explanation Tutor [2], students explain in natural language, and the system evaluates their explanations and provides feedback. The system contains a hierarchy of 149 explanation categories [3], which is a library of common explanations, including incorrect/incomplete ones. The system matches the student's explanation to those in the library, and generates feedback which helps the student to improve his/her explanation.

In a recent project [13], we looked at the effect of self-explanation in KERMIT, a database design tutor [12]. In contrast to the previous two systems, KERMIT teaches an open-ended task. In geometry and physics, domain knowledge is clearly defined, and it is possible to offer a glossary of terms and definitions to the student. Conceptual database design is a very different domain. As in other design tasks, there is no algorithm to use to derive the final solution. In KERMIT, we ask the student to self-explain only in the case their solution is erroneous. The system decides on which errors to initiate a self-explanation dialogue, and asks a series of question until the student gives the correct answer. The student may interrupt the dialogue at any time, and correct the solution. We have performed an experiment recently, the results of which show that students who self-explain acquire more conceptual knowledge than their peers.

3. Learning Data Normalization in NORMIT

NORMIT is a problem-solving environment, which complements traditional classroom instruction. The emphasis is therefore on problem solving, not on providing information. However, the system does provide help about the basic domain concepts, when there is evidence that the student does not understand them, or has difficulties applying knowledge.

The system also insists on using the appropriate domain vocabulary; “talking science” has been shown to increase learning and deep understanding of the domain.

The student needs to log on to NORMIT first, and the first-time user gets a brief description of the system and data normalization in general. After logging in, the student needs to select the problem to work on. NORMIT lists all the pre-defined problems, so that the student may select one that looks interesting. In addition, the student may enter his/her own problem to work on.

Data normalization is a procedural task: the student goes through a number of steps to analyze the quality of a database. NORMIT requires the student to complete the following steps while solving a problem [9]:

1. *Determine candidate keys*: the student needs to analyze the given table and functional dependencies in order to determine all candidate keys. A candidate key is an attribute or a set of attributes that has two properties: uniqueness (its value is unique for every tuple in the relation) and irreducibility (no attribute can be removed from the key so that each

