Towards an Evaluation Methodology of Diagnostic Accuracy for Ill-defined Domains

Nguyen-Thinh Le and Wolfgang Menzel
Department of Informatics, University of Hamburg, Germany

{le, menzel}@informatik.uni-hamburg.de

Abstract: Researchers agree that error diagnosis is one of the most important components of an Intelligent Tutoring System (ITS). Therefore, the diagnostic accuracy of an ITS within an ill-defined domain should attract attention. In this paper we introduce our constraint-based error diagnosis approach for logic programming and demonstrate an evaluation methodology which measures diagnostic accuracy and is comprised of two parts: evaluation of intention analysis and evaluation of diagnostic reliability.

Keywords: Evaluation, constraint-based diagnosis, ill-defined domains.

Introduction

Researchers agree that error diagnosis is one of the most important components of an Intelligent Tutoring System (ITS). This focus is motivated by the following observations:

1. The diagnosis component provides diagnostic information about student solutions and serves to build an appropriate student model. Especially, in ill-defined domains, the diagnosis of erroneous solutions is uncertain because for an error there might be many conflicting explanations ([1]). For example, the student is requested to write a subgoal to check whether a person is an adult. An erroneous student solution might be `Age>18`. The student has either made an mistake at the operator or the operand (18). If we want to give a feedback to the student, we have to choose between two correction proposals: `Age>=18` or `Age>17`.
2. Subsequent pedagogical interactions, which are dependent on the correct interpretation and diagnosis of student errors ([3]), might mislead students. For example, to compute the result of an investment of amount of money X with an interest rate Y after N years, we can apply the analytic strategy `Result=X*(Y+1)^N` or we can apply the strategy of recursive computation. Even recursive computation can be distinguished between accumulative and naive recursion. If we would not identify the student's intention correctly, all feedback is useless and force the student to follow another strategy which does not agree with his/her intention.

Therefore, the diagnostic accuracy of ITSs of an ill-defined domains should attract more attention than other ITS components as Littman ([6]) predicted: "*It appears that progress in evaluation will go hand in hand with progress in diagnosing students and identifying appropriate student models*".

In this paper we introduce our constraint-based error diagnosis approach for logic programming which has characteristics of an ill-defined domain ([5]). Furthermore, we demonstrate an evaluation methodology which measures diagnostic accuracy and is comprised of two parts: evaluation of intention analysis and evaluation of diagnostic

reliability. In the first section, a coaching system for logic programming (INCOM) is briefly described. Then, we review methodologies for the evaluation of diagnostic accuracy of ITS in the second section. Next, we describe our evaluation and present its result. The advantages and limitations of this evaluation methodology are discussed in the last section.

1. INCOM: a Constraint-based Coaching System for Logic Programming

1.1 Training Scenario

INCOM is intended to help students of a logic programming course overcoming difficulties when doing their homeworks. The system prompts the student with a programming problem and provides feedback to coach her/him composing a correct solution. Given a task, the student is guided to go through two phases: 1) the task analysis and 2) the design and implementation. In the first phase, the student is requested to input an adequate signature for the predicate to be implemented. A predicate signature consists of information about: 1) the number of argument positions, 2) type for each argument position, 3) mode of intended usage of the predicate (input, output or indeterminate). If the signature is not appropriate, INCOM helps her/him to understand the task. This way, the student's ability to analyze a problem is determined. In the second phase, the student is invited to compose a solution for the given exercise in an unrestricted form. The user interface neither requires the student to adhere to an anticipated solution strategy nor does it specify the arrangement of solution elements (i.e. no input templates are used).

The student is allowed to create programs using pure Prolog, a logic programming language. Cuts, disjunctions or if-then-else operators are currently not supported. Similarly, no assert, retract, abolish or other database-altering predicates can be used. The set of built-in predicates which can be employed by the students are: =, =.=, =\=, ==, \==, >, >=, <, =<, =.., +, -, *, /, ^ and 'is'. Helper predicates are provided explicitly in the problem statement or must be defined by the student.

Under these conditions, the student has a free choice of variable and predicate names, can arrange the parameters within a subgoal or a clause head freely and define helper predicates as needed. Due to this degree of freedom, solving a programming task required the student to make numerous decisions to compose a solution. Thus, composing a program freely is an instance of a design problem which is usually considered ill-defined.

1.2 Knowledge Representation and Error Diagnosis

In order to cover the solution space for a logic programming problem under the conditions above, we need techniques to represent the semantics imposed by the problem description without the necessity to enumerate individual solutions. For this purpose, we specify an appropriate predicate signature, develop a so-called semantic table and extend the constraint-based modeling (CBM) approach ([8]) with constraint weights. We clarify our approach by using the problem task *Salary* described in Appendix.

