Program Diagnosis System using World Wide Web

Kenji Kaijiri
Faculty of Engineering, Shinshu University
500 Wakasato Nagano 380-8533 Japan
+81-26-226-4101
kaijiri@cs.shinshu-u.ac.jp

Abstract

We have implemented a program diagnosis system using World Wide Web in order to assist programming education. This system has the following characteristics; 1)We describe the exercise/answer data using html format, 2)Student only need to input selected parts instead of whole programs, so program variation decreases, 3)Students use WWW browser as the GUI, so this system is independent from users' environment.

Subject Category: Learning of Programming, Distance Learning, Program Understanding

1 Introduction

In order to learn programming languages, programming exercises are indispensable. In order to evaluate programming exercises, it is needed to recognize the target programs and to evaluate them as the answer programs for questions, so in order to implement program diagnosis systems, we need program recognition system. There are some program recognition systems for educational use, but their recognition targets are whole programs, so there are the following problems:

- There is no constraints in program construct, so unpredictable variants become possible. This decreases the recognition ratio.
- Students, who do not understand programming well, often construct ill-structured and/or nonsense programs. These programs cannot be recognized using standard program patterns.

In order to cope with these difficulties, answer descriptions become complex, and preparing many exercises also become difficult. So we have developed the program recognition/diagnosis system for educational use which has the following characteristics:

- We consider the programming exercises as the filling problems[2].
  - diagnosing based on the local recognition. We can restrict the range of recognition and specialize the diagnosis.
  - We can give the skeleton of the programs, so students can construct programs focusing the key theme, for example the usage of function. We can make holes from any syntactic unit(identifier, expression, statement, etc), so the filling is not so simple.
  - We can designate the meaningful identifiers.
  - The skeleton can be a hint for students, so the number of ill-structured programs decreases.
- We use the web system. This makes a user interface platform independent and students can use this system from any machine. The hole is implemented using form command in html language, so the GUI design becomes easy.
- We have also developed an exercise editor. This makes easy for teachers to develop exercises for this system.

The program recognition is based on the matching the given standard fragment with the answer fragment using parse tree. We apply some standardization technique to reduce the variation. We also plan to use the plan notation to describe several variations as well.

We will describe the recognition methodology in chapter 2, the system overview in chapter 3, the exercise description in chapter 4, the recognition and diagnosis technique in chapter 5, and the evaluation in chapter 6.

2 Program Recognition

We categorize the program recognition as follows: software recognition from the reverse engineering point of view and the program diagnosis for the educational use. The goal of the former recognition is to recover software architecture and/or design decision, but the goal of the latter is to detect the logical or semantic errors. The objective of the former is programming-in-the-large and that of the latter is programming-in-the-small. We take the latter point of view.

There are several researches about program recognition for program diagnosis [1, 3, 4, 5, 6, 7, 8, 10]. The main problem in these systems is the treatment of variations. There are several kind of variations, for example, design variation, implementation variation, coding variation. In the case of program diagnosis for novice programmers, there are unpredictable variations, so the recognition often fails. For this problem, standardization and plan representation are the usual solutions.

But the construction of the solution programs using plans is not so easy, and plans are in principles well-formed fragment, so the application to ill-formed programs is difficult. These methods take the whole programs as the recognition unit, and this makes variations large.

We resolve this problem using holes that restrict the recognition domain.

