Combining Evaluative and Generative Diagnosis in ActiveMath

George GOGUADZE [a], Erica MELIS [b]
[a] *Univesity of Saarland*
[b] *German Research Center for Artificial Intelligence (DFKI)*

Abstract. Generative and evaluative approaches are two different ways of diagnosing students' input that have been realized in a number of intelligent tutoring systems. We describe how ActiveMath's exercise subsystem realizes both types of diagnosis and how it can combine them. A framework for distribution of diagnostic services is extending the existing web-service approach to mathematical services with the notion of context. In addition to an appropriate diagnosis, a goal is the interoperability of diagnosis services.

Keywords. diagnosis, evaluative and generative approach, user modeling, interoperability

Introduction

Generative and evaluative approaches are two different ways of diagnosing students' input that have been realized in a number of intelligent tutoring systems so far. The most common instances of generative and evaluative diagnoses are model-tracing (e.g., in cognitive tutors and Andes1) and constraint-based diagnosis (e.g., in SQL tutor, database place, KERMIT, NORMIT, CAPIT), respectively. Both approaches can yield feedback to errors. That is, a generative diagnosis can find a student's solution path (as well as an expert solution path), while the evaluative diagnosis deal with the student's solution (as well as with an expert's solution).

Mitrovic, Koedinger, and Martin as well as Kodaganallur, Weitz, and Rosenthal [6,4] have discussed the specificities, advantages and shortcomings of these approaches and even extended this discussion in critique and its response[7,5]. These discussions led to a better understanding of techniques at a general level and to suggestions on when to use which.

Wouldn't it be nice to use both, possibly even combined, to deliver appropriate diagnoses which are – after all – the basis for useful feedback and for student modeling?

Contribution. This paper describes ActiveMath's current diagnosis framework that makes use of generative mechanisms as well as evaluative ones and even combines these in cases. The employed generative principles are not novel themselves, just domain reasoning – although quite efficient by design. What is new for the

generative diagnoses is the (web-)service-approach for domain reasoners, which makes it possible to call several domain reasoners on the web even interleaving and to use standardized queries for this purpose. We describe a generic format for queries and the set of queries we currently use in ActiveMath.

The evaluative diagnosis differs from the typical constraint-based diagnoses in that it is represented by three evaluations : syntactic, numeric and semantic equivalence which are very generic – maybe more generic than what is claimed in [4].

The paper is organized as follows. First we review relevant parts of the knowledge representation and architecture of ActiveMath's exercise subsystem. Then we describe the diagnosis framework and queries to mathematical services, and finally evaluative and generative diagnoses and their interoperability.

1. Preliminaries about ActiveMath

ActiveMath is a web-based learning platform for mathematics. In addition to its adaptive course generation [11] and student model [3] a central component is its subsystem for interactive exercises that features ActiveMath's main functionalities of 'intelligent tutoring' including the student input's diagnosis. We briefly describe the overall architecture of ActiveMath's exercise subsystem.

Moreover, the knowledge representation of mathematical expressions (which belong to the problem to be solved and the student's input) and of interactive exercises is also relevant for the diagnosis as described below.

1.1. Knowledge Representation

Throughout ActiveMath, the OpenMath standard [10] is used to encode mathematical formulas. This semantic representation is a basis for the interoperability of various (web-)services in the diagnosis process. It allows for semantic diagnosis using external Computer Algebra System (CAS) and domain reasoner services. In order to enable communication to external diagnosis services their representation formats should be mapped to OpenMath. For this, so called phrasebooks are used. A phrasebook is a software application realizing the translation back and forth from OpenMath to the native language of a particular CAS or domain reasoner.

Exercises in ActiveMath are represented as finite state machines (possibly to be generated) which consist of nodes representing system-provided tasks, feedbacks and interactions, and transitions between nodes representing conditions that have to be satisfied for the learner's input in order for the transition to fire as well as corresponding diagnosis. A condition represents a query to the diagnosis services. Such a query can represent a constraint upon the user answer which can be evaluated by a CAS or it can be a query to a domain reasoner service. We describe the diagnosis services in more detail in section 2.

