

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

Nguyen-Think 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). Since in ambitious domains a perfect error diagnosis can not be guaranteed, the diagnostic accuracy of an ITS within an ill-defined domain should attract more 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). There are two reasons for such a focus: 1) The diagnosis component provides diagnostic information about student solutions and serves to build an appropriate student model; 2) Subsequent pedagogical interactions are dependent on the correct interpretation and diagnosis of student errors ([5]).

In ill-defined domains, however, the diagnosis of erroneous solutions becomes more difficult because of the following reasons:

1. For a problem task, there might be several alternative solution strategies. For example, to compute the return of an investment X after N years for a fixed interest rate Y , we can apply the analytic strategy $\text{Result} = X * (Y+1)^N$ or the strategy of recursive computation where again the cases of accumulative and naive recursion can be distinguished. As long as we are not able to identify the student's intention correctly, feedback might turn out to be useless or misleading guiding the student to another strategy which does not coincide with his/her own intention.
2. For an error there might exist many conflicting explanations ([10]). For example, the student is requested to write a subgoal like this X is $(A+B)*C$. An erroneous student solution might be $A+B*C$. The student has either missed the parentheses around $A+B$ or a factor C of a multiplication with A . If we want to give a feedback to the student, we have to choose between two correction proposals: “*You should put $A+B$ in parentheses*” or “*You should multiply the factor A with C* ”.

The problem of error diagnosis depends on the degree of freedom to compose solutions. We distinguish three freedom levels:

1. Problems for which there is just one possible solution. For example, “*What is the sum of 4 and 5? Please fill in: $4+5=$ __*”. For this task, the diagnosis is straightforward because only a number can be filled into the slot.

2. Problems for which there is one possible solution strategy which can be implemented by many variants. The problem statement of this kind of tasks is usually highly pre-specified. For example: the task “*Write a Prolog predicate to compute the return R of a investment X after N years for a fixed interest rate Y using the analytic strategy.*” and the following template are provided: investment($X,Y,N,Return$):-_____. The error diagnosis knows implicitly that the student can not apply other solution strategies than the analytic one which requires just one clause.
3. Problems which have alternative solution strategies and many implementation variants. For example: “*Write a Prolog predicate to compute the return of an investment after a period for a fixed interest rate.*” Diagnosing errors in solutions for this kind of problem tasks also requires identifying the intention of the students.

Therefore, the evaluation of diagnostic accuracy of ITSs for ill-defined domains should attract more attention than other ITS components as Littman ([8]) stated: “*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 ([7]). Furthermore, we demonstrate an evaluation methodology which measures diagnostic accuracy and is comprised of two parts: evaluation of intention analysis and 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. Afterwards, we compare our evaluation result with other systems in the fourth section. 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 homework. 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 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: the number of argument positions, the type for each argument position, and the mode of intended usage of the argument (input, output or indeterminate). If the signature is not appropriate, INCOM gives feedback to achieve a better understanding of the task. At the same time, the student's ability to analyze a problem is determined. In this paper we focus on the second phase where 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. That is, the system offers the students the freedom of level three described above.

The student is allowed to create programs using pure Prolog, a logic programming language. In the current version, helper predicates are provided explicitly in the problem statement. Under these conditions, the student has a free choice of solution strategies and can decide for an implementation variant. In addition, he/she may name variable and predicate identifiers as needed. Due to this degree of freedom, solving a programming task

requires the student to make numerous decisions to compose a solution. Thus, composing a program 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 a so-called semantic table and extend the constraint-based modeling (CBM) approach ([12]) with constraint weights. We clarify our approach by using the problem task *Salary* described in the Appendix.

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 1 specifies the required characteristics of correct solutions which implement the predicate *Salary/2*.

Strategy	Clause	Head	Body	Description
naive recursion	1	salary(OldL,NewL)	OldL=[] NewL=[]	Old list is empty New list is empty
	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 list recursively
	3	salary(OldL,NewL)	OldL=[N,S T] NewL=[N,Snew Tnew] S>5000 Snew is S+S*0.02 salary(T,Tnew)	N, S: name, salary build a new salary list Salary greater than 5000 Salary is increased Decompose old list recursively

Table 1: A partial semantic table for the predicate *Salary*

Constraints are means to model the principles of a domain and characteristics of correct solutions. Constraints of the latter category establish a connection between the entries in the semantic table and the student solution. They are used to examine whether the student solutions have the required characteristics of a correct solution. As there might be different solution strategies for a problem task, it is required to hypothesize which entry in the semantic table fits the student solution best. For this purpose, we attach a weight to each constraint. A constraint weight 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 also in order to measure the plausibility of different correction proposals for an error.

In principle, the diagnosis is carried out as an interaction of hypothesis generation and hypothesis evaluation. Hypotheses are interpretation possibilities for the student solution. 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 and the created mappings are hypotheses about the intention of the student; 2) Hypothesis evaluation: for each mapping, the plausibility of a selected hypothesis is computed based on the weights of the constraints it violates 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 ([6]).