3 Program Recognition System

We show the overview of this system in figure 1

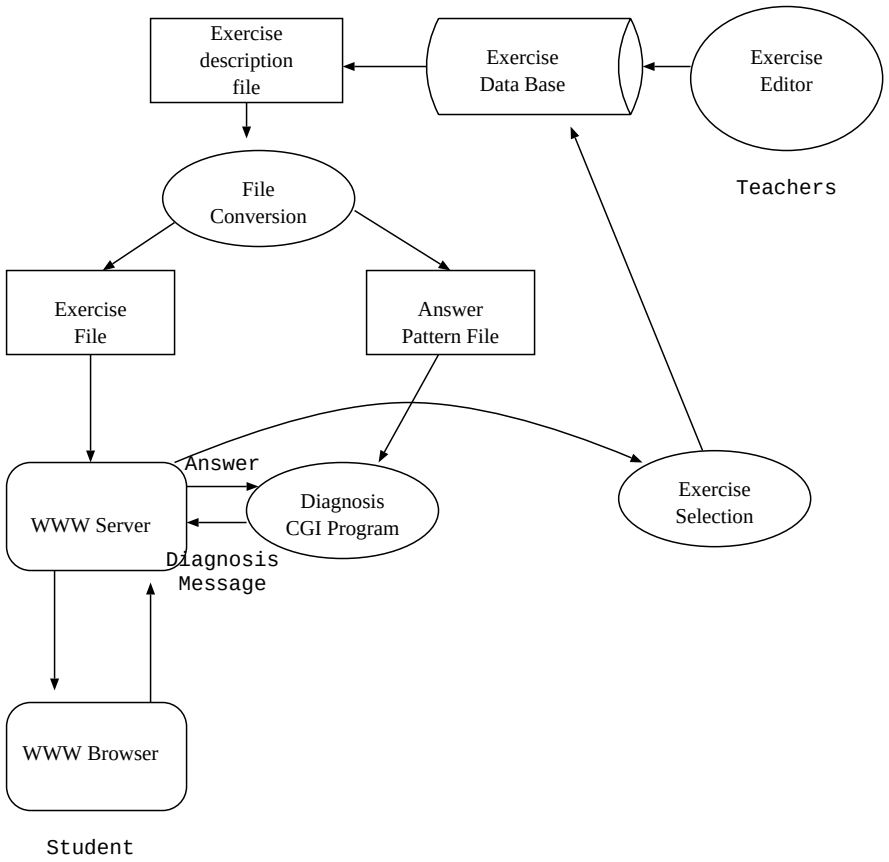


Figure 1: Program Recognition/Diagnosis System

As shown in figure 1, each exercise is stored in an exercise data base. The exercise description is based on html language, and correct and wrong answer fragments are also described in the exercise description.

The whole course materials are described using html. Students select certain exercise from a given set. Each exercise consists of a sample program with some holes, and each hole has its corresponding correct program fragments and wrong program fragments. This description is converted into a html file and an answer file. The html file is used by itself as the web home page file, and the answer file is used to check student programs. Students practice these exercises using some web browsers, and write program fragments for each hole on an

exercise home page. When students select "diagnosis" button, the inputted fragments are passed to Web server and diagnosis cgi program is invoked with these fragments as parameters. The diagnosis cgi program reads the answer file, and compares each answer with the corresponding program fragment.

This system does the following preprocess before matching in order to make correct diagnosis as many as possible:

- to convert an inputted program fragment into standard notation
- to provide several candidate pattern for each hole
- to convert the correct answer into regular wrong answer

Above strategies are not new one, but previous methods take the whole program as matching candidate. Our system restricts the matching candidate area, so the number of variation becomes less than previous methods. In previous methods, there may exist secondary error messages (error diagnosis) which are due to erroneous previous matching. In our system, matching is independent with each other in principles, so there may not exist secondary error messages.

Our system now supports the C language.

4 Exercise Description

The exercise description describes each exercise, which consists of 1)exercise explanation, 2)program for this exercise including some holes, 3)answer program fragment set for each hole, 4)typical illegal answer program fragment set for each hole (optional).