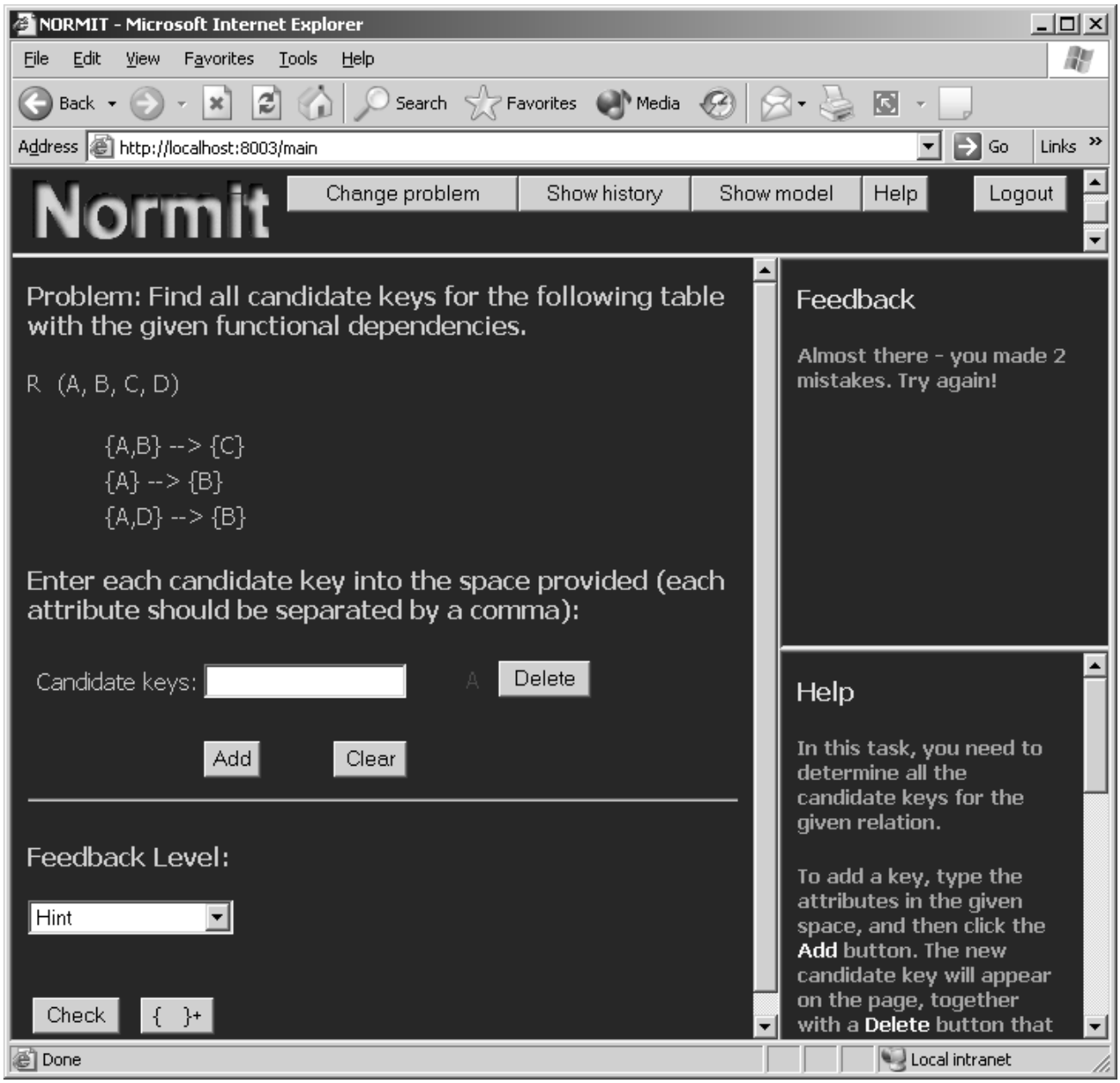

Fig. 1. A screenshot from NORMIT

value of the key is still unique). Figure 1 illustrates this task: the student is currently working on a table consisting of 4 attributes (A, B, C and D), for which three functional dependencies are given. The student enters the candidate keys one at a time, and may ask the system to evaluate the solution at any time. In Figure 1, the student has already specified one candidate key (A).

2. *Determine the closure of a set of attributes*: if the student is unsure whether a set of attributes makes a candidate key or not, he/she may compute the closure of that set under the given set of functional dependencies. This step is optional, and the student will only see that page corresponding to this task if he/she requests it (using the button labelled $\{\ \}^+$ on the candidate key page shown in Figure 1).
3. *Determine prime attributes.* A prime attribute is an attribute that belongs to any candidate key. The page for this task is shown once the student successfully determines all candidate keys for the given problem.
4. *Simplify functional dependencies*: if any of the given functional dependencies has more than one attribute on the right-hand side, the student needs to turn it into as many

