A Method of Constructing Case-base for Evaluation Assistant of Novice Programs

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

Abstract

This paper presents a method of indexing cases for case-based evaluation assistant systems of novice programs. Program lists in evaluation cases should be represented in intact target programming language, because special forms of program lists put heavy burdens on teachers who are users of the systems. However, intact forms of program lists cannot cover that many variations. Therefore, indexes to cases should be constructed by using information of generalized program lists in order to expand the variations of program lists covered by one case. We propose a three level index of evaluation cases for novice programs written in a simple assembly language. Practical use in actual classes and retrieval experiments demonstrated the effectiveness of the proposed index method.

Keywords: Program evaluation, Programming classes, Supporting teachers and Case-based reasoning.

1 Introduction

In programming education, programming exercise courses play an important role, because writing programs is indispensable to learning programming. However, teachers' workloads tend to be very heavy in typical programming exercise classes. Teachers give students various problems in order to get the students to understand important concepts in programming. Teachers give advice to students and they have to read very many programs and/or reports to see if the students understand the concepts.

There are two approaches to support teachers. The first one is to provide students a diagnostic tool of their programs and students can learn from the output of the tool (Adam 1980, Johnson 1990, Murray 1987, Schorsch 1995, Ueno 1995, Kim 1996 and Xu 1999). Hopefully, the tool reduces teachers' loads of advice giving. Most of the approach aims at automating the whole evaluation work of programs using a knowledge-based technique. The second approach is to support teachers in their evaluation work (Konishi 1995 and Suzuki 2000). We took this latter approach, because supporting teachers with their evaluation work can assist them in efficiently supporting students. That is, reducing teachers' evaluation work gives teachers extra time to advise students.

We implemented a case-based evaluation assistant system of novice programs written in the assembly language CASL and have used the system in actual classes of the CPU and assembly course at our university (Watanabe 1999, 2000 and 2001). The result shows that the system can reduce teachers' evaluation work drastically. At the same time, we found a problem with constructing case-bases. That is, teachers, as users of the system, have to know about the generalized representation of program lists, which we defined in order to enable cases to cover several variations of program lists for the same implementation. The evaluation assistant system of Pascal programs implemented by Konishi and Itoh (Konishi 1995 and Suzuki 2000) also has the same problem. The problem will be quite big for teachers (users of the systems) who are not involved with developing the systems, although it may be a small problem when the developers of the systems use the systems.

In this paper, we describe a method of constructing case-bases for program evaluation systems. Program lists in the cases are represented in intact target programming language so as not to require teachers to know about the special representation of the program lists. Indexes to cases are constructed using information of generalized representation of program lists in order to expand variations of program lists covered by one case and retrieve cases efficiently.

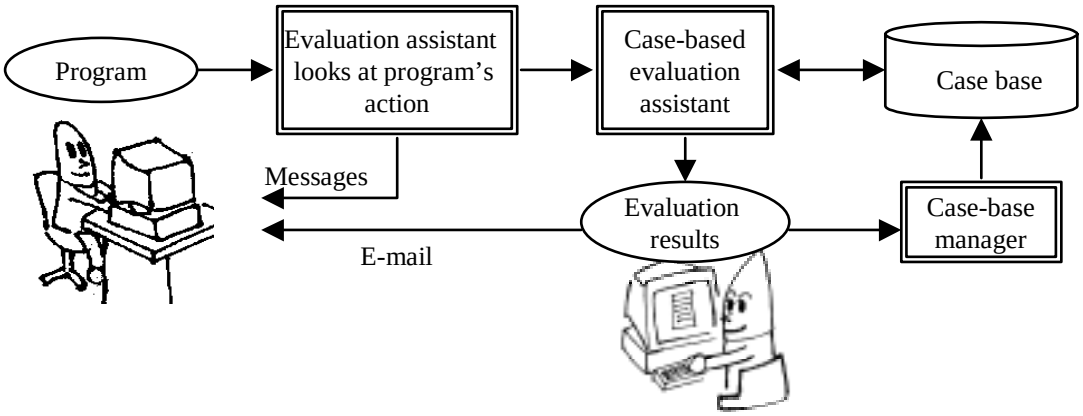


Figure 1: Evaluation processes in the implemented evaluation assistant system.

