Wong, L.-H. et al. (Eds.) (2013). Proceedings of the 21st International Conference on Computers in Education. Indonesia: Asia-Pacific Society for Computers in Education

Code Analyser in CSTutor - a C# Intelligent Tutoring System

Budi HARTANTO[a*], **Jim REYE**[b*]
[a, b] *Science and Engineering Faculty, Queensland University of Technology, Australia*
*b1.hartanto@qut.edu.au, j.reye@qut.edu.au

Abstract: This paper describes the process that is performed by CSTutor to analyse each student program. CSTutor is an Intelligent Tutoring System that supports the student's learning by doing. Built as an integrated part of Visual Studio 2010 or 2012, CSTutor can give assistance to a student writing programs in Visual Studio from the earliest stage. The analysis process starts by capturing the student's program from the Visual Studio Editor. The program is then parsed and simplified into facts in a knowledge base. This knowledge base also contains rules, actions, constraints, and a goal to be achieved. The goal can be decomposed into several sub-goals to give a finer detail of feedback to the student. So that it can be used as a practical supplement to classroom instruction, CSTutor provides a number of exercises that can be tried by the students. Further, the number of exercises can be increased without having to change CSTutor's program code. The teacher just needs to add the description of the exercise, the constraints, and the goal that should be achieved in the new exercise. The evaluation of CSTutor is in progress and it is expected that CSTutor will help students learn programming to an improved degree.

Keywords: Program analyser, C#, Intelligent Tutoring System

1. Introduction

Many Intelligent Tutoring Systems (ITSs) have been developed to help students learn programming (Johnson & Soloway, 1985; Sykes, 2007; Weragama & Reye, 2012). However, not many ITSs are really used in classrooms (Pears et al., 2007). Some possible reasons for this are: the number of exercises in the ITS are too few, and the ITSs can only help students with problems that are related to program syntax and not to program logic.

The first reason concerns the number of exercises that are supported by the ITS. Because of the complexity of recognising the students' code, an ITS may only be able to handle such code for a few problems only. For example, in Johnson & Soloway (1985), there are only two unrelated exercises that can be used by the student to practice. When there are only a small number of exercises and more over they are not related to each other, the exercises cannot be used to incrementally build up the student's knowledge about how to write programs.

With regards to the second reason, some ITSs are designed to help the student write a syntactically correct program, rather than a syntactically and logically correct program (For example: Sykes, 2007). Because there are many development environments (nowadays) that can give assistance about the syntax errors that may exist in a program, it would be much better if the ITS can concentrate on giving assistance about any logical problems. The ITS that gives support to syntactic problems only will lose its charm because the student can get similar support from modern development environments, without having to run the ITS.

The goal of this research is to help students learn programming, a subject area that is widely known as difficult (Teague & Roe, 2008). To achieve the goal, we have developed CSTutor, an Intelligent Tutoring System that can tailor its assistance and feedback based on the state of the user's program. CSTutor is expected to be used as a supplement to the classroom instruction. Unlike other ITSs in the programming domain, CSTutor uses natural learning as its teaching method. Schank and Cleary (1995) state that natural learning can improve the learning process. By using natural learning as its teaching method, CSTutor is expected to makes the learning process more enjoyable and effective.

In order to enable CSTutor to be used as a supplement to the classroom instruction, CSTutor should provide a sufficient number of exercises. Where desired, the teacher can add new exercises to CSTutor without having to modify the program code. Currently, CSTutor provides 16 programming problems for the student. These problems cover the topics of assignment statements, conditional statements, repetition using ‘for’ statements, and one dimensional arrays. More programming problems are planned. This paper shows how CSTutor recognizes the student program for a given problem and gives feedback on any student logic errors.

2. Analysing the Student Code

Checking the correctness of a student program can be complex and difficult because the solutions to a given problem can be numerous and varied. In order to check the correctness of the student program as well as to give some feedback and assistance, a knowledge base is employed in the ITS. This knowledge base contains facts, actions, and rules along with the goal for a particular problem.

How can this knowledge base is used to check the student code as well as to give feedback to the student? For example, assume that the student is asked to find the amount of discount from a total of items purchased. In the problem description, it is stated that a 10% discount will be given to the customer when his/her total purchased is greater or equal $100. Otherwise the customer will only get 5% discount. Figure 1 shows parts of some possible solutions of the problem:

Version 1.	**Version 2.**	**Version 3.**
`if (purchase >= 100)`	`if (purchase < 100)`	`if (99 < purchase)`
`  disc = 0.1 * purchase;`	`  disc = 0.05 *`	`  disc = purchase *`
`else`	`purchase;`	`10.0/100;`
`  disc = 0.05 *`	`else if (99 < purchase)`	`else`
`purchase;`	`  disc = purchase * 0.1;`	`  disc = purchase * 0.05;`

Figure 1. Parts of some possible solutions of the problem of calculating a discount

2.1 Setting up the Goal

For the problem described above, we can use the value in variable *disc* as our goal to see if the student program can solve the given problem or not. In this case, the final value of this variable is 0.1**purchase* if the value of variable *purchase* is greater than or equal to 100, and 0.05**purchase* if the value of variable *purchase* is less than 100.