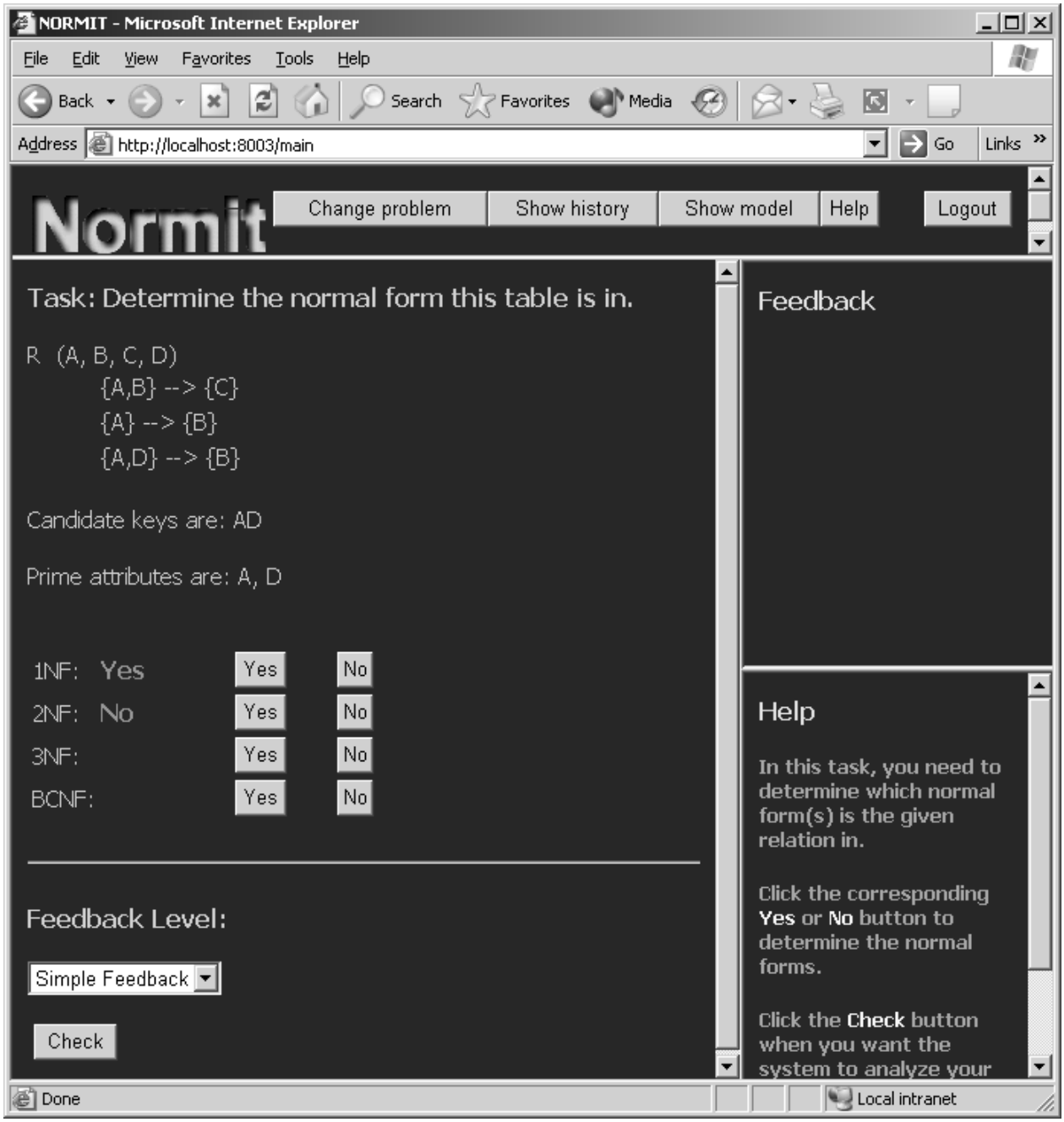

Fig. 2. The page for specifying normal forms

dependencies as there are attributes on its right-hand side (this step is the application of the decomposition rule).

5. *Determine the normal form* the table is in. The student needs to specify whether the given relation is in the first (1NF), second (2NF), third normal form (3NF), or in Boyce-Codd normal form (BCNF), as shown in Figure 2. If the student believes the relation does not satisfy a normal form, the system requires that the student specifies the functional dependencies that violate that normal form.
6. If necessary, *decompose the table* so that all the final tables are in Boyce-Codd normal form.

The sequence of the steps is fixed: the student will only see a Web page corresponding to the current task. When the student submits the solution to the current step, the system analyses it and offers feedback. The first submission receives only a general feedback, specifying whether the solution is correct or not, as shown in Figure 1. If there are errors in the solution, the incorrect parts of the solution are shown in red. Candidate key A is shown in red in Figure 1, as it is not a key. On the second submission, NORMIT provides a general description of the error, specifying what general domain principles have been violated. On the next submission, the system provides a more detailed message, by providing a hint as to how the student should change the solution. For the situation shown in Figure 1, the message on this level is: *"Almost there - still a few errors though. A candidate key you specified does not determine all other attributes. The closure of the candidate key must contain all other attributes of the relation. Try again!"* The correct solution is only available on student's request.

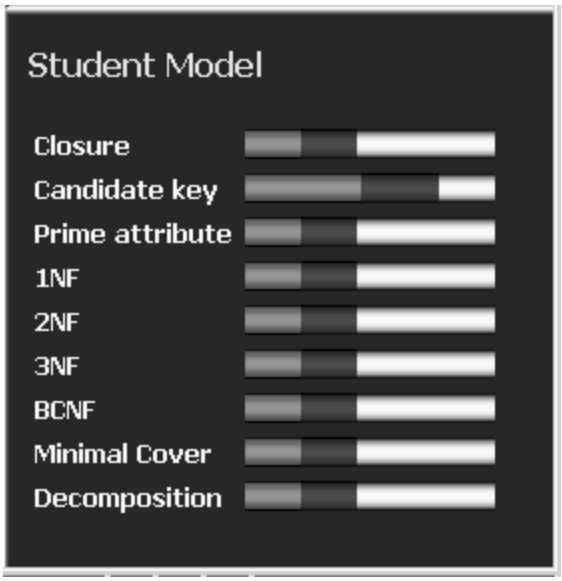

Fig. 3. The visualization of the student model

The student is not required to complete each problem; he/she may ask for a new problem at any time during problem solving. In addition to that, he/she may review the history of the current session, or examine the global view of the student model (Figure 3). The progress bars in the student model show how much of the domain the student has not covered yet (shown in white), how much of the covered part of the domain the student knows correctly (shown in green) or incorrectly (red).

4. The Architecture of NORMIT

NORMIT is a Web-enabled tutor with a centralized architecture (Figure 4). All tutoring functions are performed on the server side, where student models are also kept. NORMIT is developed in AllegroServe Web server, an extensible server provided with Allegro Common Lisp. At the beginning of interaction, a student is required to enter his/her name, which is necessary in order to establish a session. The session manager requires the student modeller to retrieve the model for the student, if there is one, or to create a new model for a new student. NORMIT identifies students by their login name, which is embedded in a hidden tag of HTML forms. Each action a student performs is sent to the session manager, as it has to link it to the appropriate session and store it in the student's log. Then, the