2. Evaluation Methodology for Diagnostic Accuracy: State of the Art

Diagnostic accuracy is a particular performance metrics which is used to explore individual factors or features of an ITS ([2]). According to [8], 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 ITSs. More often, 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 problem solving environments whose freedom ranges between level one or two. That is, problem statements are highly pre-specified and input interfaces are structured, i.e. solution templates, slots, or selection menus. Using such an environment, the student inputs unambiguous solutions and the student's intention is identified uniquely. Providing this kind of problem solving environment, several constraint-based tutors have been developed for various domains such as SQL, database design and database normalization ([11]). Although those domains have characteristics of ill-definedness pertaining to tutoring, the mentioned constraint-based tutors take design decisions which actually should be made by the student themselves, or as Kodaganallur and colleagues stated: "*These tutors deliberately reduce task complexity to the point where they are not design tasks anymore.*" ([4], p.308).

Yet several ITSs, which allow alternative solution strategies and different implementation variants, emphasize the diagnostic accuracy in their evaluations. For example, PROUST ([3]) performs intention-based diagnosis of errors in novice PASCAL programs; PITS ([9]) and Hong's Prolog Tutor ([1]) diagnose errors in Prolog programs. These systems 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 (not recognized) bugs, and 3) number of false alarms which are bugs detected by the systems but not expected by a human tutor.

3. Evaluation of 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 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. To provide a more comprehensive presentation, for the latter

evaluation, we combine the measures of correctly recognized (not recognized) bugs and false alarms into the usual evaluation measures (Recall, Precision) for selection tasks.

3.1 Evaluation of Intention Analysis

To conduct the evaluation of intention analysis we have chosen seven problem tasks which require knowledge of recursive programming from past written examinations and selected corresponding student solutions for them. 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. The criteria for selecting student solutions have been: 1) A piece of code which satisfies a minimal requirement of interpreting it as a Prolog program; 2) Syntax errors have been removed because during the written examination the students did not have access to a computer; 3) Both correct and incorrect solutions have been selected. In total, we collected 221 solutions. For them INCOM was able to analyze 87.9% (sd=17.1%) correctly [7].

3.2 Evaluation of Diagnostic Reliability

3.2.1 Recall and Precision

According to [13] Recall and Precision are defined with respect to Table 2 as follows:

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

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

Table 2: 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, *retrieved* and *not-retrieved* bugs are decided by the system's diagnosis. The above measures can be understood as follows:

- A high precision means 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

The gold standard for our evaluation 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 to check every bug returned by INCOM for a given student solution. In order to facilitate his work, we developed an assessment tool. For each student solution, the expert is shown the exercise description, a manually added hypothesis about the

predicate's signature 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 a 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, e.g. he/she thought that crucial bugs have been missed.

The gold standard, that means the set 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 *should-be* set consists of bugs approved by the expert together with the bugs indicated as missing. The *should-not* set is the difference between the relevant constraints for the solution being investigated and the set of *should-be* bugs.

The human expert was not able to specify a gold standard for 7 collected student solutions because they were not interpretable.

3.2.3 Evaluation Results

For the evaluation of diagnostic reliability we applied the diagnosis to 214 student solutions for which a gold standard was available. For them, Table 3 summarizes the results for the seven different problem tasks with Task 3 being the problem task *Salary* described in the Appendix.

The result of our evaluation of diagnostic reliability shows that Recall 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 Precision is always lower than the Recall for each task. That means the diagnosis puts emphasis more on quantity than quality. Task 1 has the lowest Precision (0.8426) which indicates 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 for an error due to swapped argument positions, the system detects many bugs which are formally correct, but, not really helpful. Instead, a bug considering argument position should be enough, according to our expert. Note, that due to the method we used to determine the gold standard, precision measures are fairly optimistic, since the choice offered to the expert is strongly biased by the result of the system's diagnosis. The underlying data of our evaluation are available on (<https://nats-www.informatik.uni-hamburg.de/view/INCOM/Dokumentation>).

Task	1	2	3	4	5	6	7	Overall
Recall	0.9479	1.0000	1.0000	0.9010	1.0000	0.9811	0.9517	0.9688
Precision	0.8426	0.8750	1.0000	0.8906	0.9737	0.9528	0.9517	0.9266

Table 3: Evaluation of diagnostic reliability

4. Related Works

We compare INCOM with the systems PROUST, PITS and Hong's Prolog tutor with respect to the intention analysis and diagnostic reliability because those systems provide similar difficult problem tasks (Rainfall problem, List reversal). The categories for Recall and Precision have been calculated for PROUST as Table 4 shows, similarly, for the other systems. The results of Recall and Precision for different systems are reported in Table 5.