In CSTutor this goal can be represented with a predicate HasVarValue that has three arguments in it. The first argument will refer to the ID of the variable, the second argument is the value of the variable, and the third argument is a condition or list of conditions. Figure 2 shows how the goal of the above problem is defined in CSTutor.

```
Goal:  HasVarValue(varID_purchase, val_purchase)
       HasVarValue(varID_disc, 0.1*purchase,  [GE (val_purchase, 100)])
       HasVarValue(varID_disc, 0.05*purchase, [LT (val_purchase, 100)])
```

Figure 2. The goal to find the discount based on the value of total purchasing

2.2 Parsing the Student Code

The student’s code is parsed and changed to facts and actions. For an *if* statement, CSTutor will try to make explicit of all the implicit conditions and then simplify them. An action will be used to store any assignment statement inside an *if* statement along with the simplified condition to become facts.

For example, let’s see how the condition and the assignment statement in the “else if” statement of Figure 1 - Version 2 be read and stored as facts. Firstly, this condition is made explicit to become:
[(! (purchase < 100) && (99 < purchase)]. This explicit condition is then simplified to become: (99 < purchase). The whole code of Figure 1- Version 2 will be stored in knowledge base as:

```
HasVarValue(varID_disc, 0.05*purchase, [LT(val_purchase, 100)]).
HasVarValue(varID_disc, purchase*0.1 , [LT(99, val_purchase)]).
```

2.3 Checking the Student Code

The last step that is performed is to check if the facts generated from the student code satisfy the goal or not. There are rules in CSTutor that are used to do this task. Some of them are: a rule to check if the arithmetic expression in the "fact" is actually the same as the arithmetic expression specified in the goal or not; a rule to check if the logical expression in the "fact" is actually the same as the logical expression specified in the goal or not; etc.

For example using the first rule, all of the following arithmetic expressions are considered equal: "0.1*purchase", "purchase*10.0/100", "10*purchase/100.0", etc. On the other hand, using the second rule we can check that for an integer variable, the following logical expression are considered equal: "LessThan(val_purchase, 100)", "GreaterThan(100, val_purchase)", "LessEqual(val_purchase, 99)", etc.

By these and some other rules, CSTutor can check if the facts generated from the student code satisfy the goal or not. This information is used to give feedback to the student. If necessary, a sub goal can be defined to give a finer level of feedback.

3. Evaluation Design

At the time of writing, the evaluation of CSTutor is still in progress. The users of this system are the students at QUT who are taking a C# programming course. Ethical problems would have occurred if only some students enrolled in the course were allowed to use the ITS. This meant that it was impossible to have a control group to compare against the students who were using the ITS. Therefore the evaluation of the natural learning ITS will be measured by comparing the students' programming skills before and after they use CSTutor.

Two types of evaluation are being used. One is to measure CSTutor's capability for checking the correctness of the students' code, and the second is to measure CSTutor's performance in helping students learn programming. For the first evaluation, a database that records the student program and the feedback from CSTutor, will be used. Using the database, we can see how many times CSTutor considers a correct program as correct, and an incorrect program as incorrect.

For the second evaluation, a questionnaire will be used along with the database. This questionnaire is designed to obtain student feedback about his/her experiences in interacting with CSTutor and the effectiveness of CSTutor in helping the student to learn programming.

References

Johnson, W. L., & Soloway, E. (1985). PROUST: Knowledge-based program understanding. *Software Engineering, IEEE Transactions on*, (3), 267-275.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. In *ACM SIGCSE Bulletin* (Vol. 39, No. 4, pp. 204-223).

Schank, R.C. & Cleary, C. (1995). *Engines for education*. New Jersey: Lawrence Erlbaum.

Sykes, E. (2007). Developmental process model for the Java intelligent tutoring system. *Journal of Interactive Learning Research*, *18*(3), 399-410.

Teague, D., & Roe, P. (2008). Collaborative learning: towards a solution for novice programmers. In *Proceedings of the tenth conference on Australasian computing education-Volume 78* (pp. 147-153). Australian Computer Society, Inc.

Weragama, D., & Reye, J. (2012). Designing the knowledge base for a PHP tutor. In *Intelligent Tutoring Systems* (pp. 628-629). Springer Berlin Heidelberg.