Teaching the Reduction Technique with Interactive Visualizations: Report on a Simple Automatic Tutor

Christian Pape
Universität Karlsruhe
Institut für Logik, Komplexität und Deduktionssysteme
76128 Karlsruhe
pape@ira.uka.de

Abstract

The reduction technique is an important problem solving method often used in computer science and mathematics to solve a new problem by the known solution of another problem. In this paper we investigate the potential of interactive visualization for teaching the reduction method and focus on the following two questions: What part can visualizations play in the presentation and understanding of the reduction technique? and: How can we build systems for teaching the reduction technique that can be used by students in a teleteaching environment? We try to answer the latter question by reporting on a field test of an automatic tutor which we have build for our students.

1 Introduction

Most branches of computer science deals with problem solving with the help of a computer. For this purpose many methods have been developed over the years to construct algorithms and solutions for new problems systematically. One of the most important problem solving techniques in computer science is the reduction method: the solution of a new problem is constructed with help of the known solution of another problem. But even if no priority is given to the algorithmic solution of a problem then the reduction method still can be used to investigate the problems themselves: For instance, their complexity (like in complexity theory), or whether a problem is solvable with a computer program at all (like in computability theory). In particular, the reduction technique is an important method used in theoretical computer science.

We will report in this paper on our ongoing activities to use visualizations in teaching theoretical computer science at the undergraduate level. This, of course, includes teaching the reduction technique. This area poses a challenge to the multi-media approach. How can we visualize the abstract concepts that occur in theoretical computer science? In our recent work we presented a framework together with an example for enhancing the presentation of rigorous proofs with interactive visualizations [4]. We also developed a special assistant with which our students are able to solve (as an exercise during their homework) a certain NP-complete problem much better than to try to solve it only by paper and hand [3]. All our visualizations are integrated into an HTML-based hyper text lecture notes[1] which we developed for our introductional course in theoretical computer science. The hyper text can be used by our students for further studies at home or at the universities computer lab with every Java capable WWW-browser.

It is commonly agreed that an active training is essential for successful learning. Merely reading texts on the World-Wide-Web, looking at visualizations in electronic textbook and watching videos or live transmissions of lectures is not sufficient to learn and finally to apply analytic abilities like the reduction technique. Traditionally, exercises play an important role to acquire new techniques: The student solves at home a given exercise and later on the solution is checked by and discussed with the teacher or a human tutor. This approach can also be integrated in a teleteaching environment by exchanging the formulation of the exercise, the students solution and its correction via letter post or e-mail. However, even in the case of using e-mail a response usually is delayed for a significant period of time because the human tutor can not devote himself to only one student. Hence, in a teleteaching environment it is helpful to partially replace the human tutor by an automatic tutor or an intelligent tutor system [5].

In this work we focus on teaching the reduction technique and address two questions:

- What part can visualizations play in the presentation and understanding of the reduction technique?

- Can we build systems for teaching the reduction technique that can be used by students instead of human tutors to solve exercises in a teleteaching environment?

After an abstract description of the reduction technique given in section 2 we briefly describe our recent experiments of visualizing certain reductions (sec-

[1]`http://i12www.ira.uka.de/~info3/skript/companion/companion.html`

tion 3). In particular, we point out how an automatic tutor for certain reductions can be used by students to improve their solutions of reductions carried out as an exercises during their normal homework. We hope that with our approach to teach reductions in theoretical computer sciences the students are able to build own mental models of the abstract objects and their relation much better. Moreover, we expect that the additional interaction facilities of an automatic tutor help them to uncover misconceptions before their final solution is checked by human tutors. In section 4 we report in detail on our implementation of such an assistant. An evaluation of this automatic tutor among our students shows that such an automatic tutor often helps the students to improve their solution of the exercise before it is checked by human tutors.

2 The Reduction Technique

One of the most important problem solving methods in computer science and mathematics is the reduction method: a the solution of a new problem is constructed with help of the known solution of another problem. This technique is used in various domains of computer science like, for instance: in complexity theory to find lower time bounds for problems or to prove the NP-completeness of problems; in the theory of computability to prove the undecidability of decision problems; or in applied computer science like compiler construction where a high level computer program is transformed (reduced) to a low level computer program. The basic structure of such reductions is always the same: One problem is mapped to another one such that certain domain dependent properties are preserved, e.g. the solubility of the decision problem or the semantic of the computer program to be compiled. Because we are normally interested in solutions that can be implemented on a computer one essential property of the transformation is its computability.