The notation of the exercise description is defined as follows:

```
<ExerciseDescription>::=<ExerciseName>@@<ExerciseExplanation>@@<StandardProgram>
<StandardProgram>::=/* A C program including some holes */
<HoleDescription>::=#<CorrectAnswerList>#<WrongAnswerList>#
<CorrectAnswerList>::=<CorrectAnswer>|<CorrectAnswer><CorrectAnswerList>
<CorrectAnswer>::=@<CorrectAnswerIndex>@<ProgramPattern>
<WrongAnswerList>::=<Empty>|<WrongAnswer>|<WrongAnswer><WrongAnswerList>
<WrongAnswer>::=@<ProgramPattern>@<Message>
```

We describe the answer for an exercise as a C program including some <HoleDescription>s. A <HoleDescription> is recognized as "expression" or "statement" by the parser. A <HoleDescription> is realized using "form" or "textarea" in the html language, and the size (width and height) is determined based on the size of standard correct answer. A <CorrectAnswerIndex> identifies the correct answer in the whole program. If a <CorrectAnswerIndex> has a positive value, then <ProgramPattern>s having the same <CorrectAnswerIndex> are the correct combination. If a <CorrectAnswerIndex> has non-positive value, then the combination of the <ProgramPattern>s for each hole is free. A <Message> is an error message when a <ProgramPattern> for the <WrongAnswer> is matched. <WrongAnswer> is optional, and if <WrongAnswerList> is empty, only wrong answers which are generated automatically are checked.

A <ProgramPattern> is a C program fragment, which must match an answer program fragment. In the description of a <ProgramPattern>, identifiers need not have unique name except for the predetermined identifiers. For this purpose, we provide pattern variables. The form of a pattern variable is "$"+"alphabet character". If the same pattern variable is used in multiple <ProgramPattern>s for different <HoleDescription>s, it means that the identifiers, which matched this pattern variable, must be the same. If the pattern variable is used only in one <HoleDescription>, then the identifier which matches with this pattern variable is free.

We show an example as follows:

```
@@
Read characters (maximum five) from standard input, and write these characters to
standard output. Implement program using one array.
@@
#include <stdio.h>
main()
{
  int i;
  #@0@char $a[6];#char $a[5];@You need the null character at the end of the string
     @char *$a;@ You need to use an array instead of a pointer #
```

```
  scanf("%s", #@0@ $a #@&$a@ Array name can be used as pointer, so & is needless "#);
  #@0@
  i=0;
  do
  {
    printf("%c", $a[i]);
  }while($a[i++] != '\0');
  @0@
  printf("%s", $a);
  ##
}
```

In this example, three holes are given, and for each hole the following correct and wrong answer program fragments are given.

- Form1
  - Correct answer 1 char $a[6];
  - Wrong Answer 1 char $a[5]; "You need the null character at the end of the string"
  - Wrong Answer 2 char *$a; "You need to use an array instead of a pointer"
- Form2
  - Correct Answer 1 $a
  - Wrong Answer 1 &$a "Array name can be used as pointer, so & is needless"
- Form3
  - Correct Answer 1

```
i=0;
do
 {printf("%c", $a[i]);}
while($a[i++] != '\0');
```

  - Correct Answer 2

```
 i=0;
 do
  {printf("%c", $a[i]); i++;}
 while($a[i] != '\0')
```

  - Correct Answer 3

```
   printf("%s", $a);
```

We show the screen image for this exercise in figure 2 and the corresponding html program as follows:

```
<HTML>
<HEAD>
<TITLE>Question</TITLE>
</HEAD>
<BODY>
<H1>Exercise 1</H1>
Read characters (maximum five) from standard input, and write these characters
to standard output. Implement program using one array.
<FROM method=POST action="shindan.cgi">
<TT>
<PRE><H2>
#include <stdio.h>
main()
```

```
{
  int i;
  <INPUT name="string1" SIZE=14>   *1
  scanf("%s", <INPUT name="string2" SIZE=7>;   *2
  <TEXTAREA name="string3" rows=9 cols=31> </TEXTAREA>   *3
}
</PRE>
<INPUT type="submit" value="Diagnosis">
<INPUT type="reset" VALUE="Cancel"><P>
<HR>
</H2>
</TT>
<INPUT name="dummy" TYPE="hidden">
</FORM>
</BODY>
</HTML>
```