	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 4: A part of evaluation statistics of PROUST on the Rainfall problem

System	Intention Analysis	Precision	Recall
INCOM	87.9%	0.9266	0.9688
PROUST	96%	0.8787	0.8143
PITS ¹	80%	0.9580	0.9913
Prolog tutor	80%	1.0000	0.6886

Table 5: A comparison of the diagnostic accuracy.

The comparison shown in Table 5 is based on statistics reported in the corresponding literature and under the assumption that the gold standard of those systems has been specified comparatively. This comparison shows that PROUST is superior with respect to intention analysis but its Precision is lowest. Hong's Prolog tutor achieves the highest Precision, however, at the cost of a low Recall. Overall, INCOM combines an acceptable quality of intention analysis with a high diagnostic accuracy compared to the other systems.

5. Conclusions and Future Works

Combining the evaluation of the system's ability to correctly analyze the student's intention with an evaluation of its diagnostic reliability into an overall assessment of diagnostic accuracy has the following advantages:

1. From the perspective of a system developer, this two-step evaluation of diagnostic accuracy helps him/her to detect easily which part of the system should be improved. The fairly low accuracy of intention analysis of 87.9% indicates the necessity to enhance the ability of intention analysis of the system further. There are 8 solutions that have been implemented using helper predicates. Extending our system with the ability to correctly process them would raise the performance of the intention analysis to 92.7%.
2. The measures Recall and Precision help us to tailor the diagnosis according to our requirements. In our opinion, a coaching system would be more useful if it could provide as much diagnostic information as possible. That is, Recall should be maximized. However, if Recall increases, Precision will decrease 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.

Of course, our methodology of evaluating the diagnostic reliability shares a fundamental shortcoming with other methods because it assumes that "*the correctness of diagnoses can be unambiguously determined*" ([5]). Unfortunately, this assumption is not realistic in a domain like logic programming which allow alternative solution strategies and different implementation variants. Therefore, we should expect conflicting opinions if an erroneous solution is diagnosed by several human experts. Indeed, while defining the gold standard for our evaluation of diagnostic reliability, the system developer did not

¹ The calculation is based on statistics of the task "list reversal" and it is assumed that the recognition of applied algorithm roughly corresponds to the intention analysis of a program.

agree with 3 of 535 comments which the Prolog expert made for the system's diagnosis. This disagreement was eventually resolved by directly negotiating the gold standard. To compensate the inherent shortcomings of an off-line evaluation, we are planning to complement it with an on-line evaluation. It is intended: 1) to compare the ability of intention analysis evaluated in an interactive environment with the results from the off-line evaluation, 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] Hong, J. (2004). Guided programming and automated error analysis in an intelligent Prolog tutor. *International Journal of Human-Computer Studies*, 61(4), p.505-534.
- [2] 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 für Informatik (GI) und der SAP A*, p.169-181. Teubner.
- [3] Johnson, W. L. (1990). Understanding and debugging novice programs. *International Journal of Artificial Intelligence*, 42, p.51-97.
- [4] Kodaganallur, V.; Weitz, R.; & Rosenthal, D. (2006). An assessment of constraint-based tutors: A response to Mitrovic and Ohlsson's critique of "a comparison of model-tracing and constraint-based intelligent tutoring paradigms". *International Journal of AI in Education*, 16, p.291-321.
- [5] 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), p.209-226.
- [6] 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.
- [7] Le, N. T. & Menzel, W. (2008). The Coverage of Error Diagnosis in Logic Programming Using Weighted Constraints - The Case of an Ill-defined Domain. *Proceedings of 21st International FLAIRS Conference, Special Track on Intelligent Tutoring Systems*, p.421-426.
- [8] Littman, D. & Soloway, E. (1988). Evaluating ITSs: The Cognitive Science Perspective. In Polson, M. C. & Richardson, J. J. (Eds), *Foundations of Intelligent Tutoring Systems*, p. 209-242. Lawrence Erlbaum Associate.
- [9] Looi, C. K. (1991). Automatic debugging of prolog programs in a prolog intelligent tutoring system. *Instructional Science*, 20, p.215-263.
- [10] Menzel, W. (2006). Constraint-based Modeling and Ambiguity. *International Journal of Artificial Intelligence in Education*, 16, p.29-63.
- [11] Mitrovic, A.; Suraweera, P.; Martin, B. & Weerasinghe, A. (2004). DB-suite: Experiences with Three Intelligent, Web-based Database Tutors. *Journal of Interactive Learning Research (JILR)*. 15(4), p.409-432.
- [12] Ohlsson, S. (1994). Constraint-based student modelling. In Greer, J. E. & McCalla, G. I. (Eds), *Student Modelling: The Key to Individualized Knowledge-based Instruction*, p.167-189. Springer Verlag.
- [13] Rijsbergen, C. J. van (1979). Information retrieval. Butterworths.

Appendix

A sample task for evaluation: A salary database is implemented as 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 a new salary list based on the given one according to following rules: 1) a salary below or equal 5000 Euro will be raised by 3%; 2) a salary above 5000 Euro will be raised by 2%.