The objects transformed by the mapping depend heavily on the application of the reduction technique. As in the above example they can be as different as, for instance, computer programs or decision problems. We can abstract of these specific objects by encoding them somehow into words over a finite alphabet, e.g. as sequences of zeros and ones as it is done with every representation in a computer. Therefore, we can use languages (sets of words) as an abstract description of the objects. The reduction technique then can be described as follows:

Definition 1 (Reduction)
Given a finite alphabet Σ and two languages $L_1, L_2 \subseteq \Sigma^*$. A *reduction* of L_1 on L_2 is a computable transformation $\tau : \Sigma^* \to \Sigma^*$ with $w \in L_1$ iff $\tau(w) \in L_2$ for all $w \in \Sigma^*$.

In this definition L_2 is the collection of problems for which we know (or assume to know) a solution and L_1 is a collection of problems for which we do not know a solution. For example, if L_1 is the set of all syntactical correct C-programs, then τ may be a compiler that transforms every C-program w to a lower level computer program $\tau(w)$ which can be executed directly on a computer. But even if we only assume that all problems in L_2 can be solved, then we can use the reduction technique to prove certain properties of L_2 by known properties of L_1. This is done, for example, in reduction proofs in the theory of NP-completeness. Therefore, it is a good idea to distinguish between the problem for which an algorithmic solution exists (or we assume one to exist) and the problem for which we know some properties. Let us call the first one the *solvable problem* and the second one the *known problem*. In a reduction, the solvable problem always is L_2 and the known problem can either be L_1 (e.g. an NP-complete problem) or L_2 (e.g. the target program language of a compiler).

3 Teaching the Reduction Technique with Visualizations

We teach the reduction technique at the University of Karlsruhe in our introductional course in theoretical computer sciences on the undergraduate level. In this course we present many different reductions with the support of an additional visualization of the relation between the known problem and the new one. During the lecture this is done in the traditional fashion by a pictorial description of an instance w of one problem and a pictorial description of the corresponding transformed problem $\tau(w)$. By nature of the presentation (slides and overhead projector) these pictures are, of course, static and the examples are fixed.

In addition, our hyper text lecture notes contains some *interactive* visualizations — static and dynamic — of reductions. The instances of the problems are visualized by pictorial descriptions as well, but instead of choosing a fixed example the student can type in arbitrary instances of the problem and the transformation then is applied to this instance automatically. With such interactive visualizations the students have the opportunity to investigate the reduction in detail for further studies at their own pace — either at workstations in the universities computer lab or at home via a modem and a personal computer. In [4] we report on an example of such an interactive visualization which deals with a transformation τ of Turing machines with two-way infinite tape onto Turing machines with one-way infinite tape. The students can type in an arbitrary instance T of a Turing machine with two-way infinite tape. Then the relation between T and the transformed Turing machine $\tau(T)$ with one-way infinite tape is visualized by a simultaneous simulation of T and $\tau(T)$. This is similar to compile (reduce) a computer program (the Turing machine T) to another one (the Turing machine $\tau(T)$) and visualize in parallel the changes of internal states of both programs.

In all reductions presented during the lecture and in the above example the transformation of the reduction is well known in advance: the students only have to understand its usage in the reduction. But for learning the reduction technique it is not sufficient to merely reproduce the presented examples. Therefore, our students also have to develop own reductions as an exercise, especially to prepare themselves for the final exam. The main difficulty in developing correct reductions is to find a suitable transformation, i.e. that satisfies certain conditions which depend on the specific domain. Visualizations play an important role for the investigation of a candidate τ for a valid transformation: τ can be applied to selected instances w of the problem and — like it is done in the lecture – the relation between w and $\tau(w)$ can be visualized by pictorial representations of w and $\tau(w)$. However, this task can be tedious and often mistakes are made if the transformation is applied to the instances by hand or if the instances become too large. Therefore, it is a much better idea to apply the transformation to these instances automatically. We implemented this approach for an exercise taken from the theory of NP-completeness. The students task is to construct a mapping of non-deterministic Turing machines to an NP-complete tiling problem. With our software the student is able to type in the transformation and an instance of a Turing machine which then is mapped automatically to an instance of the tiling problem [3]. The students solution of this exercise is still checked by human tutors.

