Using matching for automatic assessment in computer science learning environments

José Paulo Leal and Nelma Moreira

Technical Report Series: DCC-2000-3

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

Universidade do Porto

Rua do Campo Alegre, 823 4150 Porto, Portugal

Tel: +351+2+6078830 – Fax: +351+2+6003654

http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html

Using matching for automatic assessment in computer science learning environments

José Paulo Leal and Nelma Moreira

{zp,nam}@ncc.up.pt
DCC - FC & LIACC, Universidade do Porto
R. do Campo Alegre, 823 4150 Porto – Portugal

May 2000

Abstract

The traditional method of automatically assessing programming exercises in Computer Science uses a black-box approach where a set of test data is inputed to both students and teachers programs and their outputs compared. This approach is useful for grading but inadequate for detecting and correcting students errors. In this paper we present several cases where we were able to develop matching algorithms to compare answers with solutions and pinpoint differences between them. In some cases the matching is based on the actual structure of answers and solutions. In other cases we use execution side-effects to collect a structure that can be compared using a matching algorithm. This approach is currently being used in Ganesh - a web environment for learning Computer Science.

1 Introduction

The web is the natural environment for learning Computer Science (CS) nowadays. As computers are both the subject and the medium, students are naturally inclined to explore all the potential of this form of presenting structured multimedia content.

In any approach used for structuring the multimedia content [Boy97], whether you follow an objectivist or constructivist method, answering exercises is the corner stone of any learning environment. Giving immediate feedback to a student is arguably the greatest advantage of using a computer for delivering instruction in CS, while multimedia features (such sound or video) are seldom relevant. Automatic assessment is particularly important since computer learning environments have also several drawbacks: reading is slower on screen, computers (even notebooks) are not easy to carry and they are less affordable than printed material [Kom96].

Unfortunately, detecting errors and supporting error recovery is not straightforward. Automatic assessment in CS is usually focused in programming languages, using a combination of dynamic and static methods [FW65, JL95, vM94] with emphasis on the first. Most systems rely on a *black box* approach, where a set of test data is inputed to both the students and teacher programs and their outputs compared. Static methods complement grading by checking the readability and maintainability of the source code [ZF91]. These methods are relatively good as far as grading is concerned but do not provide relevant feedback to support the students learning process.

2 Comparing answers and solutions

In our view, answers and solutions of CS exercises are *sentences* of a certain *language*. Hence, automatic assessment consists in comparing the *meaning* of both sentences (answer and solution), measuring (grading) and highlighting (correcting) their differences.

It should be noted that two syntactically different sentences can have the same meaning. So, before comparing answers and solutions, their *semantic* representation must be obtained. Unfortunately, not all semantic representations are equally comparable and so we can divide the languages in two different categories - *comparable languages* and *incomparable languages*.

The following table presents three examples of semantic domains that can be associated with languages used in Computer Science. The second and third rows present examples of answers and solutions that illustrate the difficulties of comparing semantic representations.

	Comparable		**Incomparable**
Semantics	$\mathbb{N}$	$(\mathbb{N}, +, -)$	$\mathbb{N} \to \mathbb{N}$
Answer	$A = 1$	$A = 3 - 2$	$A : n \rightsquigarrow \begin{cases} 1 & \text{if } n = 1 \\ A(n/2) & \text{if } n \text{ even} \\ A(3n+1) & \text{otherwise} \end{cases}$
Solution	$S = 1$	$S = 1$	$S : n \rightsquigarrow 1$
Language	Multiple choice Simple verifications	Database modeling Query languages Finite automata Regular expressions	Programming languages Turing machines

In the first column the semantics of a sentence is represented by a natural number and the comparison is straightforward. This is a possible semantics for multiple choice tests. In the second column the semantic domain is an algebraic structure in the naturals. Elements of this semantic domain can be compared but may require a rewriting to a normal form. In the table, $3 - 2$ and 1 represent the same value, and that can be checked after evaluating the arithmetic expression. The third column is an example of a semantic domain, natural functions, where the equivalence of two elements may not be proved. There is a conjecture [Lag85] that the two functions represented in this column are equivalent and although empirically checked it was not proven and it is possible that no formal proof exists.