2 Target Task and Evaluation Assistant

2.1 Task of Program Evaluation

The target evaluation tasks for a student's program are (1) judging the acceptability of the program and (2) advising the student about the program.

The first task is judging whether a student's program 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. Teachers read students' programs to see whether the educational intentions are achieved. Therefore, teachers accept a student's program when the program satisfies requirements of the given problem. The first task is defined on the assumption that students have to submit their programs over and over until their programs are accepted.

When the teacher evaluates a submitted program, he or she judges whether the student's program satisfies requirements of the given problem. Usually, the teacher examines not only the action of the program but also the method of the implementations. Consider the case where 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 or she would reject a program implemented without a stack, even if the program's action satisfies requirements. Also, some teachers may reject overly complicated programs and advise students who wrote the programs to make them simpler.

The second task is giving written advice. Teachers give advice to students whether they accept a program or not: teachers give advice about the reasons why the program is rejected, and advice about bettering the program even if the program is accepted.

2.2 Evaluation Processes with the Assistant System

The evaluation assistant pre-evaluates submitted programs and teachers can edit the results from the evaluation assistant when they evaluate the programs. If the teacher trusts the evaluation assistant, the results from the assistant can be sent to students directly. Such an evaluation assistant is expected to save a teacher a lot of time and energy.

Figure 1 illustrates evaluation processes in the implemented system. First, the actions of a student's program are tested using prepared sample data. The evaluation assistant looking at a program's action rejects programs that do not run correctly. Second, the case-based assistant evaluates the implementations of those correctly executed programs using evaluation cases in the case-base. The output of the case-based assistant consists of evaluation results, the applied case and the degree of

confidence. The evaluation results include the judgment of acceptability (accept or reject) and written advice. The degree of confidence is one of *surely*, *probably* or *unknown*. The teacher edits the evaluation results generated by the case-based assistant and final evaluation results are sent to students by e-mail. The final evaluation results are also used by the case-base manager to add or over-write cases.

2.3 Case-Based Evaluation of Programs

(1) Case Representation. A case for the case-based evaluation assistant consists of a problem description, a solution description, retrieval information and maintenance information. In the domain of a program evaluation task, the problem description is a program list and the solution description is the evaluation result, i.e., the judgment of acceptability and written advice.

(2) Processes of Case-Based Evaluation. Cases in the case-base are retrieved using information generated by analyzing a given student's program. Retrieved cases are evaluated by comparing with the student's program in detail and the "best match" case is selected. When the selected case matches the given program perfectly, *surely* is assigned as the degree of confidence. When the selected case matches the given program, but not perfectly, *probably* is assigned as the degree of confidence. If there is a case that matches the given program, the judgment of acceptability on the case is applied to the given program. In addition, advice sentences on the case are available for the given program. The sentences should be modified for the given program, if needed. The modifications to advice sentences are limited to simple ones only. If no case matches the given program, judgment and advice are not generated and *unknown* is assigned as the degree of confidence.

3 Indexing Cases

3.1 Basic Idea

A program list as a problem description of the case should be represented in intact target programming language. In our first version of the system [1][2], the program lists were represented in the generalized form, in order to enable cases to cover several variations of program lists for the same implementation. However, we found that teachers (users of the system) had to know about the generalized form, because they saw the program lists when the system presented applied cases as the reasons of evaluation results.

Indexes to cases should be constructed by using information of generalized program lists in order to expand variations of program lists covered by one case and achieve efficient case retrieval. There are three types of variations with regard to generalization of program lists, that is, (a) variations of used instructions, (b) variations of instructions' order and (c) variations of redundant instructions.

To deal with (a) variations of used instructions, we define generalized forms of instructions and generalization rules to transform them. Figure 2 shows examples of generalization rules for a generalized instruction SET. The instruction SET is defined as the operation of loading a certain value into a general register and it can be implemented by an instruction LEA (Load Effective Address) or LD (LoaD).