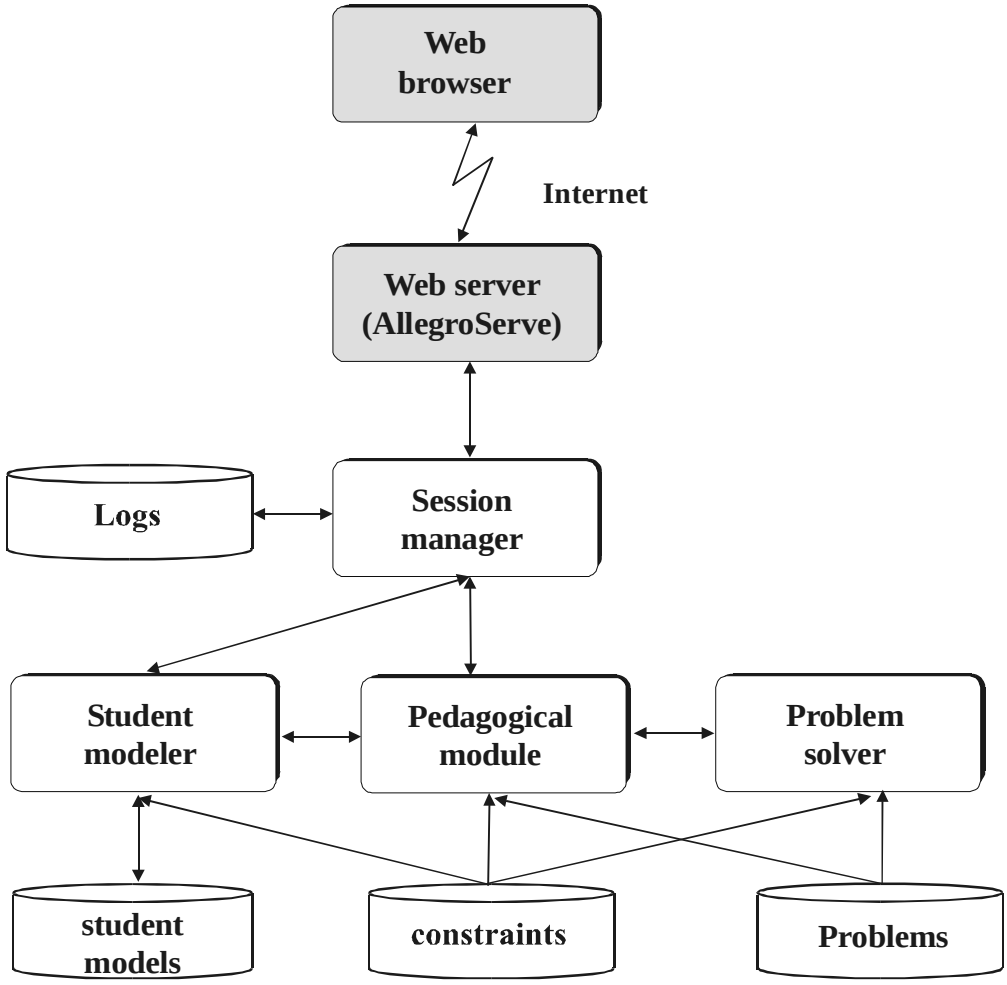

Fig. 4. The architecture of NORMIT

action is sent to the pedagogical module (PM). If the submitted action is a solution to the current step, PM sends it to the student modeller, which diagnoses the solution, updates the student model, and sends the result of the diagnosis back to PM, which generates feedback.

NORMIT belongs to the family of constraint-based tutors developed at the Intelligent Computer Tutoring Group, the University of Canterbury. All the tutors developed at ICTG use Constraint-Based Modeling (CBM) [10,11] to model the domain knowledge and the knowledge of their students. Therefore, NORMIT's domain knowledge consists of a set of constraints. CBM is a student modeling approach that is not interested in the exact sequence of states in the problem space the student has traversed, but in what state he/she is in currently. As long as the student never reaches a state that is known to be wrong, they are free to perform whatever actions they please. The domain model is a collection of state descriptions of the form: *If <relevance condition> is true, then <satisfaction condition> had better also be true, otherwise something has gone wrong.*

The constraints are written in Lisp, and can contain built-in functions as well as domain-specific functions. An example constraint is given in Figure 5. The first two lists of

```
(11 (and (equalp (current-task sol) 'candkeys)(not (null (candkeys sol)))
         (bind-all ?k (candkeys sol) bindings))
    (minimal-keyp TS (quote ?k) (problem sol))
    "You have specified candidate key(s) incorrectly!"
    "A candidate key you specified is not minimal. You need to remove the extra attributes."
    (?k "candkeys"))
```

Fig. 5. An example constraint

constraint 11 are its relevance and satisfaction conditions. The relevance condition tests whether the current task is the candidate keys task, and then it checks whether the student has specified any candidate keys. Finally, it binds variable *k* to each specified candidate key, thus forming a multiple binding list. The satisfaction part consists of a single test, which is applied to each binding of variable *k*. If a candidate key is minimal, the constraint is satisfied. In the opposite case, the student will be given feedback. There are two feedback messages in the constraint, which are given to the student if his/her solution is incorrect. The first message is shorter, and tells the student what is wrong with the solution. If the student still cannot correct the solution after this message, NORMIT will present the second message, which explains why the specified set of attributes is not a candidate key. The last element of the constraint specifies the part of the solution that is incorrect (in this case, that is the attribute to which variable *k* is bound). This binding is used for highlighting the error.

NORMIT currently contains 54 problem-independent constraints that describe the basic principles of the domain. Some constraints check the syntax of the solution, while others check the semantics, by comparing the student's solution to the ideal solution, generated by the problem solver. In order to identify constraints, we studied material in textbooks, such as [8], and also used our own experience in teaching database normalization.

