WHAT: Web-based Haskell Adaptive Tutor

Natalia López, Manuel Núñez, Ismael Rodríguez, and Fernando Rubio

Dept. Sistemas Informáticos y Programación.
Universidad Complutense de Madrid, E-28040 Madrid. Spain.
e-mail:{natalia,mn,isrodrig,fernando}@sip.ucm.es

Abstract. In this paper we introduce WHAT, an intelligent tutor for learning the functional programming language Haskell. WHAT adapts its behavior not only *individually* for each student but also by considering the performance of *similar* students. The core of its adaptive part is based on the classification of students into *classes* (groups of students sharing some attributes). By doing that, the behavior of past students of the same class determines how WHAT interacts, in the future, with students of that class. That is, WHAT *learns* how to deal with each type of student. Besides, the general model of each class is instantiated for each student in order to better fit the particular learning needs.

Keywords: Intelligent Tutors, Education, e-learning.

1 Introduction

During the last years, and due to the wide implantation of the WWW, there has been a proliferation of web-based courses. The first systems were quite rudimentary and rather rigid. By *rigidity* we mean that they were mainly based on a fix set of questions; usually, multiple-choice questions. In particular, it was rather difficult to evaluate the profit gained by the users of the course. The situation changed in the moment that artificial intelligence techniques appeared in the development of this kind of systems. Good examples of these new generation tutoring systems are [3, 1, 12, 6] among many others. Besides, by using adaptive hypermedia techniques (see, for example, [2, 5, 13]) when developing intelligent tutors, we get a system that is automatically adapted according to the responses of students (e.g. [11]).

In this paper we present WHAT:Web-based Haskell Adaptive Tutor. In short, WHAT represents a system which can be used by students as a complement to the classroom learning of the functional programming language Haskell [9]. During the last years, in order to introduce programming to first-year students of Mathematics, different languages (actually, Pascal, Haskell, and Java) have been used in our department. Unfortunately, no common agreement has been reached and the question of which language is more suitable for Mathematics students is still open. The main reason for developing a Haskell tutor, and not a Java or a Pascal one, is that the existing environments for Haskell are not *friendly*. Most Haskell compilers/interpreters only allow the user to interact with a text-based interface and only a programming environment (`winhugs`

at `http://www.haskell.org/hugs`) provides a simple graphical interface. Thus, one of the main disadvantages of using Haskell to teach first-year students is that they do not feel comfortable with the environment: Even though the language could be a good choice, the environments are certainly not. Therefore, in order to fairly compare the difficulties the students have to deal with each language, a better interface is needed for Haskell.

Next we briefly sketch the main characteristics of WHAT. First, WHAT combines individual profiles for each student with general profiles for each *class* of students (the class mechanism is explained in Section 3). By doing so, WHAT is able to adapt its behavior with respect to a student not only *individually*, on the basis of her[1] previous fails and successes, but also by taking into account the performance of the rest of students (current as well as former students). That is, the system *learns* how to interact with a student by adapting the general profile of her class(es) to her necessities. However, the progress of the users while experimenting with the system is individually controlled. WHAT keeps profiles for each student where all the relevant information (from her previous sessions) is recorded. In particular, the system points to the next topic that the user should explore once she has reached a certain command in the current topic. Another important feature of WHAT is given by its skill to (automatically) generate new problems. Exercises are randomly created according to the knowledge that the student has obtained so far.

We think that WHAT may increase the success rate of students in two ways. On the one hand, students can regularly check their progress by self-evaluation. On the other hand, teachers can find out which parts of the course are more difficult for the students (WHAT provides the teacher with a private interface that allows her to access to information about students performance). In the presentation of WHAT we have tried to avoid Haskell details, so that the main ideas (how WHAT has been developed) can be followed even if the reader is not an expert in functional programming. The rest of the paper is organized as follows. In Section 2 we describe the behavior of WHAT from a user point of view. In Section 3 we explain the adaptive behavior of our tutor. We concentrate on the class mechanism underlying WHAT. In Section 4 we present some implementation details. Finally, in Section 5 we present our conclusions and some lines for future work.