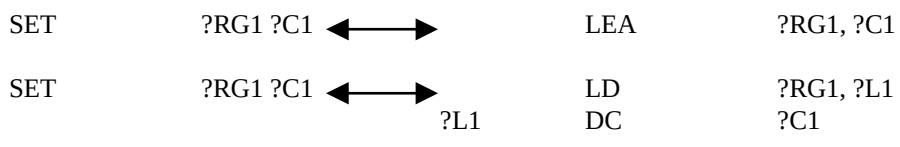
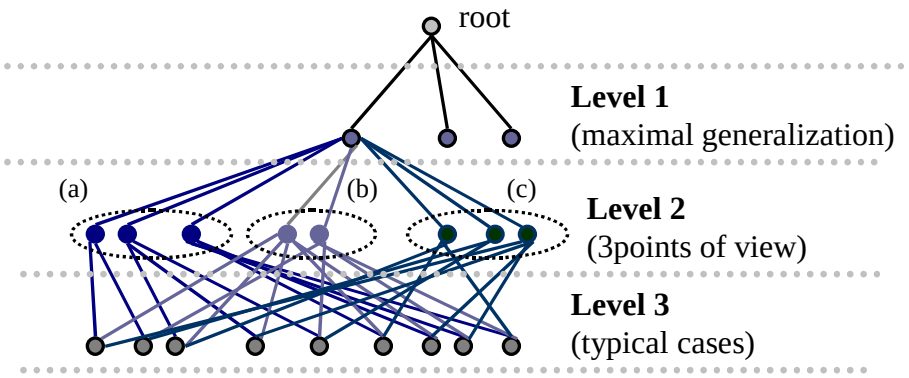


Figure 2: Examples of generalization rules.



(a) Variations of used instructions.
(b) Variations of the instructions' order.
(c) Variations of redundant instructions.

Figure 3: Indexes of cases

Indexes are constructed in three levels based on the degree of generalization. Evaluation cases are placed in the lowest level, i.e., level 3. While index-nodes with maximally generalized program lists are placed in level 1, index-nodes with program lists that are generalized within the range of perfect match are placed in level 2. In case of a perfect match, evaluation results are always the same between matched program lists. Because conditions of such perfect matches depend on teachers' educational intentions, teachers can customize a set of generalization rules for nodes in level 2.

3.2 Method of Indexing

Figure 3 illustrates three level indexes constructed by the proposed method. The indexes enable pruning of cases that have no possibility of matching the given program. More precisely, if the given program does not match some nodes in level one, then descendants of the nodes are pruned. Also, the indexes enable use of a similar case, which is not completely the same. For example, if a given program does not match any cases (nodes in level 3), but matches some node in level 2, then the node's children are available.

Level 1. Nodes in level 1 represent generalized program lists derived by applying all the generalization rules we have. Hence, the nodes are called maximal generalization. The nodes have information about maximal and minimum numbers of every opcode in the program lists, which are covered by the nodes. The information is used for estimating the possibility of matching with the nodes.

Level 2. There are parallel links between level 1 and level 3 taking different routes, (a), (b) or (c) as they go through level2 based on three points of view, so that the priority of the links is defined in case retrieval algorithms. In fact, nodes in level 2 are classified into three groups based on the three types of variations described in Section 3.1:

(a) Variations of used instructions. A node in group (a) represents a generalized program list, which covers some programs of the same implementation. While nodes in level 1 are derived by applying all generalization rules, nodes in group (a) of level 2 are derived by applying a subset of the generalization rules we have. We defined the subset based on our experience and teachers can customize it.

(b) Variations of instructions' order. A node in group (b) covers the same order of instructions. To deal with the variation, the instructions' order of the first seen program list of certain types of programs is used as an index node in level 1 and nodes in group (b) of level 3 are linked from the node.

(c) Variations of redundant instructions. A redundant instruction means a removable one, in other words, an instruction which does not affect the program's action. A node in group (c) covers some cases with the same situations about redundant instructions, that is, the same kind of redundant instructions or no redundant instructions. A program list without redundant instructions is used as an index node in level 1 and nodes in group (c) of level 2 are linked from the node.

Level 3. Nodes in level 3 are typical cases. Programs consisting of the same instructions in the same order are regarded as the same, even if names of labels or numbers of registers are different.

3.3 Processes of Case Retrieval

Retrieval processes consist of (1) generating program features, (2) selecting a node in level 1, (3) selecting the most similar case and (4) generating information about correspondences between the given program and the selected case.

(1) Generating program features, namely, redundant instructions and numbers of opcodes. The system removes instructions one by one and checks out the execution results to detect redundant instructions. An instruction is regarded as redundant, when the execution results are the same regardless of removing the instruction or not. The system also counts numbers of each opcode in the given program, that is, numbers of LD, LEA, ADD, etc.