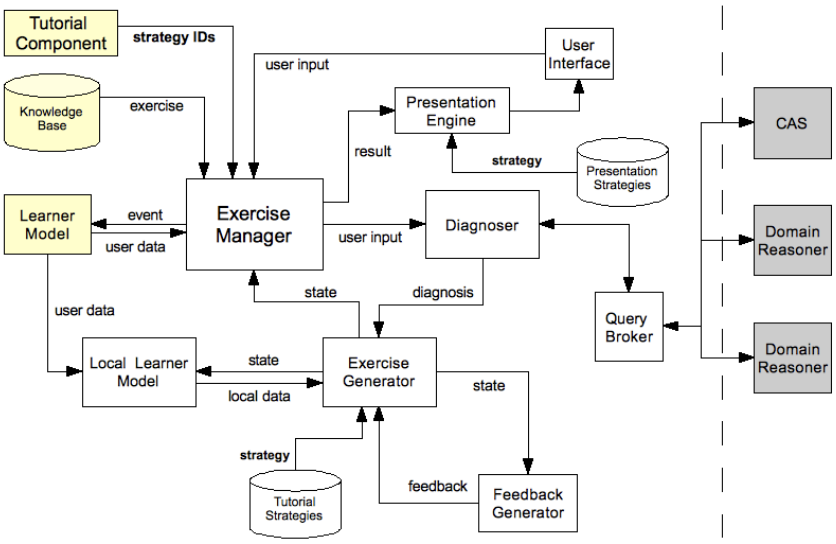

Figure 1. Exercise subsystem architecture

1.2. Architecture of ActiveMath *exercise subsystem*

Figure 1 shows components of the exercise system architecture. Pale yellow are the ActiveMath components external to the exercise subsystem and grey components are the CAS and domain reasoner services that are external to ActiveMath and communicate with the exercise system.

The central component of the exercise subsystem is the `Exercise Manager`, which coordinates the other components of the subsystem and controls the exercise process. Another important component is the `Exercise Generator` which is responsible for generating the nodes of the exercise.

This nodes can either be obtained from an authored part of the exercise representation, as well as generated automatically. Such a generation depends on the diagnosis of the learner's answer and on the tutorial strategy. The `Diagnoser` remotely connects to external services capable of generative or evaluative diagnosis via the `Query Broker`. Based on the diagnosis of the user action provided by a CAS or by a domain reasoner, the `Feedback Generator` component generates feedback automatically. Various `Tutorial Strategies` can be applied to the exercises [1] that define, e.g., what type of feedback has to be generated depending on the situation of the student, his previous activities and the strategy's pedagogical approach.

2. Diagnosis in ActiveMath

The `Diagnoser` module can access and query two kinds of systems, – actually two kind of services – that use mathematical domain knowledge for diagnosing the student's input:

- the first are generic computer algebra systems, which have their own solution algorithms that may be very different from how humans solve mathematical problems. Currently, ActiveMath integrates and communicates with the CASs: YACAS, Maxima, and WIRIS; phrasebooks for Maple and Mathematica are available too
- the other type of systems/services are domain reasoners such as SLOPERT [12], which encapsulate expert (and buggy) rules as humans use them in a specific (mathematical) domain, e.g., symbolic differentiation

Correct/incorrect diagnosis provided by a CAS can lead to the generation of a flag feedback. CASs also deliver final correct solutions. Authored exercises can encode queries to CAS that match the user answer against a buggy rule. A more detailed diagnosis can be obtained when a domain reasoner is available for the domain of the exercise. Additional queries can be sent to the domain reasoner in order to generate hints for the learner such as next step hint, or what is the correct answer for a current step, etc.

Both kinds of systems - CAS and domain reasoners can be queried by the `Diagnoser` for authored as well as for generated or for partially generated exercises.

