Model Based Diagnosis in ITS for Programming with Pedagogical Patterns

Karina Valdivia Delgado

Supervisor: Leliane Nunes de Barros
University of São Paulo, Department of Computer Science,
Rua do Matão, 1010, Bloco C. Cidade Universitária. 05508-090.
São Paulo, SP. Brazil

Abstract. It is not easy for a student to develop programming skills and learn how to construct their own problem solving algorithms. Well designed materials and tools can guide programming students knowledge and skill construction. Such tools may allow students to acquire better and faster the necessary programming skills. In this paper we show the results of some experiments realized on a set of faulty programs using ProPAT Debug, an automatic program debugger, based on the Model Based Diagnosis technique of Artificial Intelligence. The results show that during the interactive debugging process it is possible for a student to learn by answering the questions posed by the AI diagnosis system to discriminate a set of fault hypotheses.

Keywords: Model Based Diagnosis, ITS, Pedagogical Patterns, Debugging Programs. Level: MSc. dissertation. Date of conclusion: 19/12/2005. On-line version of the full dissertation: http://www.ime.usp.br/~kvd/disskvd.pdf

1 Introduction

Model based diagnosis (MBD), also called diagnosis from the first principles, can be described as the interaction between observations and predictions [1]. MBD has been typically applied to troubleshooting physical device. On one hand, we have the actual device whose behavior can be observed, on the other hand we have a model of that device (system description) which is used to make predictions about its behavior. Such model typically describes the components of the system, their connections and the behavior of the components. The difference between an observation and a prediction is called a discrepancy. A match between an observation and a prediction is called a corroboration. Both are used to identify which parts of the device are possibly incorrect [1].

While engineers troubleshoot mechanical or electrical systems to find broken parts by trying to understand the differences between the physical systems and their models, a computer programming teacher tries to understand the differences between the student program code and his intentions (or the problem goals). If we see the student program as a system the teacher wants to troubleshoot, it is interesting to notice that he does not have the program correct
model to reason about, once there is a huge number of different solutions for a single computer problem. Instead, the teacher reasons about program fragment models with the semantics of each expression and sentence being given by the programming language. Further, when the teacher tries to really understand the student program intentions, he also reasons in terms of general and well-known programming strategies, trying to identify them into the student code and to check how well they were instantiated to solve the current problem.

In this paper we show the results of some experiments realized on a set of faulty programs using ProPAT\_deBUG, an automatic program debugger, based on the Model Based Diagnosis technique of Artificial Intelligence. The results show that during the interactive debugging process it is possible for a student to learn by answering the questions posed by the AI diagnosis system to discriminate a set of fault hypotheses. In Section 3 we describe the development of an automatic debugging system, named ProPAT\_deBUG, which is part of a programming learning environment based on elementary patterns, called ProPAT [2]. Section 4 compares the ProPAT system with Intelligent Tutoring Systems for programming and automatic debugging systems. Section 5 gives the background knowledge on Model Based Diagnosis. In Section 6 we discuss how the general theory of MBD can be applied to diagnose student’s programs. In Subsection 6.2, we give some details about the implementation of the MBD diagnosis system for computer programs, named ProPAT\_deBUG. Sections 7 and 8 present the research method and results of some experiments realized on a set of faulty programs using ProPAT\_deBUG. In the Section 9 we draw the final conclusions.

2 Research Motivation and Goals

The automatic diagnosis of student programs in computer based instruction systems for programming is essential. The few existing tools that can detect semantic and logical faults depend on the specification of a correct program and the problem sub-goals. The main difficulties with this approach is that there are many solutions for one same problem and it is difficult determine the independent problem sub-goals. Therefore, the main goals of this dissertation are to:

- Develop a diagnosis system of programs, called ProPAT\_deBUG, to be used in an integrated development environment based on elementary patterns, for a first Computer Programming course.
- Propose a classification for student program faults.
- Evaluate the diagnosis system with a set of student’s programs, including faults classified according to a taxonomy of typical student faults.

