On the Use of a Computer, to Teach More and Better, in a Collective Manner

Jérémy Barbay
Department of Computer Science,
University of British Colombia
201-2366 Main Mall
B.C. V6T 1Z4 Vancouver
CANADA
jeremy@cs.ubc.ca

Abstract: The use of computers to help teaching overcomes a fast development. One of the main objectives is to teach isolated people who can't join regular university courses. We show in this paper that computers can be used to efficiently teach in a classroom without the need of modern and expenseive equipement for students. The paper is divided in two parts: in the first part we expose the principles, results and evaluation of two experiments performed in parallel during the teaching of a one-term fourth year undergrad course. In the second part we expose how those two techniques can be extended in a single more general technique, by using XML to define and manipulate a general purpose database of solved problems; and how such a database can be built in a dynamic and collective way.

For the first experiment, the students of the course had to fill a marked quiz every week. Each week, 61 distinct quizzes were randomly generated from an SGML database of multiple choice questions on the material covered in the 3 last courses. Each student received a distinct personal quiz. This dimish the likely hood of rewarding answers that are copied from one's neighbour. Using various techniques, those 61 quizzes of 4 questions each were marked by a single person in less than one hour, including the time needed to report the marks. For the second experience, each week the students were given an assignment of 4 problems, and received the correction and solution by the next week. The heavy work of writing a detailed solution for each problem was amortized (and will be further amortized) by the constitution of a database of normalized solved problems. Assignments and exams were written in a very short form as references to problems in this database. The index of the database forms then a book of solved problems, with notes of when and where they were previously used. This eases the reuse of exercises over consecutive years, which is otherwise possible but seldom done. The prototypes used in those experiments were developed using a mixture of Java and LaTeX. The next version of the software will use XML to define the solved problems in the database, the assignments and exams, and the transformations performed on those documents to obtain the final result in various formats. Adding a single random operator to the set of transformations already available in XML permits to define all the transformation hardcoded in the prototypes. Such techniques to develop a database of solved problems are more efficient if many instructors work on the same kind of courses. The only technology required is already available to most instructors. Such collaboration could work worldwide in a community of instructors sharing the same language, or even among communities of different languages, with additional constraints.

A tool such as the one described in this paper would make easier the writing of e-teaching software. We think that the implication of the Spanish-speaking research and teaching community in the development and use of such a tool is primordial to escape English/American supremacy on education on the net in the next years.

1. Introduction

At its origins, Computer Science was the art of automation. For a long time computers were able to perform automatic tasks for a large number of professions, but curiously not many for teaching. There are automatic tools for the edition of teaching book of course, but they are merely by-products of scientific edition. Some automatic tools for marking simple exams exist, but they are mostly based on very crude and specialized technologies such as punch cards.

Great efforts have been put into digital publication tools for "E-Teaching". Among those are some automatic tools for the generation of problems, mainly in the software to teach languages. Lastly, some databases of problems were made available on the internet, but they are not normalized, and cannot be easily indexed nor searched. In general, a meaningful integration of the information processing methodes in the education community is yet to come.

We describe in this paper two distinct experiences performed during an undergraduate course in the fourth year of university, and the prototypes used. Those are all based on different databases: we discuss the conclusions from the experiences, and how the features of each prototype which have been proved useful could be unified into another project. Such project can be effective only if developed by a community in a collaborative way, so we discuss various ways to realize this development.

The remaining part of the paper is structured in 4 sections: In the second section we expose the setup of our first experiment, concerning multiple choice question randomly generated and semi-automaticaly corrected. In the third section we describe the database of solved problems developped and used for our second experiment. In the fourth section we summarize the conclusion of those experience. In section 5 we discuss several perspectives of the project.

2. Quiz Automatic Generation and Semi-Automatic Correction

A good way to ensure that students work regularly their course is to give tests often. Obviously, the drawback is that it is very time consuming to generate the exams and to mark them.

One could consider then to give simple Multiple Choices Quizzes, which are easy to generate and much easier to correct, and for which there even exists some mechanical tools of automatic correction. One drawback of this evaluation method is directly linked to its simplicity: it is too easy for the students to copy from their neighbor and cheat when a single quiz is distributed.