(2) Selecting a node in level 1. First, the system retrieves nodes, which meet conditions of opcode numbers generated in the first process. Second, the system compares program lists of the retrieved nodes with the given program. This task is called "program matching". Finally, the system selects a node that matches the given program. If no node matches the given program, the case-based evaluation can not produce a decisive result and *unknown* is assigned as the degree of confidence.

(3) Selecting the most similar case. The system investigates nodes in level 2, which are children of the node selected in the second process, and retrieve nodes that match the given program. If there is a case whose parents in all three groups, i.e., group (a), (b) and (c) match the given program, then the case is selected. This situation is called "perfect match" and *surely* is assigned as the degree of confidence. When there is not such a case, *probably* is assigned and the most similar case is selected based on the following criteria.

- A case that has two matching parents is given priority over one that has one matching parent.
- Groups, which case's parents belong to, are given priority in the order of (a) (b) (c).

(4) Generating information about correspondences. The system performs program matching between the given program and the program list in the selected case. If the program matching succeeds, the correspondence information on instructions, labels and registers between the given program and the case is generated. If the program matching does not succeed, the correspondence information is generated by matching the given program against the case's parent node in group (a).

3.4 Processes of Updating the Case-Base

The case-base manager updates the case-base using a pair of a program list and final evaluation results from teachers. First of all, the case-base manager retrieves index-nodes and cases that match the program list, that is, it performs processes of (1), (2) and (3) described in Section 3.3. Suppose that retrieved index nodes are represented as follows: N1 represents a node in level 1, N2a represents a node in group (a) of level 2, N2b represents a node in group (b) of level 2, N2c represents a node in group (c) of level 2 and N3 represents a node in level 3. Note that N3 represents the most similar case.

The case-base manager performs processes of updating the case (N3) and adding index nodes following the retrieval processes of (1), (2) and (3). However, if N3 matches the given program list perfectly and the evaluation results of N3 are the same as the given final evaluation results, these two processes are not performed.

Updating the case. If N3 matches the given program list perfectly and the evaluation results of N3 are different from the given final evaluation results, the evaluation results of N3 are over-written by the given evaluation results. If N3 matches the given program list but not perfectly or N3 is null, i.e., there is no case matches the given program list, the case-base manager adds a new case to the case-base.

Adding index nodes. If some nodes of N1, N2a, N2b and N2c that match the given program list do not exist, the nodes are added to the indexes. The process of adding index nodes consists of the following sub-processes:

- Adding a node in group (a) of level 2. If N2a is null, a new node with a program list generalized by applying the subset of generalization rules to the given program list is added and linked to the existing indexes.
- Adding a node in level 1. If N1 is null, a new node with a program list generalized by applying all generalization rules to the given program list is added and linked to the existing indexes.
- Adding a node in group (b) of level 2. If N2b is null, a new node with information of instructions' order is added and linked to the existing indexes.
- Adding a node in group (c) of level 2. If N2c is null, a new node with information about redundant instructions is added and linked to the existing indexes.

4 Experiments and Discussion

4.1 Practical Use in Classes

The implemented assistant system was successfully utilized for actual classes of the CPU and assembly language course at Teikyo University for two years. We used the first version in 1999 and the second version in 2000. Case-bases of the first version are flat, while three levels indexes are constructed in case-bases of the second version. During the utilization of the systems, 691 programs were submitted for five problems in 1999 and 2227 programs were submitted for twelve problems in 2000. The judgment accuracy, which is defined as the percentage of the same judgment between the case-based system and the teachers, was 99.4% in 1999, and 98.6% in 2000. Figure 4 shows that checking programs' actions reduces target programs for evaluation by about 40 to 50% and the case-based evaluation can reduce the programs by about another 30 to 40%.

4.2 Experiments with Submitted Programs

We made case retrieval experiments with two versions of case-based assistant systems in order to evaluate the proposed index method of cases. For each of the seventeen problems, which we presented in last two years, the following was performed using only correctly executed programs of submitted programs.

(1) The case-base is initialized to null.
(2) For each program, first, the processes of case retrieval are performed. If a case that matches the given program is retrieved, it is counted as a successful match. Second, the case-base is updated with the given program, that is, a new case is added to the case-base unless a perfectly matched case was retrieved.

