7-2007

Predicting Students' Performance with SimStudent: Learning Cognitive Skills from Observation

Noboru Matsuda
Carnegie Mellon University

William W. Cohen
Carnegie Mellon University

Jonathan Sewall
Carnegie Mellon University

Gustavo Lacerda
Carnegie Mellon University

Kenneth R. Koedinger
Carnegie Mellon University

Predicting Students' Performance with SimStudent: Learning Cognitive Skills from Observation[1]

Noboru Matsuda[1], William W. Cohen[2], Jonathan Sewall[1], Gustavo Lacerda[2], and Kenneth R. Koedinger[1]

[1]*Human-Computer Interaction Institute*
[2]*Machine Learning Department*
Carnegie Mellon University

Abstract. SimStudent is a machine-learning agent that learns cognitive skills by demonstration. SimStudent was originally built as a building block for Cognitive Tutor Authoring Tools to help an author build a cognitive model without significant programming. In this paper, we evaluate a second use of SimStudent, *viz.,* student modeling for Intelligent Tutoring Systems. The basic idea is to have SimStudent observe human students solving problems. It then creates a cognitive model that can replicate the students' performance. If the model is accurate, it would predict the human students' performance on novel problems. An evaluation study showed that when trained on 15 problems, SimStudent accurately predicted the human students' correct behavior on the novel problems more than 80% of the time. However, the current implementation of SimStudent does not accurately predict when the human students make errors.

Keywords. Cognitive modeling, programming by demonstration, inductive logic programming, Cognitive Tutor, intelligent authoring.

1. Introduction

SimStudent [1] was originally built as an intelligent building block for the Cognitive Tutor Authoring Tools (CTAT) [2]. In this authoring environment, an author can build a cognitive model that represents domain principles for a target task without significant AI-programming. Instead, the author is asked only to *demonstrate* the task, both correctly and incorrectly, using a tutor interface built with CTAT. SimStudent then learns how to perform the target task (or how to make a mistake) by generalizing the author's demonstration. The fundamental technology used in SimStudent is *programming by demonstration* [3].

Since SimStudent can generalize actions taken in the tutor interface, theoretically speaking, SimStudent should be able to model the domain principles that a human student acquired while solving problems with the Tutor. In other words, SimStudent can be used to dynamically generate a *student model* for an individual student using Cognitive Tutor.

The ultimate goal of the current project is to evaluate whether SimStudent can be used for student modeling. A preliminary study showed that SimStudent can model human students' *correct* behaviors [4]. On the other hand, modeling students' erroneous behaviors is quite challenging, partly because the human students make non-systematic errors such as slips and random errors. Although building a descriptive model of student misconceptions (telling why a particular behavior was observed) might be challenging, building a *predictive model* of student performance (telling whether a student would perform a next step correctly or not) might be tractable. Such a predictive model might give us further insight into designing a descriptive model of the student's knowledge, which represents the student's misconceptions. As an initial step towards our ultimate goal, the current paper addresses two research questions: (1) Can SimStudent learn domain principles by observing an individual student's performance? (2) Can

[1] The research presented in this paper is supported by the National Science Foundation Award No. REC-0537198. This work was also supported in part by the Pittsburgh Science of Learning Center, which is funded by the National Science Foundation Award No. SBE-0354420

SimStudent *predict* an individual student's behavior on novel problems with the domain principles learned from the individual student?

2. Related Studies

There have been a wide variety of machine-learning techniques studied so far for student modeling, including synthetic, analytic, and stochastic methods. Ikeda et al [5] built an inductive logic learner in conjunction with a truth maintenance system to inductively construct a hypothetical student model. Baffes et al [6] applied a machine learning technique to modify a given set of domain principles to be consistent with the student's incorrect behaviors. Similarly, Langley et al [7] applied a theory-refinement technique to model an individual student's behavior given a general model of the domain. MacLaren et al [8] applied a reinforcement learning method based on the ACT-R model [9] to model the process in which the students' knowledge activation patterns are strengthened. There are more studies on applications of machine-learning for student modeling.