2 An Overview of WHAT

In this section we explain the main functionalities of our adaptive tutor. Technicalities about the implementation are presented in following sections. The main aim of our web-based system is to provide students with an *easy-to-use* tool where they can practice the knowledge previously gained in the classroom. So, students will be able to check whether they have fully assimilated the concepts they were supposed to. Even though we have designed our system to help students having another (main) source of learning (i.e. a teacher) the current system

[1] During the rest of the paper we suppose a generic female student.

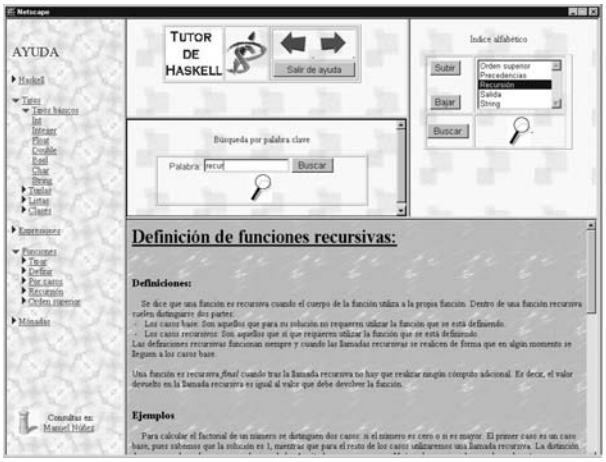

Fig. 1. Help page in WHAT.

could be used as a completely independent tutor. Actually, all the concepts covered by the course are already documented by means of a friendly navigation interface (see Figure 1). Topics can be accessed in three different ways: By using an index of topics (left frame), by incremental searches (center frame), or by an alphabetic list of keywords (right frame).

In order to ensure a personalize treatment, students access the system through a login page. This allows the system to recover the data from previous sessions. At the beginning of the course, students are provided with a password. They log in by giving their ID-numbersand the password. This mechanism tries to avoid *attacks* to previous sessions of students. For example, an attacker could ask the system for previous exercises and provide wrong answers. Then, when the real student logs in, she will find out that the system thinks that she did not understand the concepts covered in previous sessions.

Once a student has been recognized by the system, a new session starts. The initial page allows the student to choose among three different types of problems:

- Evaluation of expressions.
- Typing functions.
- Solving programming assignments.

Let us remark that these three parts are not taught sequentially, but in parallel. For example, students will be able to define a function adding or multiplying two numbers before they are able to type/evaluate an expression as

```
foldr (&&) True (zipWith (==) "hello" ['h','e','l','l','o'])
```

As it is widely recognized, any intelligent tutoring system should have a good underlying curriculum model (see for example [7]). In order to avoid incoherences, the curriculum model should be carefully organized according to some

rules (see [4]). We have designed our system taking into account the following considerations. At each moment, the student will be able to choose which category of exercises she would like to practice. After reaching a certain level in a type of problems, WHAT suggests the student to solve exercises from a different type. However, the student may continue practicing exercises from the same category. In this case, the curricular dependencies restrict the level she can reach in this category. Even though the student can choose freely the type of exercises, the difficulty of them will be constrained by two main factors:

- It will increase according to the level that the student has reached in previous sessions.
- A student will not be allowed to practice topics which have not been covered yet in the classroom, even if she has correctly learnt all of the previous topics.

These two restrictions are strongly related to the curriculum theory underlying our system. Let us remark that even though the first restriction is applied, the system will sometimes ask for exercises which the student is supposed to know already (according to her previous sessions). We pretend to control that students do not forget topics that were covered at the beginning of the course. By using the second restriction we try to avoid that students simply guess the answers to questions that have not been already taught. There is an exception to this constraint. If a student took the course in previous years but she did not pass it, then she will have access to any topic. Let us remark that, even in this case, the first limitation will be applied. Nevertheless, this second restriction could be easily removed in order to allow good students (that is, the ones having an acceptable rate of right answers) to speed up their learning. In this case, the help system (that covers all the course) can (somehow) substitute the teacher.