The specification of a predicate signature describes the meaning, type and modus of argument positions which are required in the problem statement. For instance, Table 1 specifies the signature for a predicate *Salary/2*. The information in this table is used for the declaration diagnosis.

Argument	Type	Mode	Description
Arg1	list	input	This argument stands for the old salary list
Arg2	list	output	This argument stands for the new salary list

Table 1: Signature for the predicate Salary

The semantic table contains necessary information for the implementation of a given task. It is specified by means of *generalized* sample solutions which describe the framework of a predicate definition in relational form. That is, clauses, subgoals and argument positions are not restricted to a particular sequential arrangement. In addition, all unification conditions are expressed explicitly and clause heads as well as subgoals are represented in normal form. The normal form representation reveals the underlying programming techniques. Thereby, the diagnosis becomes more adequate on the conceptual level and the resulting feedback becomes more useful. If there are several solution strategies for a problem, each of them is specified by a separate entry in the semantic table. Table 2 specifies the required characteristics of solutions which implement the predicate *Salary/2*. The information in this table is used for the implementation diagnosis.

Strategy	Clause	Head	Body	Description
naive recursion	1	salary(OldL,NewL)	OldL=[] NewL=[]	Old list is empty New list is empty
naive recursion	2	salary(OldL,NewL)	OldL=[N,S|T] NewL=[N,Snew|Tnew] S=<5000 Snew is S+S*0.03 salary(T,Tnew)	N, S: name, salary build a new salary list Salary less than 5000 Salary is increased Decompose old salary list recursively

Table 2: Semantic Table for the predicate Salary

Constraint weights can be conceived of as a measure of importance for a constraint. For example, a clause is composed of a clause head and a set of subgoals, each of which contains a functor and its arguments. A subgoal contributes more information to the overall correctness of the solution compared to an argument or a functor. Hence, a constraint which examines an argument should be specified as being less important compared to a constraint checking a subgoal. We use weighted constraints in order to measure the plausibility of different correction proposals for an error. The reason is, for a programming problem, which has characteristics of ill-definedness, there are many or even uncountably many solutions.

According to the two-phase design of the training scenario, the diagnosis procedure is separated into two steps: declaration and implementation diagnosis. In principle, the diagnosis is carried out as an interaction of hypothesis generation and hypothesis evaluation. Hypotheses are interpretation possibilities for the student solution[1]. Once the student solution is submitted for evaluation, the diagnosis is carried out as follows: 1) Hypothesis generation: structural elements the student solution are mapped to the ones in the semantic table (or the specification of predicate signature) and the created mappings are hypotheses about the intention of the student; 2) Hypothesis evaluation: for each mapping, the plausibility of selected hypotheses is computed based on violated constraints using a multiplicative model. The mapping which has the highest plausibility score represents the best hypothesis. Diagnostic information about shortcomings in the student solution is gathered from constraint violations. Superfluous and missing elements in the student solution are detected based on the hypothesis mapping. A detailed description of INCOM can be found in ([4]).

2. Evaluation Methodology for Diagnostic Accuracy: State of Art

Diagnostic accuracy is a particular performance metrics which is used to explore individual factors or features of an ITS ([1]). According to [6], it falls into the category of internal evaluations because it concerns the inner workings of an ITS, here, the diagnosis component. Diagnostic accuracy is rarely considered in the literature about evaluations of

1 The term “student solution” means both student’s input for signature declaration and implementation.

ITS. Instead, we can find evaluation methodologies which are based on comparing the learning effectiveness between a control and an experimental group or on the difference between the results of a pre- and a post-test. The reason might be that most existing ITSs provide structured interfaces, i.e. solution templates, slots or selection menus for which the evaluation of diagnostic accuracy is not necessary. Using such an environment, the student inputs unambiguous solutions and the student's intention is identified uniquely. If the location of an error is identified, it is evident that the structural element at that location is the source of the error. Therefore, diagnostic accuracy is normally not a crucial issue for ITSs with such structured interfaces. Several constraint-based tutors have been developed using structured interfaces for instance SQLTutor tutors SQL (??), NORMIT for database normalization (??) and KERMIT for database design (??). Although those domains are ill-defined pertaining to tutoring, those systems undertake the design decisions which actually should be make by the student as Kodaganallur and colleagues have commented "*These tutors deliberately reduce task complexity to the point where they are not design tasks anymore.*" (??, p.308).