The short-term student model consists of a list of violated and a list of satisfied constraints for the current attempt. The long-term model records the history of usage for each constraint. This information is used to select problems of appropriate complexity for the student, and generate feedback.

5. Supporting Self-Explanation

NORMIT is a problem-solving environment, and therefore we ask students to self-explain while they solve problems. In contrast to other ITSs that support self-explanation discussed in Section 2, we do not expect students to self-explain every problem-solving step. Instead, NORMIT will require an explanation for each action that is performed for the first time. For the subsequent actions of the same type, explanation is required only if the action is performed incorrectly. We believe that this strategy will reduce the burden on the more able students (by not asking them to provide the same explanation every time an action is performed correctly), and also that the system would provide enough situations for students to develop and improve their self-explanation skills.

Similar to the PACT Geometry Tutor and SE-Coach, NORMIT supports self-explanation by prompting the student to explain by selecting one of the offered options. In Figure 1, the student has specified the first candidate key (consisting of attribute A) for the given problem. The student would be asked to explain why the specified attribute makes a candidate key, if that is the first time he/she is specifying candidate keys, or if the student's solution is not correct. Figure 6 illustrates the next page the student will see in that situation. The student selects an incorrect option (the closure of attribute A does not contain all attributes, and therefore A is not a candidate key). The system will then ask for another explanation. In contrast to the first question, which was problem-specific, the second question is general. The student will be asked to complete the definition of a candidate key, again by selecting one of the options given, as shown in Figure 7. If the student selects the correct option, he/she will resume with problem solving. In Figure 7, the student selected an incorrect option (a candidate key is a minimal set of attributes that determine all other attributes in the relation). In such situations, NORMIT will provide the correct definition of the concept.

In summary, NORMIT asks the student to explain every action performed for the first time, or an incorrect action. The first question asked is related to the problem, and requires