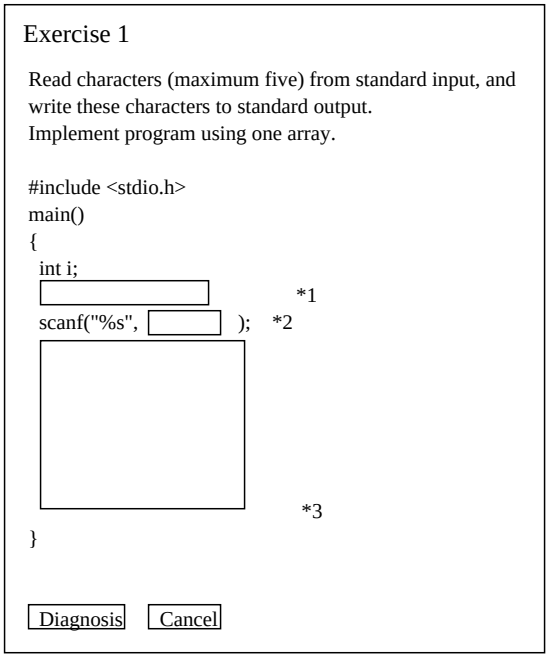


Figure 2: Exercise Home Page

We show the diagnosis example in figure 3.

5 Recognition and Diagnosis Method

The diagnosis will be done hole by hole in principles. If there are positive <CorrectAnswerIndex>s, then all of the program patterns which have the same <CorrectAnswerIndex> must match the same program fragments. We show the diagnosis flow in figure 4.

5.1 Matching

Matching method is parse tree comparison as follows:

```
Match(Answer, Pattern)
If the syntactic category of answer and pattern is the same Then
  Try match for each child
  If all matches succeed, Then succeed, Else fail
Else
```

Diagnosis Message

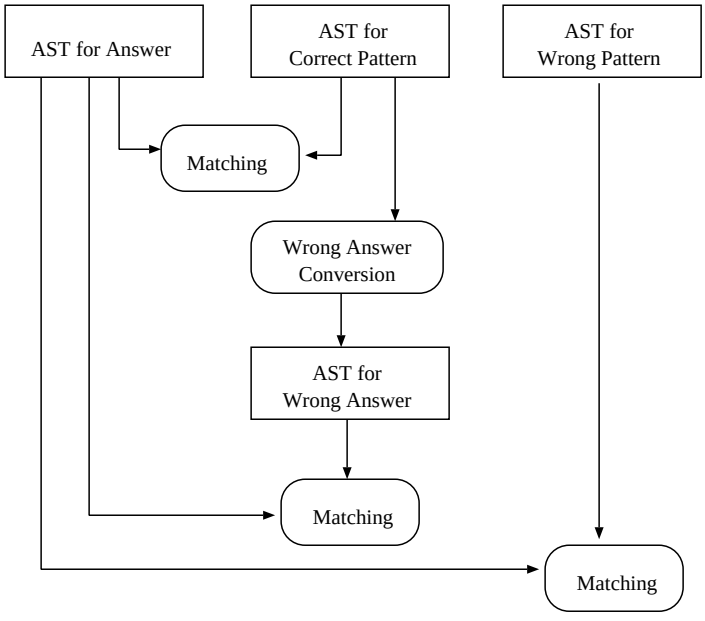
```
#include <stdio.h>
main()
{
  int i;
  char d[5];                          form1

  scanf("%s", &d);                  form2

  i=0;                                  form3
  do
    {printf("%c", d[i++]);}
  while(d[i] != '\0');
}
```

form1 You need the null character at the end of the string

form2 Array name can be used as pointer, so & is needless

form3 right

Figure 3: Diagnosis Home Page



AST means an abstract syntax tree

Figure 4: Diagnosis Flow

```
  Fail
Endif
```

This matching strategy is very simple and is not powerful, so we added the following functionality:

- If the operator is commutative(for example ==), then try the case when a left and a right sibling are exchanged.
- If the operator is binary and is not commutative, then standardize the order of children, for example, use > instead of <.
- If the logical negation operator(!) is used with the relational or logical operators, then expand the negation.