An even better way to learn reductions is a software with the above facilities and which, in addition, is able to verify the students transformation automatically and give hints on errors at once. With help of the immediate respond of such an automatic tutor the student is able to cover up misconceptions much earlier as it is possible with a human tutor. Such a software has to check certain properties of the transformation. Unfortunately this approach, in general, is impossible because even checking simple properties of a given function is known to be undecidable. This makes it very difficult (if not impossible) to build a general teaching system like, for instance, an intelligent tutor system for checking transformations. However, our example presented in the following section shows that for educational purposes we can restrict the amount of possible and valid transformations for a specific reduction problem such that implementing an full automatic tutor becomes feasible.

4 An Example taken from Complexity Theory

Computer scientist usually work on the development of algorithms to solve problems efficiently with the help of a computer. The efficiency (or complexity) of an algorithm is measured in terms of its used time and space resources in dependence of the given input. In complexity theory the problems themselves come to the fore and it is investigated how efficient these problems can be solved by algorithms. For this purpose all problems of the same complexity, i.e. problems that can be solved with algorithms of the same complexity, are put together into sets, so called complexity classes. Before we start with the description of our automatic tutor (given in section 4.1) we briefly recall some necessary notions of elementary complexity theory (we refer to [2] for a detailed description).

The problems are traditionally given as decision problems in a standard format consisting of two parts: The first part specifies a generic instance of the problem in terms of sets, functions, graphs, etc, and the second part is a yes/no question asked in terms of the generic instance. The following definition gives an example of one central problem in complexity theory: The problem is to decide whether a set of propositional clauses of a specific form is satisfiable or not:

Definition 2 (3SAT)

- INSTANCE: A set S of clauses such that each clause in S consists of at most three literals.
- QUESTION: Is S satisfiable?

As explained in section 2 we assume that instances of such problems are coded somehow into a language $L \subseteq \Sigma^*$. Furthermore, we identify the name, e.g. 3SAT, of the problem with the set of all instances for which the question can be answered with “yes”.

In our course we teach the basics of the theory of NP-completeness, in particular the relation between the classes P, i.e. the set of all problems which can be solved in polynomial time on a deterministic Turing machine, and NP, i.e. the set of all problems which can be solved in polynomial time on a non-deterministic Turing machine. For the investigation of the relation between P and NP the notion of NP-completeness is introduce:

Definition 3 (NP-completeness)
Given a finite alphabet Σ. Then a a problem $L \subseteq \Sigma^*$ is *NP-complete* iff L is in NP and for every problem $L' \in NP$ there exists a transformation τ computable in polynomial time such that for every $w \in \Sigma^*$ the following holds:

$$w \in L' \text{ iff } \tau(w) \in L$$

In other words: a problem $L \in NP$ is NP-complete iff every problem $L' \in NP$ is reducible to L in polynomial time such that an instance of L' has a solution iff the transformed instance has a solution as well. Problems in P are said to be feasible, because they can be solved in polynomial time on a computer, whereas NP-complete problems are said to be unfeasible, because currently only exponential time algorithms exists to solve these problems. Many real problems are unfeasible in this sense and it is of practical interest to know whether a problem is NP-complete or not. If we can solve an NP-complete problem L in polynomial time on a deterministic Turing machine, i.e. $L \in P$, then $P = NP$ holds and, consequently, by definition of NP-completeness all problems in NP can be solved in polynomial time as well.

It is easy to see from the definition of NP-completeness that we can use the reduction technique to prove the NP-completeness of a problem L_2 by (1) showing $L_2 \in NP$ and (2) by reducing another problem L_1 — known to be NP-complete — to L_2. The condition $w \in L'$ iff $\tau(w) \in L$ provides that the solubility of the problems is preserved by the transformation τ, i.e. if the question of an instance can be answered positively then the question for the transformed instance also can be answered positively, and vice versa. If the new problem L_2 can be solved in polynomial time on a deterministic Turing machine then L_1 can be solved in polynomial time as well. Hence, in our terms L_1 is the known and L_2 is the solvable problem.