Finally, WHAT includes an extra option: *Review*. The students will be able to use this possibility to review the lessons learnt so far. So, they will be confronted with assignments similar to those that have been already covered by them in previous sessions. This option can be very interesting (from a student point of view) when the exam is approaching.

Now we will briefly sketch what the student will find in each part of the course. Firstly, in order to gently introduce the language, the capabilities of the programming language will be restricted to the power of a complex calculator. Thus, the first exercises will be only devoted to handle simple numerical operations. So, students will get familiar with the syntax of the language. The difficulty will start increasing, taking into account the precedence of operators, asking for the answer to questions like

```
4-5+7*3-2
```

Both integer and real numbers will be used, but considering the type of operations that can be used with each of them. After dealing with numerical values, other simple types like characters and booleans will be used. Then tuples will be introduced. At this point, the student is able to start working on lists. First, strings of characters will be considered, as they are more intuitive. Then, they will be generalized to deal with lists of any type.

Regarding the definition of functions, at the beginning only very easy operations will be considered (e.g. adding two numbers). By doing so, we expect the students to avoid the usual mistakes of a beginner when declaring functions. Then, the system will concentrate on recursive programming. A next step is given by introducing simple higher order functions like `map`, `filter`, and `zipWith`. Finally, the student will be able to define her own higher order functions.

2.1 Wrong Answers and Hints

A very important part of any tutoring system is the feedback that the user receives. While designing our system, we have been specially careful at this point. Let us consider what our system returns after a question is answered. If the student provides the right answer then the system returns a congratulations message. The difficulties start when managing wrong answers. The easy solution consists in notifying that the answer was wrong and provide the right answer. We consider that this is *not* a good practice. In the best scenario, the student will try to understand what she was doing wrong by pattern matching. We have preferred to return a suggestion about what the student should do (indicating what the error was) instead of giving the right answer. We have also paid special attention to avoid cheating. As it is pointed out for example in [10], some students tend to learn how to cheat the system instead of learning the current contents. We do not claim that our system is totally fool-proof (actually, we do not think so!) but we have tried to detect some *funny* answers. For instance, if we ask for the value of `3+4` a student may answer `5+2` (non so trivial examples include the application of higher order functions in an unexpected way). Actually, this is a right answer, but it is not what it is expected. If WHAT detects such a *right* answer, it will indicate that it is correct but it will ask for the *most correct* answer. Finally, even though the management of answers has been a specially hard part to develop, we think that the effort has been worth. Firstly, students will see their mistakes and try to correct them. Secondly, they will be soon convinced (we hope) that it is senseless to spend time trying to fool the system.

Students will be also allowed to ask for hints. The type of hints, that the system provides, depends on the number of hints the student has already asked for in the current exercise. For instance, if the student has provided a wrong answer, a first hint will only provide a message saying which type of error it was, that is, whether it was a syntactic error, a type error, or a semantic error. Afterwards, in case the student needs more hints, the error will be explained more precisely. Finally, if a student is not able to provide the right answer, she can press the *give up* button and the answer will be presented. In Figure 2(left) we present an example of a question in our system. The student may ask for a hint, give a wrong answer, fail again, etcetera.

3 Adaptive Behavior of the System

The main advantage of using an intelligent tutor with adaptive capabilities is that it can be automatically adapted to the students. WHAT can adapt its

behavior not only on the basis of the fails and successes of the current student, but also on the experiences of the rest of students. Thus, there will be a module of the implementation dealing with the *class adaptive* part, while a different module will be responsible of the *user adaptive* part.

The *Class Adaptive Module* (CAM) gathers statistics about the interaction of the users with WHAT. The ID number of users will not be relevant, but only the *class(es)* of the student. This module manages a fix set of classes, that is, it does not generate new classes. For example, CAM will distinguish between students who were taking the course in previous years and fresh students. By doing so, the system will learn how to interact with the *typical* student of each class, creating different models for different classes. For instance, students with previous knowledge of an imperative language will not need many questions about evaluating simple numerical expressions, while the rest of students will need to work harder on this issue.