5.2 Error Transformation and Error Matching

Many variations still exist even if we restrict the objective program fragment using holes. We prepare some wrong answer conversion rules in order to automatically generate some wrong answers from a correct answer. We have the following rules:

1. exchange of the similar operators, for example, exchange (==) with (=)
2. misuse of pointer operators (* and &)

These transformations reduce the burden of teachers who develop exercises.
We also provided the erroneous matching rules as follows:

- a constant matches any constant with the same type.
- some expression may be skipped.

6 Consideration

We have implemented the prototype version the www-based program diagnosis system and trial use is possible, but the use in lecture is not possible.

6.1 Exercise Description

Exercise description is cumbersome so we have developed an exercise editor. Using the exercise editor, exercise description becomes very easy and courseware construction by teachers becomes possible.

6.2 Recognition and Diagnosis

Recognition is done based on parse tree comparison and diagnosis message is extracted from appended messages. This makes recognition simple comparing with the former method [8] which uses plan match technique. The following problems are not yet solved.

- Diagnosis messages concerning multiple (or whole) holes.
  Diagnosis messages are prepared for each pattern, and messages for combination error in the case of positive <CorrectAnswerIndex> is not possible.
- difficulty of identification of identifier names.
  If there are many pattern variables for identifiers, there are many variations of matching; for example if there are four pattern variables, there are twenty four cases. To decrease these cases, we predetermine the name of meaningful identifiers.
- we need plan representation in order to put various variation into one description.

6.3 Usage of WWW

The use of WWW system as the infrastructure is good. From the distance education point of view, we want to push forward this approach further [9]. However, WWW has the following defects for our application:

- Input means for program fragment is “form” and “form” has a little freedom and is not sufficient.
- Ordinal web server is state-less. If we want to use student models, the concept of state must be included.

7 Conclusion

We described the program recognition/diagnosis system based on WWW. We use the cloze procedure [2] for the exercise presentation form. This makes the program recognition easy and the number of corresponding program variations small.

WWW is an excellent infrastructure for education, and we plan to construct programming courseware based on this system.

The introduction of plan in exercise description is the powerful solution for the variation problem, but exercise description becomes complex, so we plan to extend an exercise editor, in which teachers can use plans in exercise description.

References

[1] A. Adam, J.Laurent: *Laura, A System to Debug Student Programs*, Artificial Intelligence 15 (1980)

[2] William E. Hall, etl *The Cloze Procedure and Software Comprehensibility Measurement*, IEEE Trans. On S.E., Vol.SE-12, No.5 (May 1986)

[3] W. Lewis Johnson: *Understanding and Debugging Novice Programs* , Artificial Intelligence 45 (1990)

[4] Charles Rich and Richard C. Walters: *The Programmer's Apprentice*, Readings, ACM Press (1990)

[5] Fujii, Watanabe, Tanaka, and Sugie: *An Error Identification Method in the Tutoring System of Pascal Programs*, Trans. On Information Processing Society of Japan, Vol.34, No.3 (March 1993) (in Japanese)

[6] Yu, Ikeda, Saitoh, and Mizoguchi: *Some Consideration about the Knowledge for Program Understanding*, Journal of Japan Society of CAI, Vol.10, No.1 (March 1993) (in Japanese)

[7] Okamoto, Matsuda, and Yasuda: *Study of CAI Algorithm Diagnosis System for Novice C Programmers*, Journal of Japan Society of CAI, Vol.11, No.2 (June 1994) (in Japanese)

[8] K.Kaijiri: *Novice Program Recognition/Diagnosis System Based on Goal/Plan Concepts*, System and Computers in Japan, Vol.27, No.2 (1996)

[9] Murray W. Goldberg: *World Wide Web - Course Tool: An Environment for Building WWW-Based Courses*, 5th International WWW Conference (May 1996)

[10] Hattori and Ishii: *A System to support Evaluation of Student Programs*, Trans. On Japan Society of Information and Systems in Education, Vol.14, No.1 (Apr. 1997) (in Japanese)