However, to apply the reduction technique we first need a problem which we know to be NP-complete. In our lecture we use SAT, i.e. the satisfiability of propositional formulas, as a starting point for further reductions The NP-completeness proof of SAT is based on a mapping of non-deterministic Turing machines to instances of SAT [1]. Afterwards, we present to our students a reduction of SAT on 3SAT in a proof for the NP-completeness of 3SAT. The central part of this proof is a satisfiability preserving transformation τ from instances of SAT to instances of 3SAT, i.e. an instance w of 3SAT is satisfiable SAT iff the instance $\tau(w)$ of 3SAT is satisfiable. To gain experience with such reduction proofs the students have to prove the NP-completeness of the following variant of 3SAT as an exercise:

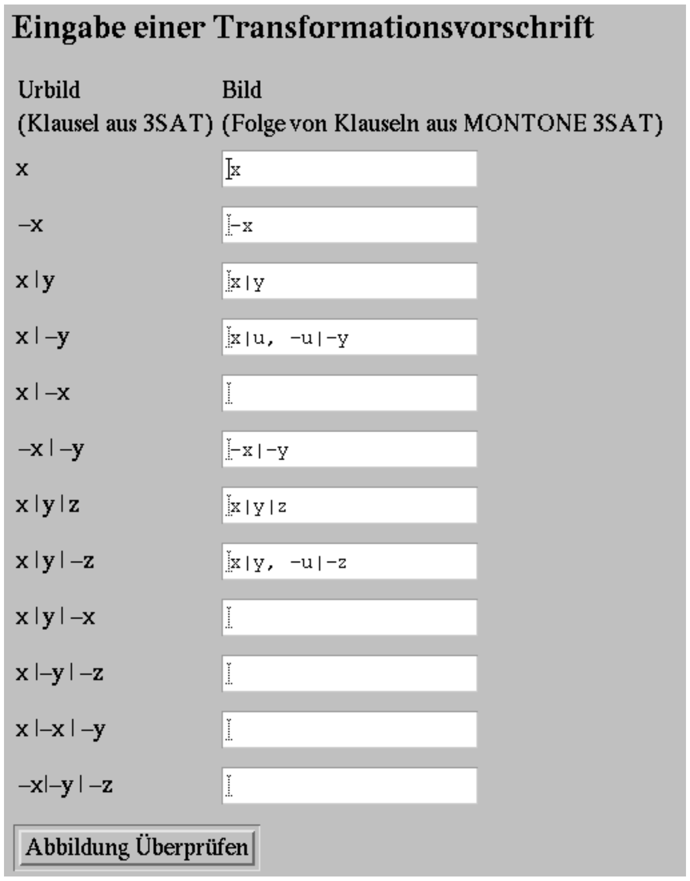
Definition 4 (MONOTONE 3SAT)

- INSTANCE: A set S of clauses such that each clause $c \in S$ consists of at most three literals and c contains at least one positive and one negative literal.
- QUESTION: Is S satisfiable?

The proof for the NP-completeness of MONOTONE 3SAT is based on a reduction from 3SAT to MONOTONE 3SAT such that every clause of an instance of 3SAT that contains only positive or only negative literals is transformed into clauses containing at least one positive and one negative literal. We have developed a computer program with which our students can check their solution. The program automatically verifies the transformation and gives detailed hints in cases of mistakes.

4.1 An Automatic Tutor for a Reduction of 3SAT on MONOTONE 3SAT

Figure 4.1 shows the input interface of our automatic tutor as the students can see it in our hyper text lecture notes (except for colours). For reasons of authenticity we will not translate the German text parts into English.