The current study is about an application of a cutting-edge technology for student modeling. The proposed system is unique in two ways: (1) the fundamental framework used in SimStudent, inductive logic programming [10], is domain-general, and hence it is applicable to a wide range of domains; (2) SimStudent has a dual role in this context – to automatically model a domain principle used in a Cognitive Tutor, and to model a student's performance while he/she is using the Cognitive Tutor. The former cognitive model enables the Cognitive Tutor to perform model-tracing (see section 3.2), whereas the latter model allows the Cognitive Tutor to diagnose a student's competency to, say, proactively select a problem on which the student is likely to make a mistake, which in turn triggers learning.

3. Experimental Setting: Algebra I Cognitive Tutor

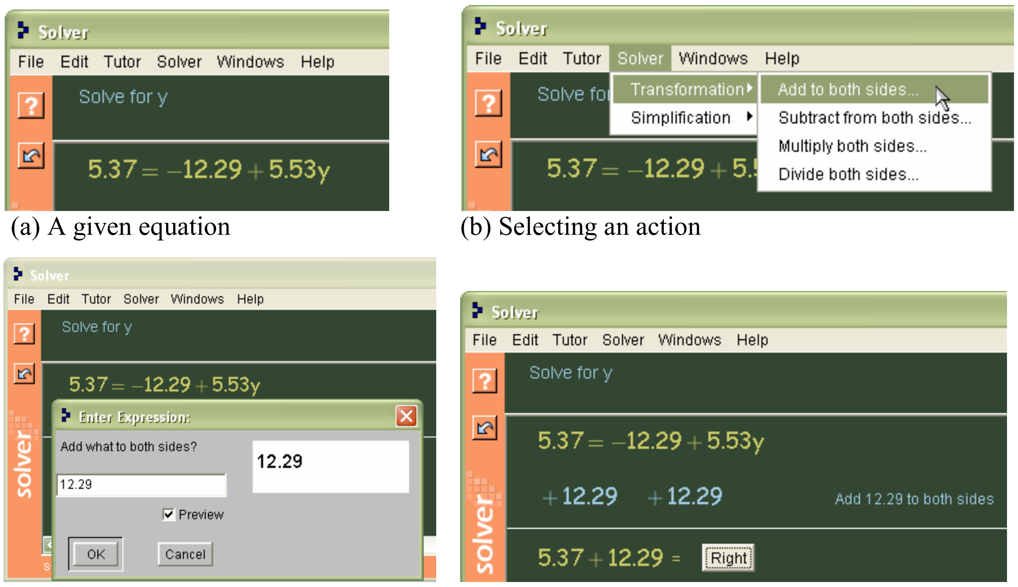
For the current experiment, we used the Algebra I Tutor developed by Carnegie Learning Inc. The Algebra I Tutor is a Cognitive Tutor for high school algebra used in real classroom situations in more than about 2000 schools nationwide in the United States [11].

We used tutor-interaction logs representing interactions between the Cognitive Tutor and individual human students. The current data were collected from a study conducted in an urban high school in Pittsburgh. There were 81 students involved in the study. There were 15 sections taught by the tutor, which covered most of the skills necessary to solve linear equations. In the current study, we focus on the log data for the first eight sections. The equations in those sections only contain a single unknown and have the form A+B=C+D where each of A, B, C, and D is either a constant or an unknown term in the form of Nx with a constant number N.

3.1. Tutoring interaction

The unit of analysis in the current paper is a single *problem-solving step* performed by the human students using the Cognitive Tutor interface. A problem-solving step is slightly different from an *equation-transformation step*, which corresponds to an action that transforms one equation into another complete equation when solving an equation with paper and pencil.