3 The ProPAT Plug-in

Research on cognitive theories about programming learning has shown evidences that experienced programmers retrieve old experiences on problem solving that can be applied and adapted to a new problem [3]. On the other hand, a novice
programmer does not have yet any real experiences, but only the primitive structures from the programming language. Inspired on these ideas, a strategy to teach how to program is to present small programming pieces, instead of leaving the student to program from scratch. That is the proposal of the elementary patterns community: a group of experienced educators engaged to recommend programming pieces for novices [4]. Elementary patterns [4] are solutions to common problems. Patterns are available for C, C++ and Java, including: selection patterns [5], repetition patterns [6] and others.

ProPAT is a programming learning environment using elementary patterns, that has been built as an Eclipse plug-in. ProPAT provides an IDE for a first Computer Science course. In this environment, the student can construct a solution by selecting and adding elementary programming patterns into the editor [7]. The ProPAT Eclipse plug-in has two main perspectives: the Student Perspective where the student can choose programming exercises and develop solutions for them, through pattern selections or write his own code; and the Teacher Perspective used by the teacher to specify new exercises and patterns that will be available to the student through the Student Perspective. This project has been developed for the C programming language.

4 Related works

4.1 A MBD for automatic debugging

An original application of Model Based Diagnosis techniques for program debugging was proposed by Franz Wotawa and Markus Stumptner in an Intelligent Debugging System of the Project Jade [8], to help advanced programmers to find bugs. In the program model, language expressions and sentences are represented as components; the information flow is represented as a connection; and the components behavior are described in a formal language based on the programming language semantics. The diagnostic solution is a set of possible faulty components: expressions or sentences of the program code with bugs.

Although the Jade Project [9] has achieved some interesting results, we believe that it can not be successfully applied to students. While in the Jade Project it is expected that a programmer will be able to easily correct his bugs by simple looking at the most probably failing parts of his program, a student will find this task very difficult and will probably not be able to learn from that. A step forward to promote such learning process is to explore the phase where the students makes his predictions about the variable values to be done, in a more comprehensive way. This work makes the assumption that this can be done by the use of knowledge about elementary programming strategies, also called elementary patterns [4], to establish a better communication between the diagnosis system and the student.

4.2 ITS for Programming

ProPAT was first created to be a special programming environment based on elementary patterns and not an intelligent tutoring system. Nevertheless, with the
addition of the model-based diagnosis system, the ProPAT has gain some important characteristics of an intelligent tutoring system. The research presented in this paper is concerned with the local part of learner modeling, that can be divide into three major functions: monitoring (performance assessment), diagnosis (behavior diagnosis) and repair (explanation generation) [10]. In ProPAT the performance assessment is made in two different ways: (1) by monitoring the student insertion of elementary programming patterns into the editor and (2) by running the bench-mark examples looking for incorrect outputs. The diagnosis (behavior diagnosis) is made by the ProPAT DEBUG and the repair (explanation generation) can be made by using the programming patterns documentation.

ProPAT, seen as an ITS, can be compared with PROUST [3] [11]: an ITS for programming learning from the 80s which development was based on the psychology programming theory. PROUST represents an important landmark in the evolution of the programming tutors. PROUST tries to identify the intentions of the student through the matching of the problem sub-goals and plans in the student program. Although the PROUST has been used as the inspiration for many others programming tutoring systems, it has serious limitations that did not let it to be effectively used in programming learning. In the ProPAT system the subgoals problem are not represented and the plan library is replaced by an elementary programming patterns library, that can be accessed directly by both: the student and the teacher. The library is used as a learning material and increasing the probability of the student using the pattern in his program. The patterns can be defined by the teacher, and different to PROUST, they do not need to correspond to all correct forms to solve problems, since we are able to monitor (performance assessment) and observe which elementary pattern the student has inserted in his program. Other works on ITS, such as LISP Tutor [12], are discussed in the MSc Dissertation and omitted here due to space limitations.

5 Model Based Diagnosis Task

5.1 MBD: the conceptual model

Diagnosis reasoning can be conceived as performing three subtasks: (1) symptoms detection, where a symptom is defined as an observation that deviates from expectation;(2) hypotheses generation, where the possible causes, taking into account the initial observation, are generated; and (3) hypotheses discrimination, that discriminates the hypotheses set based on additional observations [1].