2.1. Architecture of the diagnosis framework

Few systems try to make mathematical services such as CAS and even more advanced theorem provers accessible through the web. Examples of such are MONET services [8], or MathServe [13].

ActiveMath implements a novel service architecture for the diagnosis of student's actions in mathematical problem solving. The diagnosis task imposes some requirements upon such services, which we describe below. In ActiveMath, a broker architecture supports the `Diagnoser` to distribute queries to external diagnosis *services*, as shown in Figure 2.

The `Query Broker` accesses those services that are registered for the (mathematical) domain needed for the diagnosis. For instance, a domain reasoner for symbolic differentiation is only queried for problems in symbolic differentiation. The subscribed mathematical services themselves can also send a query back to the broker. For example, a domain reasoner for symbolic differentiation can send a query back to the broker if it needs to simplify an arithmetic expression. The Broker passes this new query to a CAS or arithmetic domain reasoner.

2.2. Evaluative Diagnosis in ActiveMath

In mathematics, different expressions can be considered 'correct' for a problem, i.e., 'equivalent' to a given solution. For instance, $\frac{1}{2}$ is (numerically) equivalent

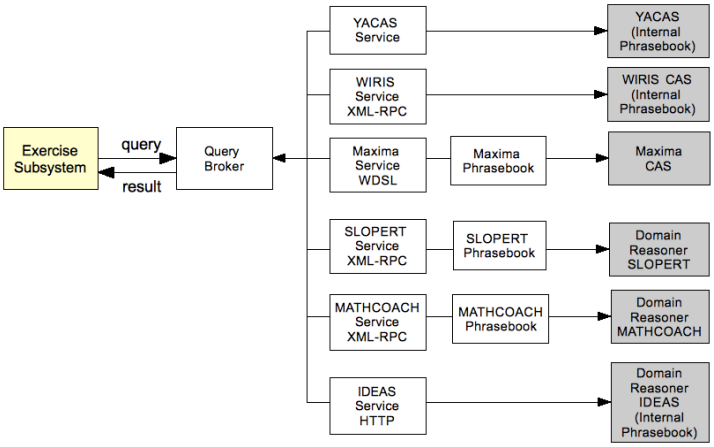

Figure 2. Diagnosis framework architecture

to 0.5 and $\frac{1}{\frac{a}{b}}$ is (semantically) equivalent to $\frac{b}{a}$. Not in all cases, the result $\frac{1}{\frac{a}{b}}$ is pedagogically as desirable as $\frac{b}{a}$, and in such cases the student's input has to be checked for syntactic equivalence too. That is, the three basic types of equivalences to be evaluated in ActiveMath's diagnoses are syntactic, numeric, and semantic equivalence. Syntactic equivalence is used for comparing the student's answer with the final correct result, numeric equivalence checks correctness up to a numeric simplification. Semantic comparison can be used for matching intermediate steps as well as for matching some typical errors and their semantic equivalents.

Syntactic equivalence holds if the student's answer and an expected correct result are literally the same. Numeric equivalence holds if the student's answer can be transformed numerically to an expected correct result. Semantic equivalence holds if the student's answer can be transformed semantically to an expected correct result.

More complex constraints are possible within semantic equivalence, for which a so called lambda comparator is used. Lambda comparator is a complex constraint that has to be satisfied for the learner's answer. In order to represent such a constraint, a lambda expression of the form $\lambda x.P(x)$ is used, where x is the bound variable corresponding to the user input and P is a predicate representing the constraint. For instance, the condition $\lambda x.x < 3$ is satisfied for any input which is smaller than 3. In general, such a comparator is needed if there are more than one input field in a required step and the learner's answer has to be handled as a vector. Then, the lambda comparator has a form $\lambda x_1 ... x_n.P(x_1, ..., x_n)$, where $P(x_1, ..., x_n)$ is an n-ary predicate. For example, $\lambda x, y.(x = 2 * y)$ is satisfied by an input pair of any expression and its double.