There are two types of problem-solving steps: (1) an *action step* is to select an algebraic operation to transform an equation into another (e.g., to declare that I will "add 3x to both sides" of the equation), and (2) a *type-in step* is to do a real arithmetic calculation (e.g., "to enter -4" as a result of adding 3x to -4-3x). By performing these problem-solving steps, a given equation is transformed as follows: a student first selects an action and then applies it to both sides of the equation. For example, for the equation shown in Figure 1 (a), the student selected "Add to both sides" from the pull down menu (b), which in turn prompts the student to specify a value to add (c). This completes the first problem-solving step, which is an action step. The student then enters the left- and right-hand sides of the new equation separately, in two type-in steps. Figure 1 (d) shows the moment when the student has just typed in the left-hand side.



(a) A given equation (b) Selecting an action

(c) Entering a value to be added (d) Typing-in a left-hand side

Figure 1: Screen shot for the Algebra I Tutor.

In summary, three problem-solving steps correspond to a single equation-transformation step, which transforms one equation into another. Sometimes, however, the tutor carries out the type-in steps for the student, especially when new skills are introduced.

When a student makes an error, the tutor provides feedback. The student can also ask for a hint (by pressing the "[?]" button on the left side of the tutor window) when he/she gets stuck.

Every time a student performs a step, the tutor logs it. In this study, we are particularly interested in the following log information: (1) The place where the step was made, called *Selection*. This corresponds to an element on the graphical user interface, e.g., the left-hand side of an equation, or a pull-down menu on the menu bar. (2) The name of the skill used, called *Action*, which is either the name of the algebraic operation selected from the menu for an action step, or the symbol "type-in" for a type-in step. (3) The value entered, called *Input*. For example, the value specified (12.29 in Figure 1 (c)) to be added to both sides for the "add" action step mentioned above, or the left- and right-hand side entered for the type-in steps. (4) The correctness of the step, which is either "correct", "error", or "hint" (when the student asked for a hint).

Using the first three pieces of information mentioned above, a problem-solving step is represented with a tuple <Selection, Action, Input>. The tuple plays an important role for SimStudent's learning, as described in section 4.2.

3.2. Cognitive model and model-tracing

As mentioned above, the tutor provides immediate feedback on the steps. This is possible because the tutor has a *cognitive model*, represented as a set of production rules that are sufficient to perform the target task. A cognitive model can include both correct and incorrect (called "buggy") production rules. When a step is performed by a student, the tutor attempts to find a production rule that matches the step. This technique is called *model-tracing*.

4. The Architecture of SimStudent

This section is a brief overview of SimStudent. We first explain how SimStudent learns cognitive skills from demonstrations. The double meaning of "demonstration" in the current context is then explained – a demonstration by an author who is building a Cognitive Tutor, and a "demonstration" in the tutor-interaction log representing a real student's performance. Due to the space limitation, we only provide a brief overview. See Matsuda et al [1] for more details.

4.1. Modeling student-tutor interaction in the Algebra I Tutor

SimStudent learns a single production rule for each problem-solving step demonstrated. In the most general case, when demonstrating a step, the demonstrator must specify two additional things: (1) a skill name, and (2) a focus of attention. In the study using Algebra I Tutor log, however, both of these could be derived without explicit specification as explained below.

The *skill name* must be consistent across problems throughout the demonstration. In the Algebra I Tutor, the skill name for an action step is an abbreviation of the action name shown in the tutor interface. For example, in the step shown in Figure 1 (a-c), the skill name to select "Add 12.29 to both sides" is called "add." The skill name for a type-in step consists of the skill name of the corresponding action step and "-typein." So, for example, the skill name for the type-in step shown in Figure 1 (d) is called "add-typein."

The *focus of attention* is the set of elements on the tutor interface used to determine what action is to be taken. For example, the problem-solving step shown in Figure 1 (b) – "add 12.29 to both sides" – requires two elements, "5.37" and "-12.29+5.53y" as the focus of attention. There is, however, an issue in getting the focus of attention when SimStudent is used to model real students' performance – the real students do not indicate their focus of attention. We have assumed that both the left-hand side and right-hand side are used as the focus of attention for the action steps. For the type-in steps, we presume that the skill name and the expression immediately above the expression to be typed-in are the focus of attention. So, for example, for the type-in step shown in Figure 1 (d), where "5.37+12.29" was typed-in, the skill name "Add 12.29 to both sides" and the expression "5.37" are used as the focus of attention.