- Symptoms Detection. Symptoms are defined as abnormal observations of the system outputs, i.e, an output value that deviates from the expected value. If no symptoms is detected, the diagnosis task will not be performed.

- Hypotheses Generation. This task has two subtasks: find contributors and transform contributors to a hypotheses set.

  - Find Contributors. The contributor set (for the abnormality observations) is a set of components which contains at least one component
incorrect (or faulty). The contributor set is also called a conflict set [13].
A possible approach to perform this task is by simulation. Simulation yields to an expectation (e) for an initial observation (Oinit). If the observation is abnormal, and a set of contributors (c) has been used for deriving the expectation (e), then we say that (c) is the contributor set of the initial abnormal observation (Oinit,ab).

- **Transform Contributors to Hypotheses Set.** The conflict sets are transformed into a hypotheses set. Every hypothesis in the hypotheses set is an explanation of all initial observations [1] and all its elements are supposed to be faulty. This task can be implemented using the Reiter’s Algorithm [14], that constructs the hitting set tree.

- **Hypotheses Discrimination.** For each hypothesis generated, a set of new observations must be made that will be given as the new input to the hypotheses generation task. To decide in which order the hypotheses are going to be tested, it is necessary to have a fault estimate value derived for each component.

### 5.2 MBD: the formal general model

Model Based Diagnosis is formalized as a the theory of reasoning from first principles as follows [14]:

**Definition 1.** [14]: A system is defined to be a pair (SD,COMP), where SD is the system description and COMP is a finite set of constants denoting the collection of components of the system. The system description SD is comprised of a set of first-order logic sentences describing the functionality of the components within the system (behavioral model) and the connections between the components of the system (structural model).

**Definition 2.** [14]: Given an observation, OBS, (SD,COMP,OBS) is a diagnosis problem for the system (SD, COMP) with observation OBS.

**Definition 3.** [14]: A diagnosis \( \Delta \) for the system (SD, COMP) is a minimal subset of COMP such that:
\[
SD \cup OBS \cup \{\neg AB(C) | C \in \Delta \} \cup \{AB(C) | C \in COMP \setminus \Delta \} \text{ is consistent.}
\]
Where \( AB(C) \) means that the component C has an abnormal behavior.

**Definition 4.** [14]: A contributors set for (SD, COMP, OBS) is a set \( CO \subseteq COMP \) such that: \( SD \cup OBS \cup \{\neg AB(C) | C \in CO \} \) is inconsistent.

**Definition 5.** [14]: A hitting set for a collection of sets C is a set \( H \subseteq \bigcup_{S \in C} S \) such that \( \forall S \in C, H \cap S \neq \emptyset \), i.e., a hitting set is a set that intercepts all the sets of the collection C. A minimal hitting set is an hitting set such that none of its subsets is an hitting set. When the collection set C corresponds to the set of all contributors sets CO for a diagnosis problem, the minimal hitting set is the simplest explanation for the observations. The next theorem shows a constructive form to find the hypotheses from the contributor collection set.

**Theorem 6.** [14]: The set \( \Delta \subseteq COMP \) is a diagnosis for (SD,COMP,OBS) if and only if \( \Delta \) is an minimal hitting set for the collection of all contributor sets for the diagnosis problem (SD,COMP,OBS).
6 Model Based Diagnosis of Computer Programs

The basic idea of using the formalization from Section 5 for diagnosing programs, instead of diagnosing physical devices, is to derive a System Description directly from the student program and the programming language semantics. This model must represent components, connections and its behavior based on the actual student program behavior which reflects its errors. The observations are the incorrect outputs in different points of the original program code. The predictions are not made by the system, but by the student and therefore this is the situation where he must communicate his expected values for the variables and be able to understand what are the possible faults in his program. There are two approaches that can be used for program modeling: a value-based model [8] and a dependency based model [15]. They are used to detect functional faults but they are not recommended for detecting structural faults [15]. Functional faults are all faults that result from the storage of an incorrect value of some variable, in at least one possible evaluation trace. Structural faults, on the other hand, are source code bugs, which alter the structure of the underlying program. For example: missing statements. As we show in Section 8 PROPAT_deBUG can detect functional faults and some particular structural faults.