It is important to note that a student may be located in more than a single class. For example, a student can take the course for the second year and, in addition, she can also know an imperative programming language. In such cases, it seems that the fails and successes of the student should be taken into account in both of her classes. However, this would be an erroneous solution, as this student is not a good representative of any of them. The correct solution is to use a new class for the intersection of both classes. Note that there is no risk of class explosion, as the set of primitive classes is quite small.

The previous classes are *static*, that is, a student is located in a class(es) at the beginning of the course and these attributes do not change during the academic year. In addition to static classes, WHAT also handles *dynamic* classes of students. Currently, WHAT considers two dynamic classes. The first one classifies the students according to their *learning speed*, while the second one does it on the basis of their *memory*.

It is well known that not all students need the same effort to learn a new topic. Some people need to practice many times the same type of problems before mastering it, while others only need to work on it during a small amount of time. *Fast learning* students use to get bored after solving several similar problems. Thus, by repeating them, their motivation is dramatically reduced. So, they do not achieve the goals they should. On the other hand, *slow learning* students get depressed if they continuously fail the questions. Thus, WHAT will interleave easier questions while asking them new problems. By doing so, the motivation of the students will not be lost and the overall results will be improved.

It is also a fact that not everybody has the same memory. Some students easily forget concepts learnt in previous lessons, while others obtain more solid backgrounds. Students with weaker memories will be asked questions about old lessons more frequently than those with better memories.

Once we have explained the class mechanism, we briefly sketch how CAM adapts itself. The statistics recorded for each class of student and each type of assignments are used by CAM to modify the original information. This is done in two ways. First, for each class of students, CAM keeps values (one for each

topic of the course) about the number of exercises that students are supposed to solve before they know the topic. These values will be readapted according to the performance of students. The second task of CAM consists in classifying the difficulty of programming assignments. Each programming assignment has associated with it a value indicating its difficulty. A difficult programming assignment will not be asked until enough easier exercises are correctly solved. As in the previous case, CAM uses its statistics to modify the difficulty of assignments. The new values are computed by taking into account the response of all the students (regardless of their attributes). However, difficulty is a relative concept. For example, a difficulty value v will not have the same meaning for fresh students that it has for second year students knowing Java. So, the corresponding values v's are normalized for each class of students.

The *User Adaptive Module* (UAM) deals with concrete information about each individual student. As the static attributes of each student will be recorded at the beginning of the academic year, the main tasks of UAM are:

- To compute the dynamic classes in which the student is included at each moment.
- To adapt the models created by CAM to better fit the characteristics of the individual student.

Let us remark that while CAM creates models for each class (both static and dynamic), UAM classifies students inside a class. This is so because only UAM has the appropriate information about individual students. Regarding dynamic classes, WHAT is able to classify students according to both the *speed* and the *memory* of every student. Initially, all the students are located in the same classes. Afterwards, the speed of each student is obtained by computing the number of questions that were necessary before she started to answer correctly all the questions on the same topic. Let us remark that we do not take into account the time of response because this would lead to some erroneous results (for example, a bad Internet connection could indicate that the student is very *slow*). The *memory* factor is computed on the basis of the questions about previously learnt lessons. Besides, some students learn slower at the beginning and faster afterwards, while other students have the opposite behavior. If WHAT detects these situations, it will be the task of UAM to modify the dynamic attributes of the student. Let us note that the modification of dynamic attributes should be done very carefully. For example, a student should not be considered memory-weak just because of a simple mistake, that is, the system is *noise tolerant*.

The second task of UAM consists in adapting the models obtained by CAM. If a student s belongs to a class c, UAM will adapt the general model for c so that it will better fit to the peculiarities of s. For instance, UAM will record errors that the student has performed in the past. When the student has an error and ask for help, the system will be able to relate the current error with previous ones. Then, the student will remember the reason of the past error, and she will better understand the reason of the current one. Finally, let us remark that UAM does not create a complete individual model for s: it only adapts the model corresponding to the class c (designed by CAM). By doing so, we increase the