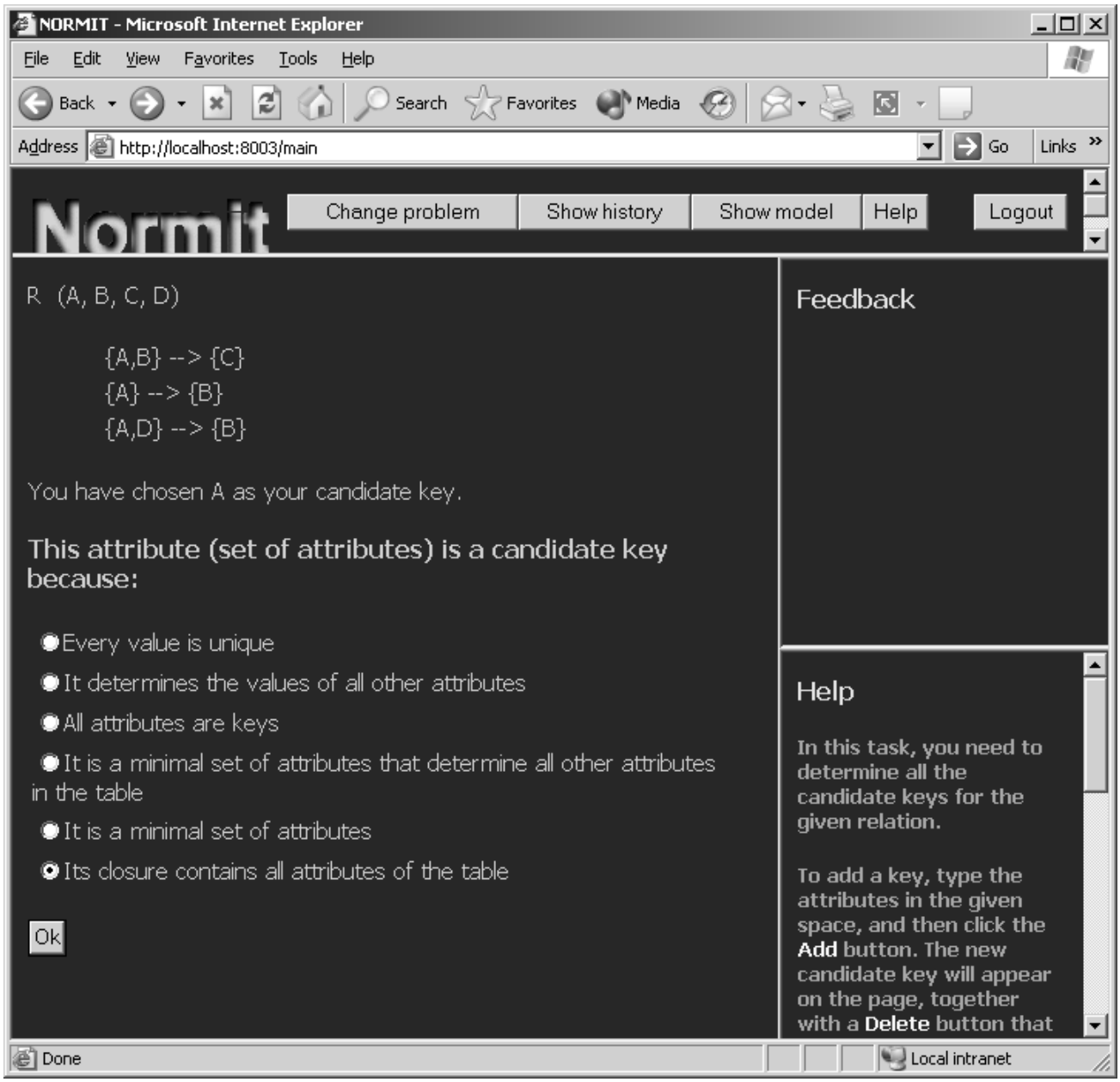

Fig. 6. Prompting the student to explain

the student to specify why he/she believes the solution is correct. If the answer is incorrect, the system will ask the student to specify the definition of the underlying domain concept. In order to answer this question, the student must relate the current problem-solving step to the declarative domain knowledge, and therefore reflect on this knowledge.

In addition to the model of the student's knowledge, NORMIT also stores information about the student's self-explanation skills. For each constraint, the student model contains information about the student's explanations related to that constraint. The student model also stores the history of student's explanation of each domain concept.

6. Experiment

We performed an evaluation study with the students enrolled in an introductory database course at the University of Canterbury in the second half of 2002. Our hypothesis was that self-explanation would have positive effects on both procedural knowledge (i.e. problem

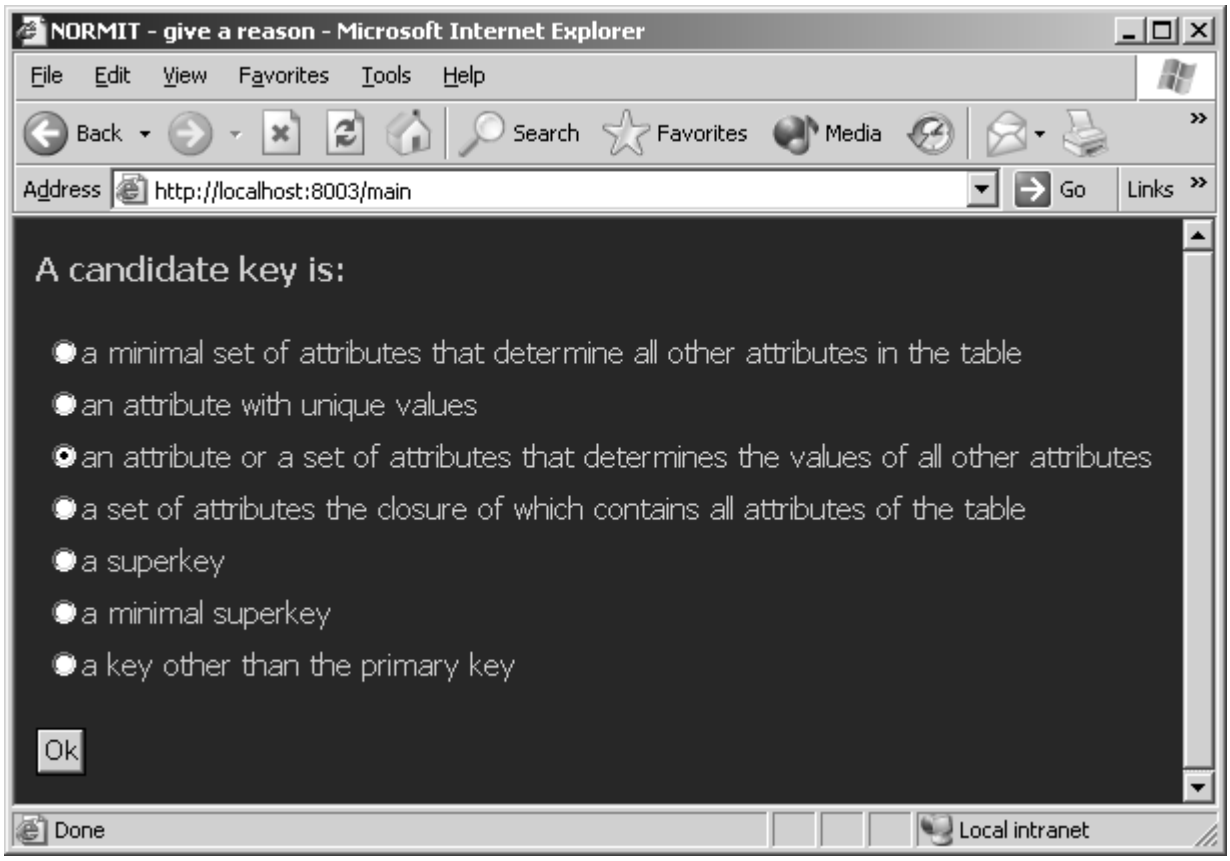

Fig. 7. Asking the student to define a domain concept

solving skills) and conceptual knowledge. Prior to the experiment, all students listened to four lectures on data normalization. The system was demonstrated in a lecture on October 14, 2002 (during the last week of the course), and was open to the students a day later. The accounts for students were generated before the study, and randomly allocated to one of the two versions of the system. The students in the control group used the version of the system without self-explanation (referred to as NORMIT in the rest of the paper), while the experimental group used the version of the system that supports self-explanation (NORMIT-SE). The participation in the experiment was voluntary, and 29 out of 151 students enrolled in the course used the system. The students were free to use the system when and for how long they wanted. There were 10 students in the control group, and 19 in the experimental group. The sizes of the groups are different, as not all students who showed interest in participating have actually used the system.