The solution we propose here consists in generating randomly one quiz per student, in such a way that:

- all quizzes are approximately of same difficulty;
- each pair of quizzes are distinct with high probability;
- the correction can be done quickly by hand, and could even be done faster with a serial scanner.

It has been tested in a class of 61 fourth-year undergrad students, with 10 quizzes, at the approximate rate of one quiz a week. The correction by hand takes less than one hour, including the report of the marks by name in numerical form. We describe in the following sections the structure of the database used, the structure of the quizzes generated, how they are generated and how the quiz is corrected.

2.2 A Database of Questions

For each exam, a small database of questions is constructed. All the individual quizzes for a

particular exam are generated from such a database. It defines a single entity called a Chain.

• A Chain is mainly composed of a sequence of one or more Sets, each of those corresponding to a question in the generated individual quiz;

• a Set is mainly composed of a sequence of one or more Questions from which only one will be chosen at the random generation of an individual quiz;

• a Question is mainly composed of a Text and a sequence of 4 or more Answers, among which at least one is correct;

• an Answer is either correct or incorrect, and composed of a proposition and a correction;

• various additional pieces of information such as a Title, a Date, a Topic, a Comment, a Text, a Proposition, or an Explanation correspond to text printed in the Quiz generated or in its solution

The Chain corresponds to one exam. A Set corresponds to a set of similar problems which share at least the same topic and the same difficulty, but which can have opposed text and answers. For instance, one question can be ``Which of the following propositions are true?" and another ``Which of the following propositions are false?", both sharing the same sets of propositions but opposed answers.

Many pieces of information in this structure, such as the Explanation of each Answer, are not used in the generation of the quiz, but are used to generate the solution of the quiz, a document which gives the solutions of all the questions asked in the quizzes, so that each individual quiz needs only to be marked with a strict minimum of explanation, as the student can refer to the published solution.

Example:

The database corresponding to a particular exam contains as many Sets as the generated quizzes contain questions. Each Set can contain several Questions, which contain each several Answer Each Answer is of the form <ANSWER> proposition <CORRECT> correction </ANSWER>.

```
<CHAIN>

<SET>   <TOPIC>         Decidability of TM properties           </TOPIC>
<QUESTION>
<COMMENT>       Decidability    </COMMENT>
<TEXT>          Given a Turing Machine $M$, which of those properties are decidable?            </TEXT>
<ANSWER>        $\varepsilon \in L(M)$          <INCORRECT>     Reduct to H?, build M' st $L(M')=\Sigma^*$ or $\empyset$.       </ANSWER>
<ANSWER>        $L(M)=\empyset$                 <INCORRECT>     Reduct to H?, build M' st $L(M')=\Sigma^*$ or $\empyset$.       </ANSWER>
<ANSWER>        $L(M)=\Sigma^*$                 <INCORRECT>     Reduct to H?, build M' st $L(M')=\Sigma^*$ or $\empyset$.       </ANSWER>
<ANSWER>        $L(M)$ is finite                <INCORRECT>     Reduct to H?, build M' st $L(M')=\Sigma^*$ or $\empyset$.       </ANSWER>
<ANSWER>        $L(M)$ is Regular               <INCORRECT>     Reduce to H?, build M' st $L(M')=H?$ or $\empyset$.     </ANSWER>
<ANSWER>        $L(M)$ is Context Free          <INCORRECT>     Reduce to H?, build M' st $L(M')=H?$ or $\empyset$.     </ANSWER>
<ANSWER>        $L(M)$ is Recursive             <INCORRECT>     Reduce to H?, build M' st $L(M')=H?$ or $\empyset$.     </ANSWER>
<ANSWER>        $M$ halts on all input          <INCORRECT>     Reduce to H?, build M' st $L(M')=H?$ or $\empyset$.     </ANSWER>
<ANSWER>        $M$ takes less than S76$ steps on $\varepsilon$.        <CORRECT>       Simulate SM$ for S76$ steps on $\varepsilon$.   </ANSWER>
<ANSWER>        $M$ takes less than S76$ steps on some input.           <CORRECT>       Simulate SM$ for S76$ steps on all inputs       </ANSWER>
<ANSWER>        $M$ takes less than S76$ steps on all input.            <CORRECT>
                Simulate $M$ for $76$ steps on all inputs shorter that $76$.    </ANSWER>
</QUESTION>
<QUESTION>
<COMMENT>       UnDecidability  </COMMENT>
<TEXT>          Given a Turing Machine $M$, which of those properties are (\em not) decidable?         </TEXT>
<ANSWER>        $\varepsilon \in L(M)$          <CORRECT>       Reduct to H?, build M' st $L(M')=\Sigma^*$ or $\empyset$.       </ANSWER>
<ANSWER>        $L(M)=\empyset$                 <CORRECT>       Reduct to H?, build M' st $L(M')=\Sigma^*$ or $\empyset$.       </ANSWER>
<ANSWER>        $L(M)=\Sigma^*$                 <CORRECT>       Reduct to H?, build M' st $L(M')=\Sigma^*$ or $\empyset$.       </ANSWER>
<ANSWER>        $L(M)$ is finite                <CORRECT>       Reduct to H?, build M' st $L(M')=\Sigma^*$ or $\empyset$.       </ANSWER>
<ANSWER>        $L(M)$ is Regular               <CORRECT>       Reduce to H?, build M' st $L(M')=H?$ or $\empyset$.     </ANSWER>
<ANSWER>        $L(M)$ is Context Free          <CORRECT>       Reduce to H?, build M' st $L(M')=H?$ or $\empyset$.     </ANSWER>
<ANSWER>        $L(M)$ is Recursive             <CORRECT>       Reduce to H?, build M' st $L(M')=H?$ or $\empyset$.     </ANSWER>
<ANSWER>        $M$ halts on all input          <CORRECT>       Reduce to H?, build M' st $L(M')=H?$ or $\empyset$.     </ANSWER>
<ANSWER>        $M$ takes less than S76$ steps on $\varepsilon$.        <INCORRECT>
                Simulate $M$ for $76$ steps on $\varepsilon$.   </ANSWER>
<ANSWER>        $M$ takes less than S76$ steps on some input.           <INCORRECT>
                Simulate $M$ for $76$ steps on all inputs       </ANSWER>
<ANSWER>        $M$ takes less than S76$ steps on all input.            <INCORRECT>
                Simulate $M$ for $76$ steps on all inputs shorter that $76$.    </ANSWER>
<ANSWER>        $M$ takes more than S76$ steps on $\varepsilon$.        <INCORRECT>
                Simulate $M$ for $76$ steps on $\varepsilon$.   </ANSWER>
<ANSWER>        $M$ takes more than S76$ steps on some input.           <INCORRECT>
                Simulate $M$ for $76$ steps on all inputs       </ANSWER>
<ANSWER>        $M$ takes more than S76$ steps on all input.            <INCORRECT>
                Simulate $M$ for $76$ steps on all inputs shorter that $76$.    </ANSWER>
</QUESTION>
</SET>
```

Note that the two problems are very similar sets of answer, whose correctness are very different: should a student see that his neighbor has chosen the answer "pocus", he would be wrong half

of the time if choosing to copy the answer. Moreover, as one correct answer and 3 incorrect answers are chosen uniformly at random for each quiz, even if two neighboring student have the same version of the question, they might not even have the same set of answers.

2.3 Generation of a Quiz

Once the Chain structure is parsed, for each quiz generated, a Question is drawn uniformly at random in each Set, and a subset of 4 Answers is chosen in this Question, among which 3 are incorrect.

Example: For instance, choosing uniformly at random one question of the set, one correct answer and three incorrect answers could result in the following quiz question:

1 **Question 1**
Given a Turing Machine M, which of those properties are decidable?

a. M halts on all input ☐

b. M takes less than 76 steps on some input. ☐

c. $L(M)$ is finite ☐

d. $L(M)$ is Regular ☐

Here the first question of the set has been chosen, one correct answer (b) and three incorrect answers. In parallel, the solution of the quiz is generated, which can be made available to students after the exam: these permits to reduce the correction to just marking. An example of such a solution is presented in the appendix.

2.4 Correction of a Quiz

To ease the correction, each quiz is marked with its encrypted solution, a new encryption key being chosen for each exam. A very simple grid encryption was tested and judged sufficient: a grid of random numbers in which are embedded the numbers of the correct answers at positions fixed for the whole exam. Then a simple paper grid permits to check the correct answers of each quiz.

