Automated Evaluation of Novice Programs Written in Assembly Language

Hiroyoshi Watanabe, Masayuki Arai and Shigeo Takei

School of Science and Engineering, Teikyo University.
1-1 Toyosatodai Utsunomiya-shi Tochigi, 320-8551 Japan
E-mail: {hiro,arai,takei}@ics.teikyo-u.ac.jp

This paper describes a method of implementing a system which judges whether students' programs are acceptable or not, based on program recognition. The criteria for an acceptable program are that the action and the implementations of the program satisfy requirements of the given problem. The evaluation process of the criteria consists of two sub-processes: first, the action of the student's program is tested with several pairs of data, and then the program is recognized by matching it against programs which were evaluated by the teacher in advance. The experiments with the implemented system demonstrated that the system judged programs properly and reduced teachers' loads of evaluating student programs drastically.

Keywords: **Performance Supporting System, System Design and Development Methodologies, Artificial Intelligence and Networked Learning**

1 Introduction

We consider the main educational objective of programming courses for beginners is to get students to understand important concepts in programming and be able to write programs using the concepts. For this purpose, teachers give various programs, and read their programs to see whether they obtain the concepts or not. However, teachers' loads of evaluation tend to be heavy. Furthermore, students have to wait quite a long time to get teachers' advice, if we intend to respond to them during a class. To avoid these unreasonable affairs, we aim at automating the reading and checking processes of the raw program; so that teachers can concentrate themselves on higher level work. There are several research projects on educational systems which detect bugs and advise about them based on program recognition [1,2,3,5]. Konishi and Itoh[4] indicated that these automated debuggers require a huge amount of knowledge on bugs and it would be difficult to constitute the systems practically. We also consider that the system would become practical if we restrict its roles to mere testing the acceptability and leave the whole judgment on programs to human teachers.

For the target of automated evaluation, we selected a simple assembly language CASL that is adopted in examinations for information-technology engineers certified by the Japanese ministry of international trade and industry.

2 The Evaluation Policy

Teachers examine students' programs on the following criteria:

(a) The action of the program satisfies requirements of the given problem.
(b) The method of the implementations satisfies requirements of the given problem.

When teachers set problems, they have educational intentions about what students should learn, namely concepts, algorithms, instructions and so on. Consider the case that the description of a problem includes a phrase such as “using a stack”. In this case, the teacher intends to make students learn about a stack and he would reject a program implemented without a stack, even if the program’s action satisfies requirements. The criterion (b) states that students' programs should achieve the educational intentions of teachers. The proposed evaluating system should accept programs that satisfy above the two criteria.

The evaluation of the action, i.e., examining the criterion (a), can be implemented by testing the programs with several pairs of sample data and perfectly automated. (Computers achieve better performance on such kinds of work than humans.) To examine the criterion (b), we use programs, called “answer programs”, which were evaluated by human teachers in advance. An answer program consists of a program list, the judgment (*accept* or *reject*) and advice by teachers. When a student’s program matches one of answer programs, the judgment on the answer program is applied to the student’s program and the advice to the student is generated using the advice of the answer program. When a student’s program does not match any answer programs, human teachers evaluate the program.

3 Processes of Program Evaluation

The evaluation process of students’ programs consists of two sub-processes: a process of evaluating the program’s actions and a process of program matching. For the evaluation, teachers need to prepare the problem information and answer programs for each problem. The problem information includes the problem description presented to students, sample data and information (names and explanations) on labels of sample data. The results of the system’s judgments are classified into three types, i.e., *accept*, *reject* and *unjudgeable*.

3.1 Evaluating Program’s Action

Several pairs of sample data are utilized in the process of evaluating the program action. A pair of sample data consists of input and output data: one datum consists of a label name, a size and values (decimal, hexadecimal or character constants). At first the system asks a student some questions to get the information on the correspondence of labels between the student's program and the sample data and then, the system performs the following processes for each pair of sample data:

(1) It rewrites input data of the student's program based on the information of the correspondence of labels.
(2) It assembles and executes the program setting random values at general registers and storage areas, which are not loaded instructions or constants.
(3) It compares output values with sample data.

The program is accepted if it runs properly for all sample data, but it is rejected if there is a pair of sample data for which it does not run properly.

3.2 Program Matching

There are many variations of a program based on the same idea or the same plan. The following causes the variations:

(a) Different registers can be used for the same purpose.
(b) Different names can be given to a label for the same purpose.
(c) Different instructions can be used for the same operations.

(d) Different orders of instructions can be applicable for bringing the same result.

The program matching process should be robust against such differences. However, the different order of the instructions including jump or subroutine call instructions, can cause different execution efficiency. Therefore, if the difference of the instruction order between a student program and an answer program is not trivial, the implementation of these programs should be regarded as different.