When a student logged on to the system for the first time, he/she was presented with a pre-test. The post-test was also administered on-line, the first time a student logged on to the system on or after November 1, 2002. The date for the post-test was chosen to be just one day before the exam. We developed two tests, which consisted of four multichoice questions each. The first two questions required students to identify the correct solution for a given problem, while for the other two the students needed to identify the correct definition of a given domain concept. Each student got one of these two tests randomly as the pre-test, and the other one as the post-test.

We collected data about each session, including the type and timing of each action performed by the student, as well as the feedback obtained from NORMIT. There were three students who logged on to the system, but have not attempted any problems. We excluded the logs of these three students from analyses.

The summary of results is given in Table 1. The number of sessions ranged from 1 to 10 (the average being 3.27), while session length varied from just a couple of minutes to

almost three hours. Three students attempted some problems, but completed none of them. The remaining 23 students solved at least one problem, while one student solved all 50 problems the system contains correctly. The control group students had more sessions on average, and therefore spent more time, attempted and completed more problems than the students in the experimental group (all differences except the last one are insignificant). The experimental group needed more time per problem, which may be the consequence of more work (i.e. specifying reasons) they needed to do when they made mistakes.

Table 1. Mean system interaction details

	NORMIT	NORMIT-SE
No of students	8	18
No of sessions	3.62 (2.97)	3.11 (1.78)
Time spent on problem solving (min.)	164.5 (119.97)	126.33 (99.41)
No. of attempted problems	19.37 (15.38)	11.33 (9.31)
No. of completed problems	18.5 (16.11)	7.05 (5.95)

The results on the pre- and post-tests are given in Table 2. The groups are comparable, as there is no significant difference on the pre-test performance. Only three students from the control group sat the post-test, and we have not analysed their results, as the sample was too small. On the other hand, a paired t-test for the students in the experimental group who sat both tests shows that their performance improved significantly ($p=0.08$). Therefore, the first part of our hypothesis is confirmed by the experiment.

Table 2. Pre- and post-test results

	No of pre-tests	Pre-test % (sd)	No of post-tests	Post-test % (sd)
NORMIT	8	65.62 (36.3)	3	79.17 (25)
NORMIT-SE	18	75 (25.88)	13	89.1 (17.8)

To test the second part of our hypothesis, we analysed their responses to the last two questions in the tests, which were related to students' conceptual knowledge. Again, we analysed only the results for the experimental group, as the number of post-tests for the control group was too small. The mean for the conceptual questions in the pre-test was 73.68%, and it increased to 84.61% on the post-test (significant at $p=0.13$). We used linear regression, with pre-test and the interaction time to predict the scores on the conceptual questions in the post-test (significant at $p=0.15$). Even better results are achieved when students' performance on the conceptual questions is predicted by the pre-test and the number of solved problems (significant at $p=0.11$). These results seem to support the hypothesis. However, the sample is not large enough to make solid conclusions. Furthermore, there were not enough students who sat the post-test in the control group to allow us to compare the performances of the experimental and control groups.

We also analysed student's explanations. Due to imperfection of the logging mechanism, we do not have all information about self-explanations that were problem-specific (those problems have been fixed meanwhile). From the data we have in the logs, it can be seen that some constraints are much more difficult for students to learn than others. For example, out of the total of 29 situations when students who were asked to explain why a set of attributes is a candidate key, the correct answer was given in only two cases (constraint 11 in Figure 5).

However, we do have data about students' self-explanations related to domain concepts. NORMIT tracks students' knowledge of eleven basic domain concepts (closures, candidate keys, prime attributes, simple functional dependencies, 1NF, 2NF, 3NF, BCNF, partial functional dependencies, and functional dependencies that violate 3NF and BCNF). Seven out of these concepts have been used by all students. The remaining 4 concepts have been covered only by some students, because these concepts do not appear in every problem, and the problems students attempted vary significantly. Figure 7 illustrates the correctness of students' explanations. The X axis represents the occasion number (first, second and so on) when the student was asked to define a concept. The Y axis shows the probability of a correct answer. Please note that the students were asked to explain domain concepts only when their problem-specific explanations were incorrect (the total of 147 cases). The probabilities of correct answers on the first and subsequent occasions were averaged over all concepts and all students. There is a very good fit to the power curve, which indicates that students do learn by explaining domain concepts.