Example: One just cuts holes in the printed grid at the positions indicated by the ones and use it for the whole exam.

ID= 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

In a quiz with the following ID, the correct solutions to each of the four questions are respectively 2,1,3 and 1, i.e. b,a,c and a.

ID= 4 4 1 4 4 2 1 2 4 3 2 1 2 1 3 3 4 2 3 1 3 2 4 1 1 4 3 2 4 1 1 1 3 1 3 4 4 3 3 2

3. A Database of Solved Problems

Multiple Choice Quizzes do not permit to address all the concepts of a course: they permit to test basic knowledge of definitions or the final results of some calculations, but they do not permit to test the ability to write a detailed proof or calculation. To test those abilities, there is no much other options than giving problems in home works, which have to be corrected ``by hand''. Still, some careful organization permits to handle this task efficiently, both for the students and the instructors.

Classically, the assigned problems of a new course are taken from books.

• usually the text of those problems have to be copied again to form assignments, work quite unnecessary when one consider that those problems have been written electronically by a fellow instructor before being printed;

• those books do not contain the solutions to the problems, hence instructors often write their own solutions to the assignments to hand to the students;

• books are static when teaching should be dynamic: some new problems are created every year and are included in books only once in a while.

An organized database of solved problems, accessible only to university instructors, would solve those problems. We tested such a database of solved problems, and describe shortly our implementation and expose our conclusions.

3.1 Description of an Assignment

The amount of work needed to publish an assignment is reduced to the minimum, which is just describing which problems to include, so that the instructors can concentrate on the content of the problems instead of their presentation. An example of the output produced by the 7 lines of code above is given in the appendix.

Example: A file describing an Assignment.

```
\documentclass{assignment}

\begin{document}

\entete{Assignment 8}{Friday, 20th of November}

\problem{ProblemsDB/TM/maxscore_1.tex}{6 marks}
\problem{ProblemsDB/TM/powertwo_1.tex}{6 marks}
\problem{ProblemsDB/TM/undecidable_problems_1.tex}{8 marks}

\end{document}
```

3.2 Description of a Problem and its solution

There can be several informations associated to a problem. Beside its text, it is desirable to specify at least one solution, maybe a name, some information about authorship, and previous usages. As seen on the following example, there can be more than one solution.

Example: A problem.

```
\begin{filename}ProblemsDB/TM/maxscore\_1.tex\end{filename}
\begin{authorship}
Exercise from Computability and Logic, Fourth Edition by George S.
Boolos, John P. Burgess, \& Richard C. Jeffrey, Solution by Albert.
\end{authorship}
\begin{usage} Assignment 8, 2003, CPSC421, University of British Columbia. \end{usage}

Turing machines can be used to calculate functions of nonnegative
integers $f:\mathbb{N}\mapsto\mathbb{N}$.

(...)

\begin{center}Show that $s(k)$ is not a Turing computable function.\end{center}


\begin{solution}
\begin{center}{\bf First solution} (by Albert)\end{center}
We show that $s(k)$ is not Turing computable by diagonalization.

(...)
\end{solution}

\begin{solution}
\begin{center}{\bf Second solution} (by J\'er\'emy)\end{center}

For commodity, let's  note $s(M)$ the score of a TM, and define

(...)

Hence $g(k)=s(k)+1$ is not Turing Computable, and neither is $s(k)$,
as it would be easy otherwise to compute $g(k)$.
\end{solution}
```

3.3 Simple Database

We grouped all problem files in a single folder called ProblemsDB, sorting them by subjects in a second row of folders. This is as simple a database as one can imagine, sufficient for a small number of files, more convenient for a prototype, and especially preferable to convince other instructors to use it as they don't need to learn how to use sophisticated database software. Nevertheless, such a simple structure permits already to store and retrieve enough information, as comments on the problem, authorship, and the previous uses of each problem, using adequate marking and functions.