6.1 The Value-based Model

In the PROPAT_deBUG system we have chosen to use the value-based approach [8] to construct the program model SD. Since the value-based model can eliminate wrong diagnosis by using the expected values of variables provided by the programmer, it achieves better results than the dependency based model in most cases [9]. In the value-based model: expressions and statements are represented as components (structural model); the semantics of the expressions and statements are described by logical sentences (behavioral model). Components are connected if there is a flow of information. An information flow between an assignment and another statement occurs, e.g., if the assignment changes the value of a variable that is accessed by other statement, and there is no assignment changing the variable in between [8].

6.2 ProPAT_deBUG System implementation

In this section we specified in more details, how the diagnosis system works and was implemented. The PROPAT_deBUG diagnosis system analysis the student program, after it has been compiled successfully. The diagnosis sub-tasks defined in the conceptual model (Section 5) were implemented as follow:

- **Symptom Detection.** Given a set of test-cases data, represented, for example, as a table of correct input/output values for the problem solution, a symptom is any difference between the program outputs and the outputs of the test-cases. If no symptom is detected, the diagnosis task will not be performed.
– **Hypotheses Generation/Find Contributors.** We implemented this task as a production rule system with a record of dependencies. The rules were constructed such that if a subset of input and output values are known, new values are computed and propagated. The simulation (constraint propagation) is done by disabling the behavior of the abnormal component (i.e., no values are propagated by it). Then, to determine the set of contributors in an inconsistency, a strategy of type ATMS was implemented: while the variable values are propagated through the production rules, we keep track of the components that had derived that value and which input values had been used (justifications). After all the values have been propagated, starting from the last contradiction, a backward search finds all components (the contributors) that helped to derive the contradiction [13].

– **Hypotheses Generation/Transform Contributors to Hypothesis Set.** The conflict sets are transformed into a hypothesis set using the Reiter’s Algorithm [14], i.e., constructing the hitting set tree.

– **Hypotheses Discrimination.** The fault estimate value is derived from test case execution as outlined in [16]. To further guide diagnosis, before we start debugging, we first run all test cases and record which fragments are executed for each test. As every test case is either classified as correct or faulty, we obtain estimate values for each fragment, representing their likelihood to be faulty. A more detailed example can be found in the dissertation documentation in pages 85-97.

7 **Research method and approach**

In order to evaluate the diagnosis system, we have selected a set of programs with typical logical and semantic faults classified according with its consequences and its type. Dominik Wieland [15] proposes a taxonomy of programming faults that we have extended with a decomposition of the semantic faults in two new categories: semantic faults caused by the student wrong interpretation about the individual language constructions and semantic faults caused by the difficulties on the understanding of the interactions between multiple structures.

The goal of the evaluation of diagnosis system is to measure the ability of the system to return the hypotheses set that includes the real program faults without returning too many plausible hypotheses (i.e., hypotheses that can explain the observations but that are not the real faults). For our goals, since the precision and recall measures are usually defined for the information retrieval area, the sets used in these measures has been redefined as follow:

\[ \text{FP} = \{\text{set of faults in the student program}\} \]

\[ \text{HV} = \{\text{set of the k first hypotheses returned by the diagnosis system ordered by a hypotheses estimate value}\} \]

**Precision (PRE)**: is the ratio of the number of correct hypotheses retrieved by the system (\(|FP \cap HV|\)) divided by the total number of hypotheses retrieved by the system.

\[
PRE = \frac{|FP \cap HV|}{|HV|}
\] (1)
By definition, PRE is a measure between 0 and 1: $\text{PRE} = 1$, means that all the hypotheses given by the diagnosis system are real faults; $\text{PRE} = 0$, indicates that the diagnosis system were not able to find any real hypotheses.