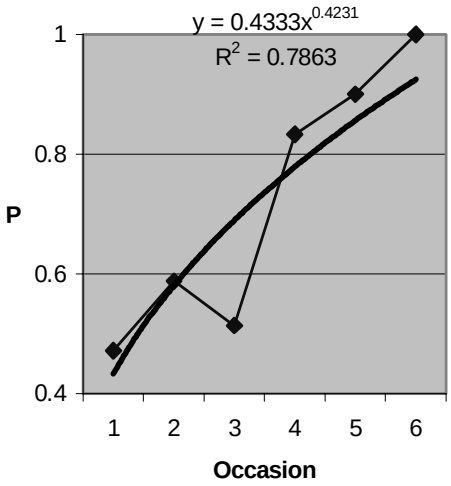

Fig. 8. Defining domain concepts

7. Conclusions

Self-explanation is known to be an effective learning strategy. Since intelligent tutoring systems aim to support good learning practices, it is not surprising that researchers have started providing support for self-explanation. In this paper, we present NORMIT, a data normalization tutor, and describe how it supports self-explanation. NORMIT is a problem-solving environment, and students are asked to explain their actions while solving problems. However, we do not require the student to explain every action, as that would put too much of a burden on the student and potentially reduce motivation. NORMIT requires explanations in cases of erroneous solutions and for actions that are performed for the first time. The student is asked to specify the reason for the action, and, if the reason is incorrect, to define the domain concept that is related to the current task. If the student is not able to identify the correct definition from a menu, the system provides the definition of the concept.

NORMIT was used in a real course for the first time in 2002. The results of the study seem to support our hypothesis: students who self-explained improved significantly in problem-solving and in answering questions about domain knowledge. However, due to the small number of participants, it is not possible to compare the performances of the students who did not self-explain to the performance of their peers who self-explained. We are

therefore going to perform a bigger evaluation study, in order to be able to assess the effects of the self-explanation support properly.

At the moment, the student model in NORMIT contains a lot of information about the student's self-explanation skills that is not used. We plan to use this information to identify parts of the domain in which the student needs more instruction. Finally, the self-explanation support itself may be made adaptive, so that different support would be offered to students who are poor self-explainer in contrast to students who are good at it. These are the future research avenues we plan to explore.

Acknowledgements

We thank Li Chen and Melinda Marshall for implementing NORMIT's interface.

References

1. Aleven, V., Koedinger, K. R., Cross, K. (1999) Tutoring Answer Explanation Fosters Learning with Understanding. In: Lajoie, S.P. and Vivet, M.(eds), Proc. AIED 1999, IOS Press, 199-206.
2. Aleven, V., Popescu, O., Koedinger, K. R. (2001) Towards Tutorial Dialogue to Support Self-Explanation: Adding Natural Language Understanding to a Cognitive Tutor. *IJAIED*, 12, 246-255.
3. Aleven, V., Popsecu, O., Koedinger, K. (2002) Pilot-Testing a Tutorial Dialogue System that Supports Self-Explanation. In: S. Cerri, G. Gouarderes and F. Paraguacu (eds.) Proc. ITS 2002, Springer, LNCS 2363, 344-354.
4. Bielaczyc, K., Pirolli, P., Brown, A.L. (1993) Training in Self-Explanation and Self-Regulation Strategies: Investigating the Effects of Knowledge Acquisition Activities on Problem-solving. *Cognition and Instruction*, 13(2), 221-252.
5. Chi, M. T. H. (2000) Self-explaining Expository Texts: The dual processes of generating inferences and repairing mental models. *Advances in Instructional Psychology*, 161-238.
6. Chi, M.T. (1994) Eliciting self-explanations improves understanding. *Cognitive Science*, 18.
7. Conati, C., VanLehn, K. (2000) Toward Computer-Based Support of Meta-Cognitive Skills: a Computational Framework to Coach Self-Explanation. *Int. J. AI in Education*, 11 389-415.
8. R. Elmasri, S.B. Navathe, (2001) *Fundamentals of database systems*. Benjamin/Cummings, Redwood.
9. Mitrovic, A. (2002) NORMIT, a Web-enabled tutor for database normalization. Kinshuk, R. Lewis, K. Akahori, R. Kemp, T. Okamoto, L. Henderson, C-H Lee (eds) Proc. ICCE 2002, 1276-1280.
10. Mitrovic, A., Ohlsson, S. (1999) Evaluation of a constraint-based tutor for a database language, *Int. J. Artificial Intelligence in Education*, 10(3-4), 238-256.
11. Ohlsson, S. (1994) Constraint-based Student Modeling. In *Student Modeling: the Key to Individualized Knowledge--based Instruction*. Berlin: Springer-Verlag, 167-189.
12. Suraweera, P. and Mitrovic, A., KERMIT: a Constraint-based Tutor for Database Modeling. In: S. Cerri, G. Gouarderes and F. Paraguacu (eds.) *Proc. ITS 2002*, Biarritz, France, LCNS 2363, 2002: 377-387.
13. Weerasinghe, A., Mitrovic, A. (2002) Enhancing learning through self-explanation. Kinshuk, R. Lewis, K. Akahori, R. Kemp, T. Okamoto, L. Henderson, C-H Lee (eds.) Proc. ICCE 2002, 244-248.