Example: An index of the database, automatically generated.

```
\documentclass[authorship,usage]{exerciseBook}
\begin{document}
\entete{Term 1, 2003}{UBC}
\section{ProblemsDB}
  \problem{./ProblemsDB/template_problem.tex}{./ProblemsDB/template\_problem.tex}
  \subsection{ArithmeticHierarchy}
    \problem{./ProblemsDB/ArithmeticHierarchy/result_1.tex}{./ProblemsDB/ArithmeticHierarchy/result\_1.tex}
  \subsection{CFG}
    \problem{./ProblemsDB/CFG/CFL_closed_by_regular_intersection_1.tex}{./ProblemsDB/CFG/CFL\_closed\_by\_regular\_intersection\_1.tex}
    \problem{./ProblemsDB/CFG/CKY_1.tex}{./ProblemsDB/CFG/CKY\_1.tex}
    \problem{./ProblemsDB/CFG/CKY_2.tex}{./ProblemsDB/CFG/CKY\_2.tex}
    \problem{./ProblemsDB/CFG/binary_1.tex}{./ProblemsDB/CFG/binary\_1.tex}
    \problem{./ProblemsDB/CFG/cfg from automaton 1.tex}{./ProblemsDB/CFG/cfg\ from\ automaton\ 1.tex}
    \problem{./ProblemsDB/CFG/chomski 1.tex}{./ProblemsDB/CFG/chomski\ 1.tex}
    \problem{./ProblemsDB/CFG/epsilon_1.tex}{./ProblemsDB/CFG/epsilon\_1.tex}
    \problem{./ProblemsDB/CFG/greibach_1.tex}{./ProblemsDB/CFG/greibach\_1.tex}
    \problem{./ProblemsDB/CFG/halloween_1.tex}{./ProblemsDB/CFG/halloween\_1.tex}
    \problem{./ProblemsDB/CFG/irdinana_jones_1.tex}{./ProblemsDB/CFG/irdinana\_jones\_1.tex}
    \problem{./ProblemsDB/CFG/mbaequbb_1.tex}{./ProblemsDB/CFG/mbaequbb\_1.tex}
    \problem{./ProblemsDB/CFG/mbaequbb_2.tex}{./ProblemsDB/CFG/mbaequbb\_2.tex}
    \problem{./ProblemsDB/CFG/parenthesis_1.tex}{./ProblemsDB/CFG/parenthesis\_1.tex}
    \problem{./ProblemsDB/CFG/parser_1.tex}{./ProblemsDB/CFG/parser\_1.tex}
    \problem{./ProblemsDB/CFG/pumping_lemma_1.tex}{./ProblemsDB/CFG/pumping\_lemma\_1.tex}
    \problem{./ProblemsDB/CFG/shuffle_1.tex}{./ProblemsDB/CFG/shuffle\_1.tex}
  \subsection{DFA}
    \problem{./ProblemsDB/DFA/design_1.tex}{./ProblemsDB/DFA/design\_1.tex}
    \problem{./ProblemsDB/DFA/design_2.tex}{./ProblemsDB/DFA/design\_2.tex}
    \problem{./ProblemsDB/DFA/design_3.tex}{./ProblemsDB/DFA/design\_3.tex}
    \problem{./ProblemsDB/DFA/design_4.tex}{./ProblemsDB/DFA/design\_4.tex}
    \problem{./ProblemsDB/DFA/induction_1.tex}{./ProblemsDB/DFA/induction\_1.tex}
    \problem{./ProblemsDB/DFA/induction_2.tex}{./ProblemsDB/DFA/induction\_2.tex}
```

The options solution, filename, authorship and usage trigger the printing of the associated information. Those triggered environments, built on the model of the environment \begin{solution}, permit to build easily an problem book with or without solutions, or an index of the database: it suffices of a Perl or shell script listing the problem files in an assignment file with all the options activated.

3.4 Evaluation

The two projects have been developed in a course in fourth year of university. The result of the

testing of those methods was positive. 57 students, out of the 60 students who took the final exam, filled the anonymous 6-questions survey distributed with the exam text. Among those,

• 30 (53%) rated ``Good'' the usefulness of having a quiz every week, 14 (25%) rated it ``Medium'' and 13 (22%) rated it ``Not Good''.

• 36 (63%) rated ``Good'' the usefulness of having an assignment every week, 19 (33%) rated it ``Medium'' and 2 (4%) rated it ``Not Good'';

And this despite the fact that both systems were developed a bit before and during the term, with the students undergoing the errors subsequent to this kind of development.