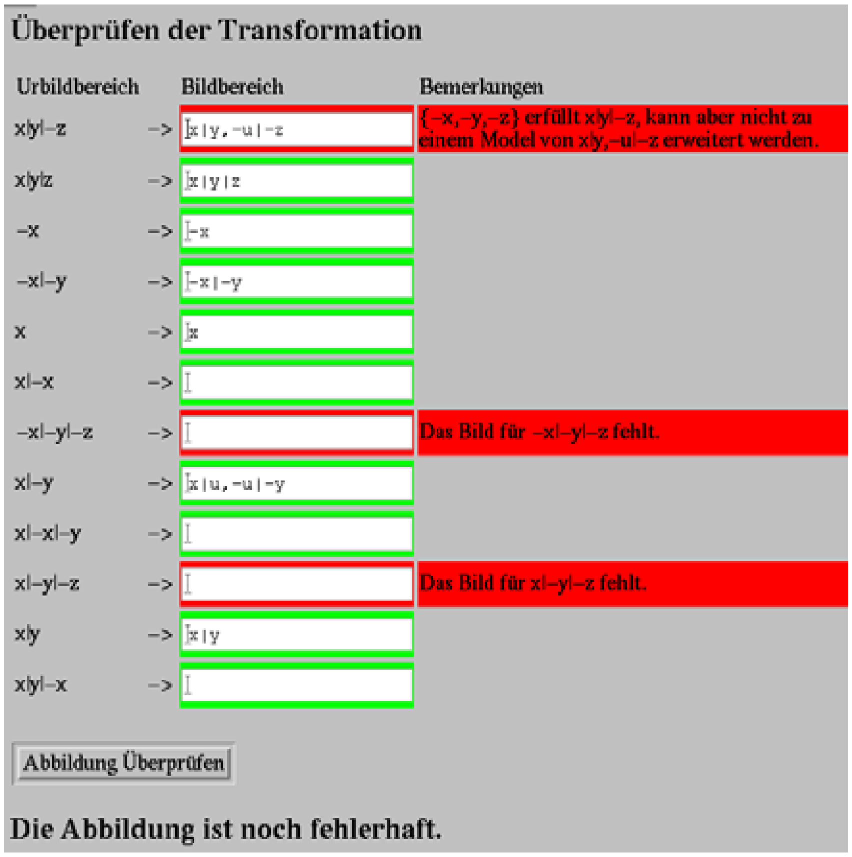


Eingabe einer Transformationsvorschrift

Urbild (Klausel aus 3SAT)	Bild (Folge von Klauseln aus MONTONE 3SAT)
x	x
–x	-x
x | y	x | y
x | –y	x | u, -u | -y
x | –x	
–x | –y	-x | -y
x | y | z	x | y | z
x | y | –z	x | y, -u | -z
x | y | –x	
x |–y | –z	
x |–x | –y	
–x|–y | –z	

Figure 1: Input of a transformation

With this input interface a transformation τ : MONOTONE 3SAT $\rightarrow$ 3SAT can be defined by typing in the images of τ on 12 different types of clauses. These clauses are all relevant combinations of clauses with one to three positive or negative literals. For each such clause C (first column “Urbildbereich” of figure 4.1) a set of clauses — the mappings image $f(C)$ — can be typed in (second column “Bildbereich”). Clauses are given as a sequence of literals separated by “$|$” (which serves as the symbol for logical disjunction). Literals are either variables (given as small letters) or variables with preceding negation sign “$-$”. A set of clauses is given as a sequence of clauses each separated by a comma. The fourth row in figure 4.1, for example, defines $\tau(x \vee \neg y) = \{x \vee u, \neg u \vee \neg y\}$ to be all non tautological clauses with exactly one positive and one negative literal. u is a newly introduced propositional variable.



Überprüfen der Transformation

Urbildbereich		Bildbereich	Bemerkungen
x|y|–z	–>	x|y,-u|-z	{–x,–y,–z} erfüllt x|y|–z, kann aber nicht zu einem Model von x|y,–u|–z erweitert werden.
x|y|z	–>	x|y|z	
–x	–>	-x	
–x|–y	–>	-x|-y	
x	–>	x	
x|–x	–>		
–x|–y|–z	–>		Das Bild für –x|–y|–z fehlt.
x|–y	–>	x|u,-u|-y	
x|–x|–y	–>		
x|–y|–z	–>		Das Bild für x|–y|–z fehlt.
x|y	–>	x|y	
x|y|–x	–>		

Die Abbildung ist noch fehlerhaft.

Figure 2: Output of automatic tutor

If the transformation is typed in completely (or only partially) and after pressing the button “Abbildung überprüfen” (check transformation) it is checked automatically whether the given transformation is suitable for a reduction to MONOTONE 3SAT, in particular if the transformation is satisfiability preserving, or not.