4.2. Learning algorithm

Production rules learned by SimStudent are written in the Jess production rule description language [12]. A production rule consists of three major parts: what, when, and how. The what-part specifies which elements of the tutor interface are involved in the production rule. The when-part shows what feature conditions must hold among the elements in the what-part for a production rule to be fired. The how-part specifies what computation should be done with the what-part to generate a correct "Input" (described in section 3.1).

SimStudent utilizes three different learning algorithms to learn these three parts separately. The what-part is learned as a straightforward generalization of the focus of attention. Each element in the focus of attention can be identified uniquely in terms of its location on the tutor interface. Constraints on the location are generalized from the most specific description using absolute positions (e.g., the *2nd* and the *3rd* cells) to the most general description that represents arbitrary elements (e.g., *any two* cells). A moderate generalization uses relative locations (e.g., *two consecutive* cells).

SimStudent uses FOIL [13] to learn the when-part. The target concept is the "applicability" of a particular skill given a focus of attention. When a step for a particular skill S is demonstrated, the instance of demonstration serves as a positive example for the skill S and a negative example for all other skills. We provide FOIL with a set of feature predicates as the background knowledge with which to compose hypotheses for the target concept. Some examples of such feature predicates are `isPolynomial`(_), `isNumeratorOf`(_,_), `isConstant`(_). Once a hypothesis is found for the target concept, the body of the hypothesis becomes the what-part on the left-hand side of the production rule. For example, suppose that FOIL found a hypothesis S(A, B) :- `isPolynomial`(A), `isConstant`(B) for the applicability of the skill S. The left-hand feature condition for this production rule says that "the value specified in the first focus of attention must be polynomial and the second value must be a constant."

SimStudent uses iterative-deepening depth-first search to learn an operator sequence for the right-hand side of the production rules. When a new instance of a particular skill is demonstrated, SimStudent searches for the shortest operator sequence that derives the demonstrated action (i.e., the "Input") from the focus of attention for all instances demonstrated thus far. The operators are provided as background knowledge.

5. Modeling Real Students

How well does SimStudent predict real students' performance? How often does SimStudent make incorrect predictions, whether reporting a correct step as incorrect or an incorrect step as correct? To answer these questions, we analyzed SimStudent's fidelity at predicting human students' performance. We are interested not only in how well SimStudent predicts real students' correct steps, but also – or even more, for educational purposes – in how well it predicts incorrect steps.

5.1. Data: Tutor-interaction log

The students' learning log data were converted into *problem files*. Each problem file contains the sequence of problem-solving steps made by a single real student for a single equation problem. There are three types of steps: *correct*, *buggy*, and *error*. The correct steps are those that were model-traced with a correct production rule by the Carnegie Learning Tutor. The buggy steps are those that were model-traced with a buggy production rule. The error steps were not model-traced either with a correct or a buggy production rule.

There were 1897 problems solved by 81 individual human students. There were a total of 32709 problem-solving steps. These steps contain 21794 (66.7%) correct steps, 2336 (7.1%) buggy steps, 4567 (14.0%) error steps, and 4012 (12.3%) hint seeking steps[2]. Since the Algebra I Tutor already has a wide variety of buggy rules based on empirical studies, the buggy steps likely capture a fair proportion of the incorrect steps that the human students are likely to make. Furthermore, we have found (by manually verifying data) that most of the error steps are likely to be due to slips and random errors in using the tutor interface, which makes the error steps most challenging to be modeled. Thus, we only used correct and buggy steps for the current evaluation.