Some students, open in their dislike of the quizzes, explained when asked that they resented the constraint of having to come to the course to pass the weekly quiz, when they preferred to follow the textbook at their own pace. None cited the randomness of the quizzes, nor the eventual randomness of the difficulty of the questions, but some cited the annoying mistakes in the database resulting in some questions having more or less than one correct answer. The difficulty of the quizzes is also accountable for their dislike:

• 3 (5%) students rated ``Easy'' the average difficulty of the quizzes, 24 (42%) rated it ``Medium'' and 29 (51%) rated it ``Difficult'', which still sums to 61 as 1 (2%) student refused to comment;

• while 2 (4%) students rated ``Easy'' the average difficulty of the assignments, 28 (49%) rated it ``Medium'' and 27(47%) rated it ``Difficult''.

All in all,

• 12 (21%) students ``Certainly'' enjoyed the course, 22 (39%) ``More or less'' did, 20 (35%) didn't and 3 (5%) refused to comment;

• but 38 (67%) students think they ``Certainly'' learned something in the course, 17 (30%) think they ``Maybe'' did, none think they didn't and 2 (3%) refused to comment;

and I ``Certainly'' enjoyed giving this course, and entertain the impression that I had a good relationship with my students.

Of course a more sophisticate approach would be needed, should such a database of problems and solutions grow larger and be used by several instructors or even several universities. In such a context, there should be a protocol to collectively produce and use such a database. We explore some possibilities in the following sections.

4. Conclusions of the experiments

The following features were identified as undesirable in a software generalizing our prototypes:

- the fixed database structure in the latex database of solved problems was not convenient: different authors have different way to index their document;
- a strong naming policy was instaured for the file included in problems, such as the ones describing figures, but it is quite inconvenient: a universal identifier would be preferable;
- some alternate semi-automatic marking schemes for the quizzes were tested, and rejected.

The main features we chose to keep from the prototypes are:

- a short description of assignment and exams by references to problems;
- the association of a solution for each problem or question;
- the grouping of variants of the same problem, to ease further reuse;
- some alternate ways to describe problems and problem generators, as for the quiz database.

5. Perspectives

5.1 Collaborative Development

The usual digital medias, such as CDs or DVDs, are not adequate to distribute digital content: they generate an over-cost in term of distribution which is easily avoided by digital diffusion, as software, music and video piracy have been proving for several years. This kind of protocol wouldn't make much sense for the distribution of a database of problems and solutions, which is better conceived as a collective work. We discuss a protocol which would insure the growth and sustainability of such a collective work. Selling Services: Coordinating Collective Production and Distribution

One way to publish digital goodies, currently under adoption by the music industry, is to sell them as small services. In the case of music, a restricted access is granted to a large database of music. This idea can be used more generally to any large database, where the price of each query is set so that, to rebuild a reasonable approximation of the original database, one must pay more than the cost of generating the database from scratch. In this case the benefit of the publication is increasing with the success of it, and the distribution costs (including publicity) are paid by the authors who have to maintain or have someone maintaining a server providing the service of distribution.

This model is adequate for the distribution of solved problems from a large database. We propose to adapt it further, incorporating in the protocol the collective construction of the database. For a given course topic, given an initial core of solved problems, let's sell to universities and colleges, for an annual subscription, the rights:

- to access the index of the database, and a short description of the topic of each problem;
- to access to a finite number of checked problems and solutions;
- to submit, for a small fee, a new solved problem P, in a normalized form.

The small fee is needed to pay for the checking of the new problems, so that one cannot submit freely a large number of incorrect problems. In exchange, if P is checked and found acceptable, the giver would win the right to access an additional finite number of checked solved problems from the database.

The subscriptions would fund the maintenance of the database and mainly the checking of new problems and solutions. The database would naturally grow when used, allowing outdated problems to be removed, hence helping to keep the database up to date. Hence the teaching community would work collectively to produce a large database of solved problems, making teaching more efficient.

5.2 XML Integration

The tools exposed in the previous sections are useful, but not generic enough to be usable by a broad community. A language defined through eXtended Markup Language (XML) permits to do so, and once such a general standard is defined, the exchange of documents made easier, a central database of solved problems can be built collectively.