2.3. Generative Diagnosis in ActiveMath

The generative diagnosis leads to feedback of the types

- goal decomposition hints

- next step hints
- strategic hints
- worked out solution of the whole problem or a sub-problem
- identifying erroneous paths of student

ActiveMath's generative diagnosis is realized via queries to domain reasoners. Queries are sent to `Query Broker` which distributes them to the appropriate domain reasoner.

As usual, the domain reasoners can return expert and buggy solution paths and, if needed, a complete problem solving space. They can return paths starting and ending at given nodes in the problem solving space.

2.4. Queries.

We defined generic queries used to access any diagnosis service. Note that the main `compare` queries, which can be answered by CASs are indeed domain independent. Insofar, they support the claim that evaluative diagnosis is domain independent, i.e., no modeling of the domain is necessary upfront.

A *context* that can be included into queries characterizes the domain in even more detail: it defines (sub-)sets of rules and functions that a domain reasoner or a CAS is allowed to use for the diagnosis. The background for this restriction is that depending on the activity history, situation of a student and on the pedagogical approach, different rules (and functions) should be usable.

Consider the following example: The task of the student is to differentiate a function $f(x) = (x + 1) \cdot x$.

If this exercise is used in a situation when the student has not yet learned the product rule, a possible correct next step would be an arithmetic transformation that opens brackets, rather than product rule. In this case the evaluation of the student's answer needs to be done in a context excluding the product rule but including arithmetic context.

In order to formalize queries for evaluative and generative diagnosis we defined a representation that consists of :

- **action** of the query (e.g. `getResults`, `getUserSolutionPaths`)
- **(list of) expressions** e.g., task, user answer, correct answer
- **context** of action identifying the set of applicable rules (e.g. arithmetic, differentiation, logic)
- **number** of iterations defines how many atomic steps the domain reasoner should perform in the given context

In the following e, e1, e2, are OpenMath expressions, C is a context of a query, N is the number of iterations. A solution path is a list of results of consequent rule applications, annotated with rule identifiers.

Here are some queries currently used in ActiveMath:

- query(`getResults`, e, C, N) - returns the list of final nodes of all paths of length N starting at e in the context C
- query(`compare`, e1,e2, C, N) - returns true if there exists a path of the length N from e1 to e2 in the context C, false otherwise

- query(`getRules`, e, C) - returns the list of identifiers of expert production rules applicable to e in the context C
- query(`getUserSolutionPaths`, e1, e2, C, N) - returns the list of all paths of length N from e1 to e2 in the context C

Consider a following example query to a domain reasoner:

Example query: *Calculate the next two steps for computing derivative of the function* $f(x) = (x+1) \cdot x$ *using only arithmetic simplifications and differentiation rules except for a product rule.*

Using our query format we can formalize the example query as follows:

query(`getResults`, $(x + 1) \cdot x$, C, 2), where C is the composite context consisting of arithmetical context and differential rules without product rule.

It is easy to see that the comparisons relating to the three basic types of equivalences defined above correspond to instances of the query `compare`. Syntactic comparison corresponds to an empty context, numeric comparison to a specific numeric context, and semantic comparison to the global context.

2.5. Combinations of Generative and Evaluative Diagnosis

In manually authored exercises, transitions containing semantic equivalence queries to CAS or to domain reasoner can be authored. On the other hand, when a reasoner for the domain of the given problem is available, each new step of an exercise can be automatically generated based on diagnosis of the student input provided by this domain reasoner or by a CAS. Moreover, the evaluative and generative approach can be combined for exercises that are partially authored and enhanced on fly as explained below.

Currently, there are three ways to combine evaluative and generative diagnosis in ActiveMath.

Domain reasoner calls CAS As mentioned above, the domain reasoner might send a query back to the `Query Broker` since it might need to evaluate an expression in a different context then its own, and the query can then be forwarded to a CAS. An example from above is the domain reasoner for symbolic differentiation quering a CAS for arithmetic simplification.