5.2. Method: Validation

For each individual human student, we have taken the first 15 problems for training and the following five problems for testing. Using these problems, the validation was conducted for *each individual human student* separately as follows:

```
For each of the 15 training problems
    Train SimStudent on the correct and buggy steps  /* learning */
    For each of the five test problems
        Attempt to perform the correct steps  /* validation */
```

Production rules had been updated each time a five-problem validation test was taken place. Notice that only the steps correctly performed by the real student were used for validation. This is because we are evaluating whether SimStudent could correctly *predict* if a real student would successfully perform a step or not. A student's performance on a step is coded as "success" if the *first attempt* at the step is correct; otherwise it is coded as "error." The Cognitive Tutor forced real students to perform each step repeatedly until correct in order to proceed to the next step. Therefore, a chain of correct steps in a log for a single represents an entire solution for the problem. Hence, for our purpose, it is sufficient to have SimStudent perform only those solution steps: if SimStudent cannot correctly perform a step, it is a prediction that the real student would also fail to perform that step correctly on the first attempt.

[2] Asking for a hint is logged as a problem-solving step. As mentioned in section 5.3.2, whenever a real student asked for a hint on a step, the student's attempt was coded as "error."

5.3. Result: Analysis of prediction

59 students solved at least 20 problems. On average, there were a total of 116.8 correct and buggy steps in the 15 training problems, and 55.6 correct steps in the five test problems.

There were 10 different skills in the log data. One skill appeared in the training problems very rarely (27 times total across the 59 students) and hence was excluded from the analysis. The remaining nine skills included five skills for the action steps (add, subtract, multiply, divide, and clt, which is to "combine like terms") and four skills for the type-in steps (add-typein, subtract-typein, multiply-typein, and divide-typein).

We first show the overall performance of problem-solving attempts and then discuss the analysis of predicting real students' performance.

5.3.1. Learning curve – how well did SimStudent learn cognitive skills?

Figure 2 shows the average ratio of successful attempts at performing steps in the test problems, aggregated across all skills and students. The x-axis shows the number of opportunities that SimStudent had had to learn the particular skills whose performance is shown on the y axis at that point.

The average ratio of successful attempts increased as the opportunities of learning increased. After training for 16 times, more than 80% of the steps in the test problems were performed correctly.

This result shows that SimStudent did actually learn cognitive skills from the tutor-interaction log of the real students. Below, we will explore two further issues: How many correct steps in the test problems were predicted as correct? Did SimStudent learn overly general rules that had a tendency to perform steps incorrectly?

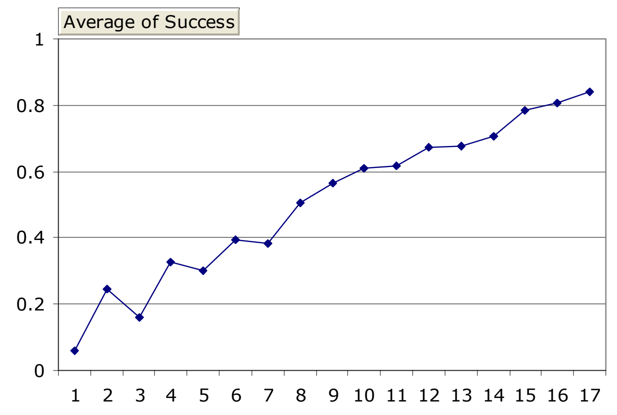


Figure 2: Learning curve showing average ratio of successful problem-solving attempts aggregated across all skills and all students. The x-axis shows the number of opportunities for learning a skill

5.3.2. Analysis of errors in predicting real students performance

The primary purpose of this study is to *predict* real students' performance. Hence SimStudent should not only perform correctly on the steps in which the real students performed correctly but also fail to perform steps in which the real students failed. Thus, a better analysis is to count the number of matches between the real students' and SimStudent's performance on the test problems.

We define the result of a prediction as follows: first, the result of performing a step is a *success* if there is a production rule fired that reproduces the step correctly; and an *error* otherwise. The real student's step is a *success* if he/she performed the step correctly at the first attempt; and an *error* otherwise[3]. We then define a prediction to be (1) *true positive* (TP), if both SimStudent and the human student's performance were a success, (2) *false positive* (FP), if the SimStudent's attempt was a success while the student's performance was an error, (3) *false negative* (FN), if the result of the SimStudent's attempt was an error, but the student's performance was a success, and (4) *true negative* (TN), if both the result of the SimStudent's and the student's performance were an error. We define the following measures: *Accuracy* = (TP+TN)/(TP+FN+FP+TN), *Precision* = TP/(TP+FP), *Recall* = TP/(TP+FN), *E[rror]-Precision* = TN/(TN+FN), *E[rror]-Recall* = TN/(TN+FP). In this study, we are particularly interested in

[3] This includes cases where the student requested a "hint" on the step.