Yet several ITSs emphasize the diagnostic accuracy in their evaluations, for example PROUST ([2]) which performs intention-based diagnosis of errors in novice PASCAL programs, PITS (??) and Hong's Prolog Tutor (??) which diagnose errors in Prolog programs. These system have been evaluated based on the following measures: 1) the percentage of programs whose solution strategy is identified correctly, 2) the percentage of correctly recognized bugs, and 3) number of false alarms which are bugs detected by the systems but not expected by a human tutor.

3. Evaluation for the Diagnosis Component

The measures described above can be categorized into two groups based on different goals:

1. The evaluation of intention analysis: a student solution is correctly analyzed if the system detects the implemented strategy in that solution and interprets it correctly.
2. The evaluation of diagnostic reliability: a system's diagnosis is reliable if it is assessed to be close to a defined gold standard.

For the evaluation of diagnostic reliability, we will not apply the measures of the second category. Instead, we adopt evaluation measures (Recall, Precision) from the Information Retrieval (IR) field because of two reasons: 1) The measures of the second group reflect the diagnostic reliability on a very low level; and 2) IR and constraint-based diagnosis are selection tasks: find a subset from a set of possible results and the measures (Recall, Precesion) conveys a more general conclusion about the reliability of a diagnosis.

3.1 Evaluation of Intention Analysis

To conduct the evaluation of intention analysis we selected appropriate exercises and solutions from past written examinations. The participants were students who had chosen their major in different branches of Informatics. The examination candidates had attended a course in logic programming which was offered as a part of the first semester curriculum in Informatics. This evaluation has been described in [5] and it is reported that INCOM is able to analyze 87.9% (sd=17.1%) of 221 collected student solutions correctly.

3.2 Evaluation of Diagnostic Reliability

3.2.1 Recall and Precision

According to [9] Recall and Precision are defined with respect to Table 3 as follows:

- $\text{Recall} = \frac{A}{A+C}$ and $\text{Precision} = \frac{A}{A+B}$

	Should-be bugs	**Should-not bugs**
Retrieved bugs	A	B
Not-retrieved bugs	C	

Table 3: Categories for Precision and Recall

Should-be and *should-not* bugs are derived from relevant constraints which should be violated and should not, respectively. Since a constraint can be relevant and applied to different structural elements of a solution, several bugs might stem from the same constraint. Whereas *should-be* and *should-not* bugs are determined by the human judge which is called a gold-standard, *retrieved* and *not-retrieved* bugs are decided by the system's diagnosis. The measures introduced above can be understood as follows:

- A high precision means that the model is based on fairly reliable constraints, which have a low risk of producing false alarms, i.e. the developer was careful to avoid risky constraints.
- A high recall means that the diagnosis considers a broad set of relevant constraints.

3.2.2 Gold Standard

In order to apply the measures introduced in 3.2.1, it is necessary to develop a gold standard. For our system, the gold standard is specified in two parts: a list of *should-be* bugs and a list of *should-not* bugs for a given student solution.

For this purpose, we requested a Prolog expert. His task was to check every bug returned by INCOM for a given student solution. In order to facilitate his work, we developed an assessment tool (Figure 1). For each student solution, the expert is shown the exercise description, a manually added hypothesis about the predicate declaration used by the student, and a list of bugs which is the result of the system's diagnosis. The expert assesses the list of bugs iteratively by choosing the options *OK* or *NotOK* if he thought that a bug is appropriate or not, respectively. In addition, the expert had the possibility to write his comment into the text field associated to each bug. At the bottom of the interface, the expert had the option to add general comments which are not specific to the presented bugs, for example, he/she thought that crucial bugs have been missed.

The gold standard, that means the amount of *should-be* and *should-not* bugs, is established based on two sources of information: 1) the system's diagnosis and 2) the information extracted from the expert's assessment. The former amount consists of bugs approved by the expert together with the bugs indicated as missing. The latter amount is the difference between the number of relevant constraints for the solution being investigated and the amount of *should-be* bugs.