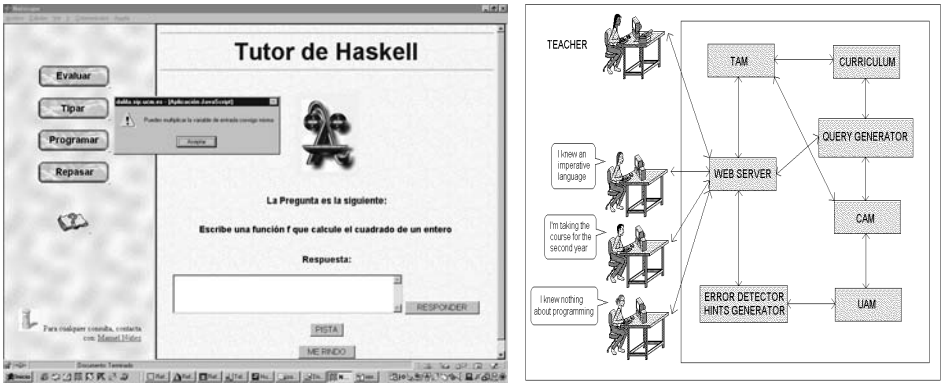

Fig. 2. Exercises page of WHAT (left) and the WHAT system architecture (right).

efficiency of the system and we also provide a hierarchical organization. Thus, it will be easier for the teacher to understand the evolution of the students.

3.1 Non-Adaptive Behavior of WHAT

As we have already explained, WHAT can automatically adapt itself to improve its ability to help students to learn. But the capabilities of an intelligent tutor cannot be compared to those of an expert human teacher. Thus, even though WHAT is a powerful tool to help teaching, we think that the overall control of the course should be taken by the human professor.

From the teacher point of view, WHAT provides information about the skills the students are getting, but also about *how* they are obtaining the skills: Their main difficulties with each type of problems. Thus, the teacher can adapt the classroom lessons to explain with more detail the parts of the course that seemed to be harder for previous students. So, by using WHAT, the teacher learns *from* the students how to improve the lessons.

Not only the classroom lessons can be modified depending on the results provided by WHAT, but also WHAT itself can be modified by the teacher. In fact, WHAT contains an additional module: *Teacher Adaptive Module* (TAM). This module allows the teacher to change the teaching strategies of the system. For instance, it allows to add/modify/remove programming problems from a given lesson. More importantly, it allows to modify by hand both the design of the curriculum and the basic models for each class of students.

4 Implementation

In this section we sketch some implementation details. In particular, we will concentrate on how assignments are generated. First, we present the architecture

of WHAT (see Figure 2, right). From the teacher point of view, she accesses the TAM through the web server. From the TAM the teacher is allowed to modify the curriculum model. Besides, she can also consult CAM to check the current models for the corresponding classes. The web server generates new queries by taking into account both the curriculum model and the models corresponding to the classes of the current student.

The generation of new exercises is automatically done for the first two classes of problems (evaluation and typing of expressions). There is a set of predefined functions, and they are combined randomly, by taking into account that the corresponding types fit. The complexity of the expressions will depend on several factors: the set of types that can be used, the functions that have been taught so far, the number of functions that can be combined in each expression, and also the number of levels of higher-order that can be used, that is, the number of nested higher-order functions that can be used in a single function. The implementation of this generator of problems has been done in Haskell because this language is very suitable for this kind of programs. Let us remark that this choice is not related to the fact that WHAT teaches Haskell. Actually, any programming language could be used to write this part of the system.

Unfortunately, it is not possible to automatically generate programming assignments. So, they are randomly selected from a wide set of predefined problems. The problems are classified according to their characteristics, so that they are only proposed to the student when they have reached the appropriate level. Even though it is not possible to automatically generate new problems, it is possible to automatically generate variations of the predefined problems, modifying the parameters of the problems. Functional languages are very adequate for that because programs can be written by combining higher order functions.