E-Precision and E-Recall, because SimStudent ought to model not only the real student's successful performance, but also errors.

Figure 3 shows the result of the analysis of predicting real students' performance. As in Figure 2, the x-axis again shows the number of opportunities to learn a skill. The y-axis shows the average for each measure aggregated across all students and skills. As conjectured from the result of the previous section, the Recall increased as SimStudent was trained on more steps – showing the ability for SimStudent to learn rules that perform steps in which the real students succeeded.

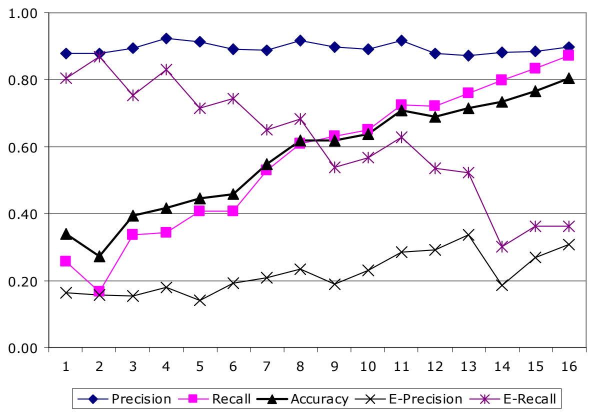


Figure 3: An analysis of predicting real students' performance. The X-axis shows the frequency of learning. The Y-axis shows the average of each measurement aggregated across all students and all skills.

That the Precision stayed high regardless of the number of training examples is rather surprising. It turned out, however, that the real students in the current particular log data made only a very few error steps – only 15% of the steps in the test problems were error steps and the ratio of error steps was stable across the different frequencies[4].

Finally, and most importantly, the E[rror]-precision increased slightly, but overall it stayed low. This means that SimStudent did not accurately predict real students' error steps. To our surprise, the E[rror]-recall decreased as learning progressed. This indicates that as learning progressed, SimStudent tended to learn more production rules that correctly performed those steps that were performed incorrectly by human students. In other words, the current implementation of SimStudent is not correctly predicting human students' erroneous performance.

5.3.3. Analysis of errors of commission

One of the key concerns in the previous analysis is whether and how often SimStudent learned overly general rules. To address this issue, we asked SimStudent to show next steps that can be achieved for each of the steps in the test problems, and assess the correctness for each of these steps. Such evaluation is quite costly, because it requires an oracle for the judgment. We have not implemented the oracle for the analysis just yet. Instead, we have manually gauged the correctness of the rule firing by inspecting the conflict set of the production rules each time a step is about to be performed by SimStudent. We have randomly selected three real students from the log data for this analysis. There were a total of 102 steps in 15 test problems where 6 skills (add, divide, subtract, add-typein, divide-typein, and subtract-typein) were tested.

Table 1 shows the total number of rules appearing in the conflict set during the test. *True Firing* shows the number of production rules that, if applied, generate a correct step. *False firing* shows the number of production rules that, if applied, generate an incorrect step. On average,

	Rule firing	
	True Firing	False Firing
add	146	22
add-typein	15	47
subtract	145	16
subtract-typein	13	15
divide	19	35
divide-typein	14	9
Total	352	144

Table 1: Total number of rules in the conflict set for each of the skills. The *False Firing* shows the number of incorrect rule applications – i.e., if applied, the production rule generated an incorrect step.