**Recall(REC) :** is the number of correct hypotheses retrieved by the system divided by the number of faults of the student program.

$$\text{REC} = \frac{|FP \cap HV|}{|FP|}$$

That is, $0 \leq \text{REC} \leq 1$: $\text{REC} = 1$ means that all faults were considered as hypotheses by the diagnosis system; $\text{REC} = 0$ indicates that the diagnosis system were did not considered any of the real faults as hypotheses. It is clear that what one expect from a diagnosis system is a balance between precision and recall. **F-measure** combines precision and recall and is defined by:

$$\text{F-measure} = \frac{1}{\frac{\alpha}{\text{PRE}} + \frac{1-\alpha}{\text{REC}}}$$

Where $0 < \alpha < 1$ is used as a weight factor to adjust the importance of the precision. The better the F-measure is, the better the precision and recall are.

**8 Results**

The ProPAT_debug system was evaluated for 28 programs with different types of faults. The results indicate an essential aspect of the program diagnosis: it returns a small number of hypotheses about the student fault. The precision, recall and F-measure were calculated for the set $HV$ with the $k$-first hypotheses ordered by the fault estimate value of the hypotheses for $k \in \{1, 2, 3\}$.

![Fig. 1. F-measure, Precision and Recall for $k=3$ and $\alpha = 0.3$.](image)

Using $k = 3$, we observed that the F-measure has shown a better combination between precision and recall. Because these results are not very close to 1, we analyze the importance between precision and recall in the diagnosis, changing the value of $\alpha$. The left side of Fig. 1 shows the values of the F-measure for $k = 3$ and $\alpha = 0.3$, i.e, we give less importance to the precision than recall. The results are that the majority of problems (64 %) has measure-F between 0.63 and 1. It
is important to notice that the recall in the majority of tested programs is big for a little number of hypotheses (Figure 1, right side). The values of F-measure used in the evaluation of the system shown that the diagnosis is more difficult for structural logical (12 out of the 28 programs analyzed) faults in the student program, while the results were close to 1 for faults of type functional logical. That is, for this type of faults, the diagnosis system always returned the real fault as a hypotheses with few (1 or 2) more hypotheses different from the real.

9 Research contributions and limitations

The main contribution of this work is the development and evaluation of the diagnosis system ProPAT:DEBUG. The evaluation are based on the ability of the system to return the hypotheses set that includes the real faults without returning too many plausible hypotheses, i.e., hypotheses that can explain the observations but that are not the real faults.

With this work we conclude that, for detecting student’s functional logical faults, it is possible to have an automatic debugger that returns a small set of possible faults and that always includes the real fault. The most fruitful task for the near future is to run a bigger number of tests and analyze how a group of students, can improve learning using this tool. Another task for the near future is to work on the elementary patterns’ based dialogues, since each hypothesis has to be communicated to the student through elementary patterns templates. Other interesting work for the future is to implement a hierarchical diagnosis system and treat elementary patterns as high-level components.

9.1 Publications of this dissertation

The proposal ideas of this Msc. dissertation were published in the proceedings of Seventh International Conference on Intelligent Tutoring Systems [2], one of the main international conferences of the Intelligent Tutoring area. Although this conference happened in Brazil for the first time, there were few Brazilian papers accepted in the ITS 2004. Another important insight about this Msc. thesis was presented at the Simpósio Brasileiro de Informática na Educação [17], where it is shown how a student solve problems using elementary patterns as a BDI model. In the Eclipse Technology eXchange [7], one of the major workshops at the OOPSLA conference (and the only one to be published at the ACM Library), we have presented the details of the ProPAT plugin construction.

There are also two papers recently accepted at the XVIII Iberamia/Brazilian Artificial Intelligence Symposium [18] and at the XIV Workshop sobre Educação em Computação at the CSBC [19]. These papers give the details about the diagnostic system, the ProPAT:DEBUG, and its evaluation on a set of programs, respectively. Finally, the most important publication of this Msc. dissertation has been accepted to be published at the 3rd Monet Workshop on Model-Based System at ECAI 2006 [20], an event organized by Franz Wotawa, the researcher that has proposed the value-based model used in this work. This paper discusses how the Model based diagnosis of programs has not been extensively explored
by the automatic debugging community and even less for educational purposes. In fact, the main contribution of this work is to propose an original way of using model based diagnosis (and the value-based approach) to support a programming student learning process.

References


This article was processed using the \LaTeX macro package with LLNCS style