For comparable languages it can be checked if the semantics of any two sentences are equivalent. Some languages used in CS, whose operational semantics involve only a finite number of states, fall into this case. For incomparable languages, checking the equivalence of two sentences is impossible in general. Multiple purpose programming languages fall into this case. In the remainder of this section we present some approaches used for comparing sentences of several languages in each category.

2.1 Comparable languages

Some formal models of computation, such as regular expressions and finite automata, are comparable languages, and so are languages such as SQL, used for querying databases, or HTML, used in web page formatting. In these languages the semantics of a sentence can be represented by finite structure, thus the problem of grading and correcting can be solved using *matching* algorithms.

Since they are comparable languages there is a normal form to which the semantics of every sentence in the language can be reduced to. An example is the comparison of expressions involving commutative operators such as those in regular expressions or in WHERE clauses of SQL queries. Another example is the comparison of sets (where the elements order is immaterial); they are usually implemented as lists and so must be sorted before compared[1]. This is the case in SQL when comparing tables in FROM clauses, or in HTML annotations when comparing attribute-value pairs.

More than simply accepting or rejecting an answer, the matching algorithms must pinpoint its differences from the solution. The errors we want to detect are insertions, deletions or translocations of sub-structures. Unfortunately, general algorithms for identifying these differences are too inefficient to be used in interactive assessment. In order to improve the efficiency of the matching algorithm we assume that the wrong or missing sub-structures are not too far away from their correct equivalent in the solution.

[1]Assuming there is an order relationship among set elements

2.2 Incomparable languages

The concept of matching answers and solutions can also be used in assessment of incomparable languages. Although the semantics of a program cannot be completely represented by a structure, the side-effects of its execution are, in general, *matchable structures*. These matchable structures play a major role in grading exercises since they compare answers and solutions on general terms but may require other approaches to complete the grading. In the following paragraphs we briefly present some domains where this approach was used.

In the Unix shell domain the students learn how to manipulate the hierarchical file system using sequences of commands (programs). Most commands have as side-effect a modification of the file system (creating, renaming or destroying files and directories) and a sequence of commands is graded by comparing the structure of the file system with the one generated by the solution program.

In the GUI programming domain, based on the Tk [Ous94] library, we use as matchable structures the widget trees. The matching algorithm uses the geometrical properties of each widget rather than just it structural position to identify the corresponding elements. After identifying two widgets their callbacks are compared using a *black-box* approach.

In the Assembly language domain, based on the *Apoo* [RM98] environment, the memory is used as matchable structure. The main issue is the data transfer between memory cells and registers. The students programs are graded by comparing the memory cells and registers after their execution with those of the expected solution.

3 Conclusions

In this paper we presented several algorithms for grading and correcting exercises in CS learning domains based on the concept of structural matching. These algorithms are currently being used in the Ganesh learning environment [LM98].

4 References

[Boy97] Tom Boyle. *Design for Multimedia Learning.* Prentice Hall, 1997.

[FW65] George E. Forsythe and Niklaus Wirth. Automatic grading programs. *Communication of the ACM*, (8):275–278, 1965.

[JL95] M.S. Joy and M. Luck. On-line submission and testing of programming assignments. *Innovations in Computing Teaching*, pages 97–103, 1995.

[Kom96] Piet A. Kommers. Multimedia environments. In Piet A. Kommers, Scott Grabinger, and Joanna C. Dunlp, editors, *Hypermedia learning environments.* LEA Publishers, 1996.

[Lag85] Jeff Lagarias. The 3x+1 problem and its generalizations. *American Mathematical Monthly*, 92:3 – 23, 1985.

[LM98] José Paulo Leal and Nelma Moreira. Automatic grading of programming exercises. Technical Report DCC-98-4, DCC-FC& LIACC, UP, June 1998.

[Ous94] J.K. Ousterhout. *TCL and the TK Toolkit.* Addison Wesley, Reading, MA, 1994.

[RM98] Rogrio Reis and Nelma Moreira. **Apoo**: an environment for a first course in assembly language programming. Technical Report DCC-98-9, DCC-FC& LIACC, UP, 1998.

[vM94] Urs von Matt. Kassandra: the automatic grading system. *SIGCUE*, (22):26–40, 1994.

[ZF91] Abdullah Mohd Zin and Eric Foxley. Automatic program quality assessement system. In *Proceedings of the IFIP Conference on Software Quality*, Vidyanagar, India, March 1991.