[4] This does not mean that the real students did not learn. Indeed, the *number* of error steps decreased, but the ratio of error steps to the correct steps stayed the same – there was always about a 15% chance that the real students made an error step in this particular data set.

there were one or two overly general rules, for each of the steps in the test problem, that lead to a wrong step. More surprisingly there were very many opportunities for “add” and “subtract” rules to be applied correctly. This observation agrees with the high precision mentioned above. It also explains why the E[rror]-recall rapidly decreased as the learning progressed – SimStudent quickly learned those rules, which resulted in covering more steps that the real students failed to perform correctly.

6. Conclusion

Using a genuine tutor-interaction log as “demonstrations” of real students performing their skills, SimStudent did learn to generate a student model that can predict more than 80% of the *correct* steps performed by real students.

However, it turned out that SimStudent does not accurately predict real students’ *incorrect* steps. Predictions are produced by a student model that is overly general hence, by definition, covered more steps than it ought to cover – SimStudent correctly performed steps that were not performed correctly by real students (low E[rror]-precision). Also, SimStudent learned rules that correctly perform steps, but in a different way than the ones performed by the real students.

Can we use SimStudent as a pedagogical component for an intelligent tutoring system? Currently, the answer leans toward the negative. We are still exploring the issues. One problem may be an incorrect model of students’ prior skills. The solution may require different learning methods; the current SimStudent is designed for fast construction of cognitive models using programming by demonstration, not for student modeling.

References

1. Matsuda, N., W.W. Cohen, and K.R. Koedinger, *Applying Programming by Demonstration in an Intelligent Authoring Tool for Cognitive Tutors*, in *AAAI Workshop on Human Comprehensible Machine Learning (Technical Report WS-05-04)*. 2005, AAAI association: Menlo Park, CA. p. 1-8.
2. Koedinger, K.R., V.A.W.M.M. Aleven, and N. Heffernan, *Toward a Rapid Development Environment for Cognitive Tutors*, in *Proceedings of the International Conference on Artificial Intelligence in Education*, U. Hoppe, F. Verdejo, and J. Kay, Editors. 2003, IOS Press: Amsterdam. p. 455-457.
3. Cypher, A., ed. *Watch what I do: Programming by Demonstration*. 1993, MIT Press: Cambridge, MA.
4. Matsuda, N., et al., *Evaluating a Simulated Student using Real Students Data for Training and Testing*, in *Proceedings of the International Conference on User Modeling*. 2007; in press.
5. Ikeda, M. and R. Mizoguchi, *FITS: a framework for ITS - a computational model of tutoring.* International Journal of Artificial Intelligence in Education, 1994. **5**: p. 319-348.
6. Baffes, P. and R. Mooney, *Refinement-Based Student Modeling and Automated Bug Library Construction.* Journal of Artificial Intelligence in Education, 1996. **7**(1): p. 75-116.
7. Langley, P. and S. Ohlsson, *Automated cognitive modeling*, in *Proceedings of the Fourth National Conference on Artificial Intelligence*. 1984, AAAI: Melon Park, CA. p. 193-197.
8. MacLaren, B. and K.R. Koedinger, *When and why does mastery learning work: Instructional experiments with ACT-R "SimStudents".* in *Proceedings of the 6th International Conference on Intelligent Tutoring Systems*, S.A. Cerri, G. Gouarderes, and F. Paraguacu, Editors. 2002, Springer-Verlag: Berlin. p. 355-366.
9. Anderson, J.R., *Rules of the mind*. 1993, Hillsdale, NJ: Erlbaum.
10. Muggleton, S. and C. Feng, *Efficient Induction Of Logic Programs*, in *Inductive Logic Programming*, S. Muggleton, Editor. 1992, Academic Press: London, UK. p. 281-298.
11. Koedinger, K.R. and A. Corbett, *Cognitive Tutors: Technology Bringing Learning Sciences to the Classroom*, in *The Cambridge Handbook of the Learning Sciences*, R.K. Sawyer, Editor. 2006, Cambridge University Press: New York, NY. p. 61-78.
12. Friedman-Hill, E., *Jess in Action: Java Rule-based Systems*. 2003, Greenwich, CT: Manning.
13. Quinlan, J.R., *Learning Logical Definitions from Relations.* Machine Learning, 1990. **5**(3): p. 239-266.