In order to check the programs that students have developed, we provide an automatic connection to a Haskell environment where the program is tested over a set of input data. These data will represent the most relevant cases of the (class of) function, that is, the values that are more difficult to deal with. In addition to the most relevant cases, there will be some randomly selected cases, just to increase the set of tests. By doing so, we will be able to detect most of the errors introduced by the student. Moreover, in the case of a wrong answer, a good hint about the source of error can be given because WHAT will know the cases that are not correctly covered. Let us remind that it is not possible to automatically *verify* that any given program is correct, as it is well known that such a task is undecidable. So, the best that we can do is just to *test* the programs.[2]

Regarding the web interface, the implementation uses both CGIs and JavaScript. The server side of the system is implemented using CGIs. The CGIs (written in C) control the interaction of the user with the system, generating appropriate web pages. These pages include JavaScript code, so that the user may perform most of the operations without actually connecting to the server.

[2] This is an easy way of cheating WHAT. However, we think that the probability of such an error is rather low because of the careful selection of the test cases.

5 Conclusions and Future Work

In this paper we have presented WHAT. We have described its features and we have given some implementation details. In particular, we have explained the class mechanism underlying WHAT. This classification of students allows WHAT to adapt itself to the necessities of each particular type of student.

As future work, we plan to provide similar systems for Pascal and Java. This will allow us to study their learning curves by using comparable tools. Finally, we would also like to apply the experience gained after constructing WHAT to a different field of expertise. Actually, we are already developing a tutor to teach process algebras. In this case, we will consider `PAMR` [8] because it allows to *abstract* some of the usual difficulties when specifying real systems.

References

1. A. Baena, M.V. Belmonte, and L. Mandow. An intelligent tutor for a web-based chess course. In *AH 2000, LNCS 1892*, pages 17–26. Springer, 2000.
2. P. Brusilovsky. Efficient techniques for adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3):87–129, 1996.
3. P. Brusilovsky, E. Schwarz, and G. Weber. ELM-ART: An intelligent tutoring system on the World Wide Web. In *3rd Int. Conf. on Intelligent Tutoring Systems, LNCS 1086*, pages 261–269. Springer, 1996.
4. H. Dufort, E. Aïmeur, C. Frasson, and M. Lalonde. Curriculum evaluation: A case study. In *4th Int. Conf. on Intelligent Tutoring Systems, LNCS 1452*, pages 106–115. Springer, 1998.
5. S. Ferrandino, A. Negro, and V. Scarano. CHEOPS: Adaptive hypermedia on World Wide Web. In *IDMS'97, LNCS 1309*, pages 210–219, 1997.
6. N. López, M. Núñez, I. Rodríguez, and F. Rubio. Including malicious agents into a collaborative learning environment. In *8th Int. Conf. on Intelligent Tutoring Systems, LNCS 2363*, pages 51–60. Springer, 2002.
7. G.I. McCalla. The search for adaptability, flexibility, and individualization: Approaches to curriculum in intelligent tutoring systems. In M. Jones and P.H. Winne, editors, *Foundations and Frontiers of Adaptive Learning Environments*, pages 91–122. Springer, 1992.
8. M. Núñez and I. Rodríguez. `PAMR`: A process algebra for the management of resources in concurrent systems. In *FORTE 2001*, pages 169–185. Kluwer Academic Publishers, 2001.
9. S. L. Peyton Jones and J. Hughes. Report on the programming language Haskell 98, 1999. `http://www.haskell.org`.
10. R. Schank and A. Neaman. Motivation and failure in educational simulation design. In K.D. Forbus and P.J. Feltovich, editors, *Smart Machines in Education*, chapter 2. AAAI Press/The MIT Press, 2001.
11. G. Weber. Adaptive learning systems in the World Wide Web. In *7th Int. Conf. on User Modelling, UM'99*, pages 371–378. Springer, 1999.
12. B.P. Woolf, J. Beck, C. Elliot, and M. Stern. Growth and maturity of intelligent tutoring systems: A status report. In K.D. Forbus and P.J. Feltovich, editors, *Smart Machines in Education*, chapter 4. AAAI Press/The MIT Press, 2001.
13. H. Wu, P. De Bra, A. Aerts, and G.J. Houben. Adaptation control in adaptive hypermedia systems. In *AH 2000, LNCS 1892*, pages 250–259. Springer, 2000.