Diagnoser Calls to Domain Reasoner and CAS are Sequenced. If a chosen tutorial strategy has to provide Informative Feedback (ITF) [9] rather than only flag feedback and bottom-out-hint, then more diagnosis is required.

Suppose the strategy requires to show the error position to the student. In this case, the `Diagnoser` first queries CAS to find out whether the student's answer is correct, and in case it is incorrect, it queries a domain reasoner in order to identify what is incorrect.

After this the `Feedback Generator` generates flag feedback, marking the error position, using the information on error position obtained from the domain reasoner.

Domain Reasoner is Called by a Tutorial Strategy. Even if an exercise is using evaluative diagnosis only, a tutorial strategy can decide that the `Feedback Generator` has to query the domain reasoner for the names of involved concepts,

or ask a domain reasoner to generate the next correct step in a solution in order to provide a hint.

3. Conclusion

We described how both, evaluative and generative diagnosis, is performed in ActiveMath and how it can be combined in places in order to use the best of both worlds, where it is appropriate. More specifically, we describe, how (mathematical) web-services are integrated to provide information for the `Diagnoser` and how they become interoperable.

Acknowledgement This article results from the ATuF project (ME 1136/5-1) funded by the German National Science Foundation (DFG).

References

[1] G. Goguadze, E. Melis One Exercise - Various Tutorial Strategies Proceedings of the International Conference on Intelligent Tutoring Systems ITS-2008 volume 5091, pages 755–757,

[2] G. Goguadze, I. Tsigler, Authoring Interactive Exercises in ActiveMath, In Proceedings of MathUI Workshop at Mathematical Knowledge Management Conference, June 2007.

[3] A. Faulhaber and E. Melis. An efficient student model based on student performance and metadata. In N. Fakotakis M. Ghallab, C.D. Spyropoulos and N. Avouris, editors, *18th European Conference on Artificial Intelligence (ECAI-2008)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 276–280. IOS Press, 2008.

[4] V. Kodaganallur, R.R. Weitz, and D. Rosenthal. A comparison of model-tracing and constraint-based intelligent tutoring paradigms. *International Journal For Artificial Intelligence in Education*, 15:117–144, 2005.

[5] V. Kodaganallur, R.R. Weitz, and D. Rosenthal. 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 For Artificial Intelligence in Education*, 16:291–321, 2006.

[6] A. Mitrovic, K.R. Koedinger, and B. Martin. A comparative analysis of cognitive tutoring and constraint-based modeling. In A. and F. de Rosis P. Brusilovsky, A. Corbett, editor, *Proceedings of the Ninth International Conference on User Modeling*, pages 313–322, Berlin, 2003. Springer-Verlag.

[7] A. Mitrovic and S. Ohlsson. A critique of Kodaganallur, Weitz and Rosenthal, 'a comparison of model-tracing and constraint-based intelligent tutoring paradigms'. *International Journal For Artificial Intelligence in Education*, 16:277–289, 2006.

[8] MONET Architecture Overview, The MONET Consortium, Deliverable D04, March, 2003

[9] S. Narciss. *Informatives Tutorielles Feedback.* Habilitationsschrift, Technische Universität Dresden, Fak. Mathematik und Naturwissenschaften, Dresden, Mai 2004.

[10] The OpenMath Standard, http://www.openmath.org

[11] C. Ullrich. *Pedagogically Founded Courseware Generation for Web-Based Learning – An HTN-Planning-Based Approach Implemented in PAIGOS.* Number 5260 in Lecture Notes in Artificial Intelligence. Springer, 2008.

[12] C. Zinn. Supporting tutorial feedback to student help requests and errors in symbolic differentiation. In K. Ashley M. Ikeda, editor, *Proceedings of Intelligent Tutoring Systems 8th. International Conference ITS-2006*, volume LNCS 4053 of *Lecture Notes in Computer Science*, pages 349–359. Springer-Verlag, June 2006.

[13] J. Zimmer and S. Autexier, The MathServe System for Semantic Web Reasoning Services, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130, pages 140–144, Springer Verlag, August 2006.