XML seems to be a good choice to integrate the two prototypes previously presented into a single normalized tool which can be used by many. On one hand such a tool must integrate the concepts of a searchable database of solved problems and of Sets of very similar solved problems, which was introduced for the quizzes. On the other hand it must integrate the input and output in various formats, from LaTeX to Word, to adapt to the various needs of the teaching community.

5.3 Generic Parametrized Games

An XML encoding of the text and answers of each problem is more adapted to the publication on the Internet. Problems and quizzes entered in the database can be used indifferently in paper or electronic publications The questions written for quizzes could then be used for other forms of interrogations, inspired by games (for instance, several people can answer the same question, the winner being the first to present a correct answer). Such a form of interrogation should provide more interactivity and could help to produce a better teaching.

We would like to enquire too the possibility to develop generic parameterized small games, which would test the knowledge of some courses. Automatic generation of crosswords and other words games is already been used for the teaching of vocabulary in language courses. We would like to consider a broader class of games, more interactive and using computers. Such games wouldn't be used to test the student as in an exam, but would be made available to students so that they can test themselves.

5.4 Fully Automatic Correction of Quizzes

One very desirable generalization of the project on quizzes would be the mechanization of their correction. There exists already expensive and specialized machines able o mark automatically large number of identical quizzes. There are now some more general machines able to scan and print a large volume of sheets, which would permit to scan and print a large volume of distinct quizzes. The solution is technically feasible: only interest, funding and time are needed.

Acknowledgements: We would like to thank Alejandro Rosete, for helpful discussions about the automatic generation of problems; Joël Friedman, for helpful discussion about the quizzes; Claude Marché, Fabien Torre and Ralf Treinen, for the original model of a database of solved problems; Holger H. Hoos, Kevin Smyth, Jonathan Backer and Albert Wong for helping to fill the database of solved problem; Jason Hartline, for helpful discussion about the distribution protocols of digital goodies. The students of CPSC421 at the University of British Columbia during the year 2003, for being involuntary ``guinea pigs" during a term. This article was written while the author was funded by the University of British Columbia for a postdoctoral fellowship position.

Bibliography

Advanced Educational Technologies--Promise and Puzzlement, Patricia A. Carlson, *Journal of Universal Computer Science, vol. 4, no. 3 (1998), 210-215;*
From the Classroom to the Boardroom, Distance Learning Undergraduate, *International Conference on Engineering Education, August 6 10, 2001, Oslo, Norway;*

- **APPENDIX**

- *Example*: The final output of the solution of a quiz

1 Decidability of TM properties

1.1 Decidability

Given a Turing Machine M, which of those properties are decidable?

$\varepsilon \in L(M)$	**INCORRECT**	*Reduct to HP, build M' st* $L(M') = \Sigma^*$ *or* $\emptyset$.
$L(M) = \emptyset$	**INCORRECT**	*Reduct to HP, build M' st* $L(M') = \Sigma^*$ *or* $\emptyset$.
$L(M) = \Sigma^*$	**INCORRECT**	*Reduct to HP, build M' st* $L(M') = \Sigma^*$ *or* $\emptyset$.
$L(M)$ is finite	**INCORRECT**	*Reduct to HP, build M' st* $L(M') = \Sigma^*$ *or* $\emptyset$.
$L(M)$ is Regular	**INCORRECT**	*Reduce to HP, build M' st* $L(M') = \Pi P$ *or* $\emptyset$.
$L(M)$ is Context Free	**INCORRECT**	*Reduce to HP, build M' st* $L(M') = HP$ *or* $\emptyset$.
$L(M)$ is Recursive	**INCORRECT**	*Reduce to HP, build M' st* $L(M') = HP$ *or* $\emptyset$.
M halts on all input	**INCORRECT**	*Reduce to HP, build M' st* $L(M') = \Pi P$ *or* $\emptyset$.
M takes less than 76 steps on ε.	**CORRECT**	*Simulate M for 76 steps on* ε.
M takes less than 76 steps on some input.	**CORRECT**	*Simulate M for 76 steps on all inputs*
M takes less than 76 steps on all input.	**CORRECT**	*Simulate M for 76 steps on all inputsshorter that*
M takes more than 76 steps on ε.	**CORRECT**	*Simulate M for 76 steps on* ε.
M takes more than 76 steps on some input.	**CORRECT**	*Simulate M for 76 steps on all inputs*
M takes more than 76 steps on all input.	**CORRECT**	*Simulate M for 76 steps on all inputsshorter that*

1.2 UnDecidability

Given a Turing Machine M, which of those properties are *not* decidable?