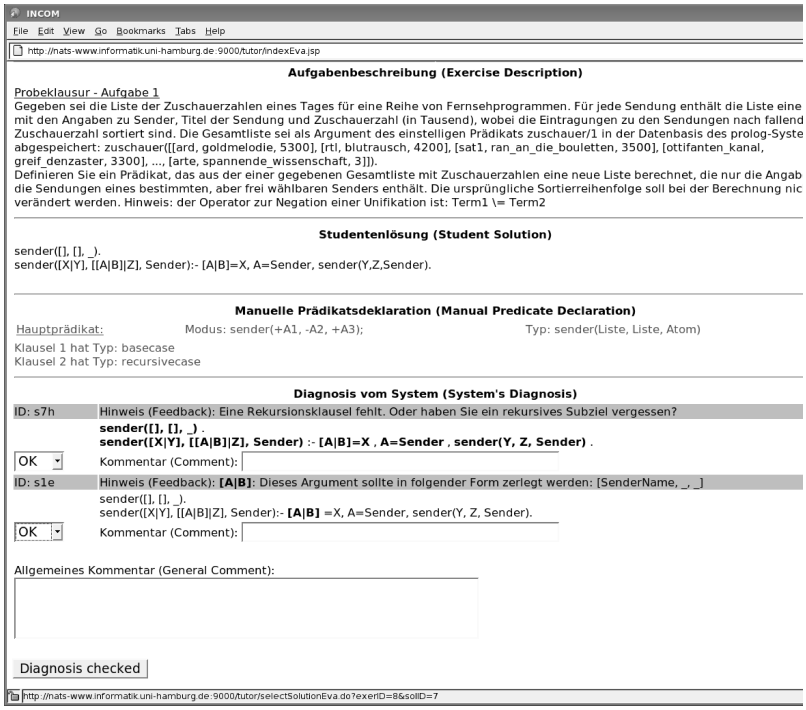

Figure 1: A tool using to assess the system's diagnosis

3.2.3 Evaluation Results

For the evaluation of diagnostic reliability we applied the diagnosis to student solutions which have been classified as *understandable* during the evaluation of intention analysis. The result of the system's diagnosis was then forwarded to the assessment tool of our Prolog expert. After a gold standard has been specified, Precision and Recall are calculated for each student solution according to the formulas introduced above. Table 2 summarizes the results for all *understandable* student solutions to each problem task.

The result of our evaluation of diagnostic reliability shows that the Recall measure is high. It ranges between 0.9010 and 1.0000. It indicates that the system's diagnosis takes a wide range of relevant constraints into account. We notice that the Precision is always lower than the Recall for each task. That means the diagnosis emphasizes the importance more on the quantity than the quality. The Task 1 has the lowest Precision (0.8426) which alerts us that the system has some weaknesses at diagnosing solutions for this problem task. Our expert, who has assessed the system's diagnosis, found out that if a solution is erroneous because of a swapped argument position. For that problem, the system detects many bugs which are correct, however, not necessary. Instead, a bug considering argument position is enough, according to our expert. It seems that the precision of system's diagnosis is too high, (0.9266 in overall). This might be caused by the method we used to determine the gold standard which is strongly biased by the result of the system's diagnosis. The underlying data of our evaluation are available on the homepage of our project (*https://nats-www.informatik.uni-hamburg.de/view/INCOM/Dokumentation*)

Task	Recall	Precision
1	0.9479	0.8426
2	1.0000	0.8750
3	1.0000	1.0000

4	0.9010	0.8906
5	1.0000	0.9737
6	0.9811	0.9528
7	0.9517	0.9517
Overall	0.9688	0.9266

Table 4: Evaluation of diagnostic reliability

4. Discussion

We compare the systems PROUST, PITS and Hong's Prolog tutor with respect to the intention analysis and diagnostic reliability because they provide problem tasks which are similar difficult as ours. In the following we demonstrate how Recall and Precision of PROUST is calculated. The results for systems are calculated similarly.

	Fully analyzed	Partially analyzed	Categories for Recall & Precision
Bugs recognized	562	61	A=562+61=623
Bugs not recognized	36	106	C=36+106=142
False alarms	66	20	B=66+20=86

Table 5: A part of evaluation statistics of PROUST on the Rainfall problem

The categories for Recall and Precision are retrieved by summing up the results of completely and partially analyzed programs as shown in Table 3.

- $\text{Recall} = \frac{A}{A+C} = \frac{623}{623+142} = 0.81$ and $\text{Precision} = \frac{A}{A+B} = \frac{623}{623+86} = 0.88$

System	Intention Analysis	Precision	Recall
INCOM	87.9%	0.93	0.97
PROUST	96%	0.88	0.81

The comparison is conducted based on reported evaluation statistics reported in the corresponding literature and under the assumption that the gold standard of those systems has been specified comparatively as ours.