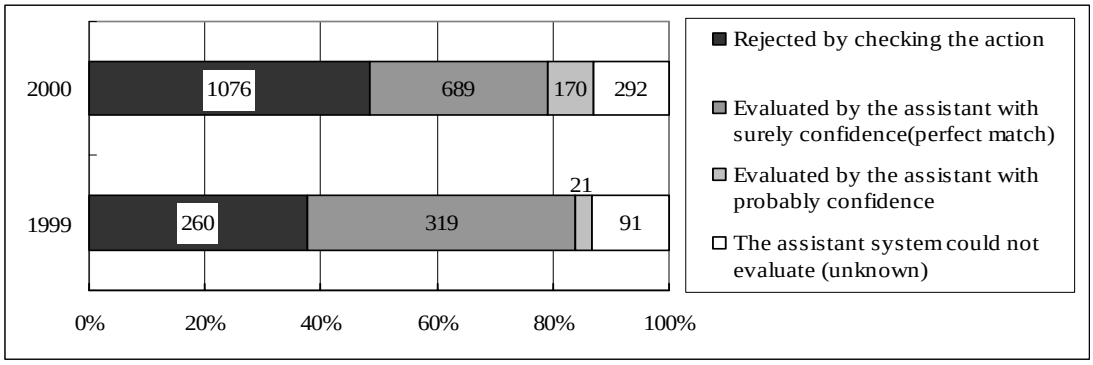


Figure 4: Results of using the implemented systems in 1999 and 2000. Numbers in the graph are numbers of programs.

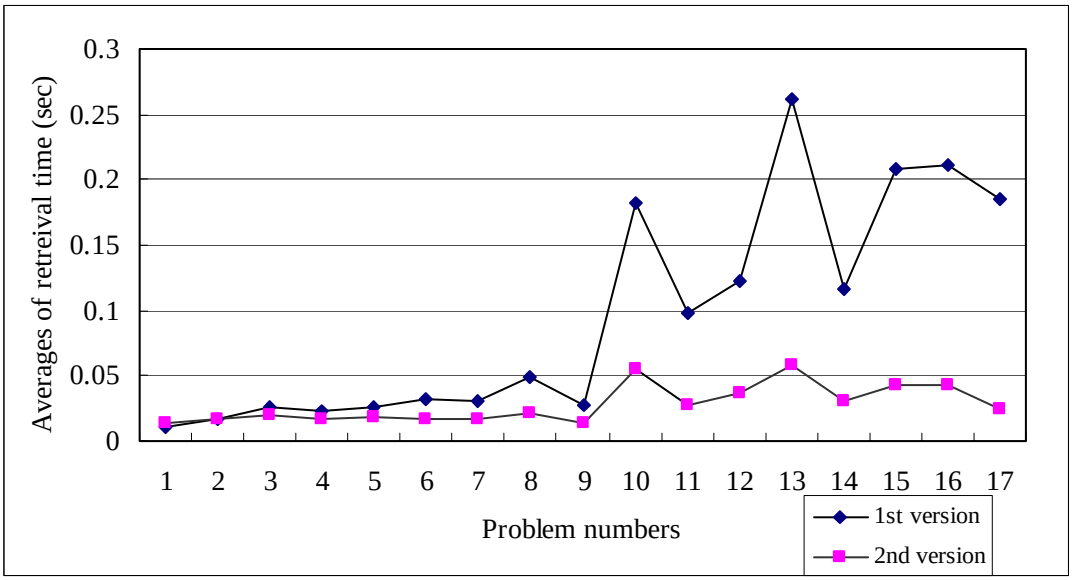


Figure 5: Comparison of case retrieval time. For problems with bigger problem numbers, there are more cases in the case-base.

Table 1: Percentages of successful matches between a given program and a case in the case-base. Asterisks show higher values.

Problem	1	2	3	4	5	6	7	8	9	10
1st version	95.0*	87.3	88.9	89.3	84.8	78.6	77.2	78.1*	83.3	58.9*
2nd version	93.8	88.6*	91.4*	90.7*	88.0*	83.3*	79.7*	71.8	85.1*	55.1
Problem	11	12	13	14	15	16	17			
1st version	67.6	62.5	46.6	61.7	44.6	63.9	44.1			
2nd version	69.6*	63.6*	55.7*	75.2*	50.5*	68.0*	47.1*			

Experimental results demonstrated that the proposed index method of cases is effective in reducing retrieval time and in expanding the applicable range of a case. Figure 5 shows the average time of retrieval processes. Although there is no difference in the time efficiency between the two versions when a case-base is small, the second version retrieves cases more efficiently than the first version. Table 1 shows the second version exceeds the first version on the percentage of successful matches.