$\varepsilon \in L(M)$	**CORRECT**	*Reduct to HP, build M' st* $L(M') = \Sigma^*$ *or* $\emptyset$.
$L(M) = \emptyset$	**CORRECT**	*Reduct to HP, build M' st* $L(M') = \Sigma^*$ *or* $\emptyset$.
$L(M) = \Sigma^*$	**CORRECT**	*Reduct to HP, build M' st* $L(M') = \Sigma^*$ *or* $\emptyset$.
$L(M)$ is finite	**CORRECT**	*Reduct to HP, build M' st* $L(M') = \Sigma^*$ *or* $\emptyset$.
$L(M)$ is Regular	**CORRECT**	*Reduce to HP, build M' st* $L(M') = HP$ *or* $\emptyset$.
$L(M)$ is Context Free	**CORRECT**	*Reduce to HP, build M' st* $L(M') = \Pi P$ *or* $\emptyset$.
$L(M)$ is Recursive	**CORRECT**	*Reduce to HP, build M' st* $L(M') = HP$ *or* $\emptyset$.
M halts on all input	**CORRECT**	*Reduce to HP, build M' st* $L(M') = HP$ *or* $\emptyset$.
M takes less than 76 steps on ε.	**INCORRECT**	*Simulate M for 76 steps on* ε.
M takes less than 76 steps on some input.	**INCORRECT**	*Simulate M for 76 steps on all inputs*
M takes less than 76 steps on all input.	**INCORRECT**	*Simulate M for 76 steps on all inputs shorter that*
M takes more than 76 steps on ε.	**INCORRECT**	*Simulate M for 76 steps on* ε.
M takes more than 76 steps on some input.	**INCORRECT**	*Simulate M for 76 steps on all inputs*
M takes more than 76 steps on all input.	**INCORRECT**	*Simulate M for 76 steps on all inputs shorter that*

- *Example:* The final output of an Assignment

-

CPSC 421: Assignment 8 — Friday, 20th of November

This assignment is due on Friday, 20th of November, at the beginning of class.
Late hand-ins will not be accepted.

Problem 1 (6 marks)
Turing machines can be used to calculate functions of nonnegative integers $f : \mathbb{N} \mapsto \mathbb{N}$. That is, given a Turing machine M with input alphabet $\Gamma = \{0, 1\}$ and input x consisting of $k \geq 0$ ones, the associated function $f_M(k)$ for M is

- undefined if the machine does not halt;
- n if the machine halts to the right of n consecutive ones on its tape.

If $f_M(k)$ is defined for all $k \geq 0$, we call f_M a Turing computable function.

Consider Turing Machines with input alphabet Γ and $(k+2)$-states. We simulate each with an input consisting of k ones, and associate to it a score of

- 0 if $f_M(k)$ is undefined;
- $f_M(k)$ otherwise.

Moreover for every integer $k \geq 0$, we define $s(k)$ to be the maximum score that can be achieved by any $(k+2)$-state Turing machine.

Show that $s(k)$ is not a Turing computable function.

Problem 2 (6 marks)
Describe a TM that accepts the set $\{a^n | n \text{ is a power of } 2\}$. Your description should be at the level of the descriptions in Lecture 29 of the TM that accepts $\{ww | w \in \Sigma^*\}$ and the TM that implements the sieve of Eratosthenes. In particular, do not give a list of transitions.

Problem 3 (8 marks)
Prove that the following questions are undecidable:

a. Given a Turing machine M and state q of M, does M ever enter state q on some input? (This problem is analogous to the problem of identifying *dead code*: given a PASCAL/Java/C/C++ program containing a designated block of code, will that block of code ever be executed?)

b. Given two Turing machines M and M', do M and M' accept the same set? (This program is analogous to determining whether two given PASCAL/Java/C/C++ programs are equivalent.)