3.2.1 Answer Programs

Answer programs are represented in accordance with syntax of CASL except for register numbers and generalized instructions. General registers are represented in the form of GRX*n* that is used as an index register or GRG*n* that is not used as an index register: *n* is a positive integer. Representing registers in such form, answer programs are robust against (a) differences of register numbers.

The same operation that can be implemented by several kinds of instructions is defined as a generalized instruction. For example, because loading a certain value into a general register can be implemented by an instruction LEA (Load Effective Address) or LD (LoaD), a generalized instruction SET is defined as the operation. A generalized instruction is transformed to a specific CASL instruction by a set of production rules, called "matching knowledge", which does not depend on problems. As a result, answer programs are robust against (c) differences of instructions.

3.2.2 Process of Program Matching

The program matching process aims at making consistent correspondences of instructions, labels and registers between an answer program and a student's program. The matching process consists of the following sub-processes:

(1) Candidates of correspondence are generated.

(2) Candidates of instruction correspondences are pruned using the correspondence information of labels and registers and vice versa. If the instruction correspondences can not be decided using only information of labels and registers, they are decided based on the order of instructions. This matching process is robust against (b)differences of labels.

(3) When all instructions of an answer program correspond to instructions of the student's program, the order of corresponding instructions is compared between the two programs. If the difference of the order is trivial, the program matching process succeeds; that is, the answer program matches the student's program.

(4) Removable instructions, called "redundant instructions" are detected to present advice about them.

When a student's program matches one of answer programs, the judgment on the answer program is applied to the student's program.

4 Experiments

The construction of the system implemented based on the described method is shown in Figure 1. The server provides the problem information and the knowledge for the clients. Also, it logs operations on the clients and students who passed the presented problem. The client systems have functions of editing, assembling and simulating programs in addition to evaluating programs. At this stage, the server retains 26 rules of matching knowledge. We had an experimental class for fourteen senior students at our school using the system. First, we showed the students how to use the system then we presented two problems as follows:

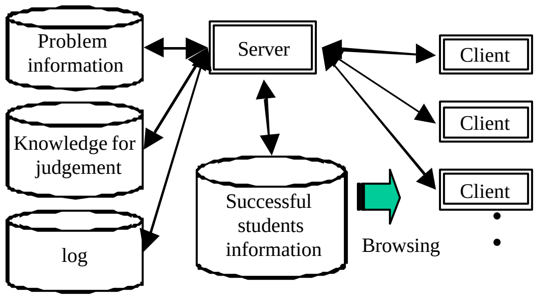

Figure 1 Diagram of system construction

Table 1 Comparisons of the system and a teacher in their judgements

The system	The teacher		Total
	Accept	Reject	
Accept	12	0	12
Unjudgeable	6	0	6
Reject	0	10	10
Total	18	10	28

Problem1: There are two integers in the memory. Write a CASL program which stores the bigger integer of the two in the memory area labeled *MAX*.

Problem2: There are *N* consecutive integers in the memory. Write a CASL program that sums them using index registers and stores the sum in the memory.

We prepared six pairs of sample data and four accepted answer programs for Problem1, two pairs of sample data and three accepted answer programs for Problem2. Five students finished both problems and six students finished either problem1 or 2. Three students who had not taken the assembler course finished neither problem. The comparisons of judgments between the system and the teacher are shown in Table 1. Note the acceptance number in Table 1 differs from the number of students who finished problems described above, because some students wrote two acceptable programs for a problem. Table 1 shows that the judgments of the system were proper. The system judged 22 of 28 programs automatically; so that teachers' load of judging programs was reduced by about 80%.

5 Conclusions

We have described an evaluation method of novice programs written in assembly language. The experiment for simple programming subjects showed that the system implemented based on the method judged acceptability of students' programs properly and reduced teachers' loads of evaluation drastically. Still, it is difficult to prepare all types of answer programs. We plan to improve the system to add new answer programs automatically when teachers judge programs which do not match answer programs. Also, we will evaluate the system more precisely through practical use for actual classes.

This research was supported in part by the Japanese Ministry of Education Grant No.11680400.

References

[1] Adam,A. and Laurent,J.P.,"LAURA, A System to Debug Student Programs, Artificial Intelligence", vol.15, pp.75-122 (1980).

[2] Johnson,W.L., "Understanding and Debugging Novice Programs", Artificial Intelligence, vol.41, pp.51-97 (1990).

[3] Kim,S. and Kim,J.H., "Algorithm Recognition for Programming Tutoring Based on Flow Graph Parsing", Applied Intelligence, vol.6, iss.2, pp. 153-164 (1996).

[4] Konishi,T., Suyama,A. and Itoh,Y., "Evaluation of Novice Programs Based on Teacher's Intention", Proc. of ICCE95, pp.557-566 (1995).

[5] Ueno,H., "Concepts and Methodologies for Knowledge-Based Program Understanding — The ALPUS's Approach", IEICE Trans. Inf. & Syst., vol.E78-D, No.2, pp.1108-1117 (1995).