4.3 Discussion

Figure 4 shows that both versions of the implemented systems reduces teachers evaluation work drastically. In terms of perfect matches, the first version, i.e. in 1999, achieved a higher ratio than the second version. The reason was that we prepared some initial cases with generalized program lists as examples of correct answers in 1999 while we did not prepare any initial cases in 2000. Therefore, the ratio of perfect matches in Figure 4 did not mean the first version with flat case-bases exceeded the second version.

Table 1 show the proposed method of constructing a case-base is effective to expand the applicable range of cases. Still, with regard to three of the seventeen problems, the first version achieved higher percentages of successful matches, although we expected the second version to exceed the first version for all problems. We analyzed the results on the three problems closely and found a flaw in our way of detecting redundant instructions. That is, there are some cases where either of two instructions is redundant but not both, and our system regards both of the instructions as redundant. As a result, applicable cases are occasionally pruned in the retrieval process.

5 Conclusions

We have proposed a method of constructing case-bases for the case-based evaluation assistant system for novice programs written in a simple assembly language. The assistant system with the case-base construction method was used in actual classes and experiments on case retrieval were performed. As a result, the effectiveness of the proposed method to expand the applicable range of cases and reduce case retrieval time was demonstrated In addition, it has a great advantage for teachers (users of the system), because they do not need to know the general form of program lists, which is special. We plan to improve the index method with regard to detecting redundant instructions in order to expand the applicable range of cases for every situation.

This research was supported in part by the Japanese Ministry of Education Grant No.12780293 and the Foundation for Artificial Intelligence Research Promotion Grant No.12AI320-1.

References

Adam,A. and Laurent,J.P. (1980) “LAURA, A System to Debug Student Programs”, *Artificial Intelligence*, 15, 75-122.

Johnson,W.L. (1990) “Understanding and Debugging Novice Programs”, *Artificial Intelligence*, 41, 51-97.

Murray.W.R. (1987) “Automatic Program Debugging for Intelligent Tutoring Systems”, *Computational Intelligence*, 3, 1-16.

Schorsch,T. (1995) “CAP: An Automated Self-Assessment Tool to Check Pascal Programs for Syntax, Logic and Style Errors”, *Proc. of SIGCSE95*, 168 -172.

Ueno,H. “Concepts and Methodologies for Knowledge-Based Program Understanding - The ALPUS's Approach”, *IEICE Trans. Inf. & Syst.*, E78-D(2), 1108-1117.

Kim,S. and Kim,J.H. (1996) “Algorithm Recognition for Programming Tutoring Based on Flow Graph Parsing”, *Applied Intelligence*, 6(iss.2), 153-164.

Xu,S. and Chee,Y.S. (1999) “Automatic Diagnosis of Student Programs in Programming Learning Environments”, *Proc. of The Sixteenth International Joint Conference on Artificial Intelligence (IJCAI99), Stockholm*, 1102-1107.

Konishi,T., Suyama,A. and Itoh,Y. (1995) “Evaluation of Novice Programs Based on Teacher's Intention”, *Proc. of ICCE95, Singapore*, 557-566.

Suzuki,H., Konishi,T. and Itoh,Y. (2000) “Applicability of an Educational System Assisting Teachers of Novice Programming to Actual Education”, *Proc. of ICCE2000, Taipei*, 128-132.

Watanabe,H., Arai,M. and Takei,S. (1999) “Automated Evaluation of Novice Programs Written in Assembly Language”, *Proc. of ICCE99, Chiba*, 2, 165-168.

Watanabe,H., Arai,M. and Takei,S. (2000) “Case-Based Evaluating Assistant of Novice Programs”, *Proc. of ICCE2000, Taipei*, 1, 133 - 137.

Watanabe,H., Arai,M. and Takei,S. (2001) “Case-Based Evaluation of Novice Programs”, *Proc. of The 10th International Conference on Artificial Intelligence in Education (AI-ED2001), San-Antonio*, 55 - 64.