If, as in our example, this is not the case then the program returns detailed information about the students mistake (Fig. 2). For instance, in the 7th row for the clause $\neg x \vee \neg y \vee \neg z$ ($-x \mid -y \mid -z$ in first column “Urbildbereich”) no image was typed in (second column “Bildbereich”). The information about this mistake is given and made stand out by using red colour (darker shades) in the third column “Bemerkungen” (remarks). On account of a serious mistake the program also returns a counterexample which describes why the corresponding part of the transformation is incorrect: an interpretation is given for which the transformation is not satisfiability preserving (e.g. in the first row of figure 2). By this assistance the students can correct and improve their solution before it is finally checked by senior students.

Because it is in general undecidable whether the function τ is satisfiability preserving or not, our program only checks a stronger condition which is decidable:

- Every model of an instance S of 3SAT can be extended to a model of the transformed instance $\tau(S)$ and
- Every model of an instance $\tau(S)$ is a model of S.

As a result of this restriction not all satisfiability preserving transformation can be checked automatically. If τ, for instance, is a satisfiability preserving mapping satisfying the above condition then we can construct another satisfiability preserving mapping τ' by negating every literal in $\tau(S)$. The resulting mapping does not satisfy the above condition but it is a valid transformation for a reduction. However, such transformations are somehow “unnatural”, in particular students usually do not think about such complicated solutions. Therefore, in an educational context the above restriction is not obstructive, quite on the contrary, it is helpful because it also allows us to provide a simple input mechanism of the transformation. Moreover, it is only possible to type in total transformations that can be computed in polynomial time. Note that this makes it not easier for the student to explain in the literal proof why his or her transformation is computable in polynomial time.

The automatic tutor is implemented in the script language perl and it is integrated into our hyper text via the Common Gateway Interface. The program itself is quite small: it contains less than 500 lines of code (without comments). The main part is an implementation of an automatic theorem prover for propositional logic which is used for two purposes: First, to check the restricted satisfiability condition of the transformation and, second, to generate the counterexamples. The source code is available upon request by the author.

5 Evaluation of the Automatic Tutor

To practise the reduction technique our students had to carry out a prove of the NP-completeness of MONOTONE 3SAT as part of their usual homework. The students had two weeks time to work on it and afterwards their written solutions were checked by senior students. As an additional help they had the (optional) opportunity to use our automatic tutor. A few days after the students received their corrected solutions we handed them out a small questionnaire. We wanted to find out about the students opinion about this exercise and, moreover, whether the students had use our automatic tutor and whether the software had helped them to solve the exercise.

58 students answered the questionnaire and 43 of them had worked 1.23h in the average on this exercise which they felt about was a little bit difficult (3.42 on a scale of 1–5 (far too easy — far too difficult)). From these 43 students 23 (53%) found a correct transformation, 11 (26%) found a partially correct transformation and 9 (21%) had written down a wrong transformation. Our automatic tutor was used by 10 (26%) students and 7 (70% out of 10) of them detected mistakes in their solution with help of the software.

This statistic shows that our automatic tutor in fact was of substantial help for many of those students who used the software. But the acceptance of such software among the students is yet quite low: 74% of the students did not use the automatic tutor. There are probably three — mainly technical — reasons for this: First, a lot of students reported that they did not use the automatic tutor because the computer lab was too crowded. Second, many students were able to solve the exercise without any additional help (only nine solutions were wrong). Third, a similar evaluation of a comparable software [3], which also could be used by our students for solving an exercise, shows that the acceptance increases if the formulation of the exercise is only available electronically and contains the software at appropriate points. Nevertheless, these results and also the additional students remarks on the questionnaires confirmed us to be on the right track with our approach to teach central topics of theoretical computer science.

References

[1] S. A. Cook. The complexity of theorem proving procedures. In *Proc. Third Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[2] Michael R. Garey and David S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[3] Christian Pape. Using Interactive Visualization for Teaching the Theory of NP-completeness. In *Proceedings of the World Conference on Educational Multimedia and Hypermedia*. Association for the Advancement of Computing in Education, 1998. to appear.

[4] Christian Pape and Peter H. Schmitt. Visualizations for Proof Presentation in Theoretical Computer Science Education. In Z. Halim, Th. Ottmann, and Z. Razak, editors, *Proceedings of International Conference on Computers in Education, Kuching, Sarawak, Malaysia, December 2–6*, pages 229–236. Association for the Advancement of Computing in Education, 1997.

[5] E. Wenger. *Artificial Intelligence and Education*. Morgan Kaufmann, Los Altos, 1987.