5. Conclusions and Future Works

The combination of the evaluation of analysis ability and diagnostic reliability to assess the diagnostic accuracy has the following advantages:

1. From the perspective of a system developer, the two steps evaluation of diagnostic accuracy helps him/her to detect easily which part of the system should be improved. The result of the first evaluation step, the system is able to analyze 87.9% of 221 student solutions correctly, motivates us to enhance the ability of intention analysis of the system. This can be done by taking helper predicates into consideration when diagnosing errors. That means, if a student solution uses a helper predicate, it can be categorized as *understandable* for error diagnosis. Improving the system, this way

corresponds to the intention of the evaluation of diagnostic accuracy because it is a type of internal evaluation ([1]).

2. From the perspective of an evaluator, the result of the second evaluation step, the diagnostic reliability, confirms the result of the first one, the intention analysis. If the result of the first evaluation step were low whereas the second one were high, this would raise a question: "How is a solution which cannot correctly be analyzed can yield a reliable diagnosis?" and we need to recheck our evaluation.
3. The measures Recall and Precision help us to suit the diagnosis according to our requirement. In our opinion, a coaching system would be more useful if it could provide as much diagnostic information as possible. That means, Recall should be maximized. However, if Recall increases, Precision will decreases because the diagnosis becomes less accurate if a wider range of bugs is considered. The Recall value of our system is higher than the Precision. This agrees with our intention.

Beside the advantages, our methodology of evaluating the diagnostic accuracy has a shortcoming. *"An important limitation to all diagnostic accuracy procedures is the assumption that the correctness of diagnoses can be unambiguously determined."* ([3]). Unfortunately, this assumption is not realizable for the error diagnosis in an ill-defined domain like logic programming. If an erroneous solution is diagnosed by several human experts, we expect different diagnostic results. While defining the gold standard for our evaluation of diagnostic reliability, the system developer did not agree with 3 of 535 comments which the Prolog expert made for the system's diagnosis. To solve the problem of disagreement between the system developer and the Prolog expert, they had to communicate and negotiated the gold standard.

In addition to off-line evaluations, we are planing to conduct an on-line evaluation for two purposes: 1) to confirm the ability of intention analysis and 2) to evaluate the educational impact of the system on students.

Acknowledgments

We would like to thank Thomas Kopinski for assessing the system's diagnosis.

References

[1] Iqbal, A.; Oppermann, R.; Patel, A. & Kinshuk (1999). A classification of evaluation methods for Intelligent Tutoring Systems. In *Software-Ergonomie '99, Design von Informationswelten, Gemeinsame Fachtagung des German Chapter of the ACM, der Gesellschaft füf Informatik (GI) und der SAP AG (pp.* 169-181*)*. Teubner.

[2] Johnson, W. L. (1990). Understanding and debugging novice programs. *International Journal of Artificial Intelligence,* 42, 51-97.

[3] Legree, P.; Gillis, P. & Orey, M. (1993). The quantitative evaluation of intelligent tutoring systems applications: product and process criteria. *International Journal of Artificial Intelligence in Education*, 4(2/3), 209-226.

[4] Le, N. T. (2007). Diagnosing errors in logic programming - The case of an ill-defined domain. *Technical Report*, University of Hamburg, Department of Informatics. FBI-HH-B-280/07.

[5] Le, N. T. & Menzel, W. (2008). The Coverage of Error Diagnosis in Logic Programming Using Weighted Constraints - The Case of an Ill-defined Domain. To be published in the Proceedings of 21st International FLAIRS Conference, Special Track on Intelligent Tutoring Systems.

[6] Littman, D. & Soloway, E. (1988). Evaluating ITSs: The Cognitive Science Perspective. In Polson, M. C. & Richardson, J. J. (Eds), *Foundations of Intelligent Tutoring Systems* (pp. 209-242). Lawrence Earlbaum Associate.

[7] Menzel, W. (2006). Constraint-based Modeling and Ambiguity. *International Journal of Artificial Intelligence in Education,* 16, 29-63.

[8] Ohlsson, S. (1994). Constraint-based student modelling. In Greer, J. E. & McCalla, G. I. (Eds), *Student Modelling: The Key to Individualized Knowledge-based Instruction (pp. 167-189).* Springer Verlag.

[9] Rijsbergen, C. J. van (1979). Information retrieval. Butterworths.

Appendix

A sample task for evaluation:
A salary database is implementedas a list whose odd elements represent names and even elements represent salaries measured in Euro. For example: [meier, 3600, schulze, 5400, mueller, 6300, ..., bauer, 4200]. Define a predicate which computes anew salary list based on the given one accordingto following rules: 1) a salary below or equal 5000 Euro wil be raised by 3%; 2) a salary above 5000 Euro will be raised by 2%.