

AUTOMATED FEEDBACK FOR MATHEMATICAL LEARNING ENVIRONMENTS

Bastiaan Heeren and Johan Jeuring

Open University of the Netherlands and Utrecht University; bastiaan.heeren@ou.nl

Digital learning environments that offer well-designed feedback have the potential to enhance mathematics education. Building such a system is typically a huge and complex undertaking. Generating informative feedback at the level of steps a student takes requires the encoding of expert knowledge about the problem domain in software. The software component that processes this knowledge is traditionally called a domain reasoner. Such a reasoner can produce various types of feedback, for example about the correctness of a step, common errors, hints about how to proceed, or complete worked-out solutions.

In this paper, we highlight the main domain reasoner components that are responsible for generating feedback: rules, problem-solving procedures, normal forms, buggy rules, and constraints. Examples are drawn from the Digital Mathematics Environment (DME), which uses feedback generated by specialized domain reasoners for solving equations and structuring hypothesis tests. Similar techniques have also been used in tutoring systems for domains outside mathematics.

Keywords: intelligent tutoring systems, feedback generation, expert domain knowledge

INTRODUCTION

An intelligent tutoring system (ITS) helps students with learning a particular topic. It typically does this by offering learning material to study, tasks to solve, and by providing various kinds of help. The help provided by an ITS may take several forms: it might be through sequencing the tasks in a way that suits the student, providing scaffolding at the level of the student, giving elaborated feedback on the steps a student takes towards a solution for a task, helping a student take a next step towards a solution, etc.

Most ITSs do not need a teacher to provide help to a student: they automatically calculate the feedback on the work of a student, or a hint for a student. What do these systems need for calculating feedback, and how do they do this? A typical ITS has an expert knowledge module that contains the information necessary to calculate feedback and hints. In the last decade we have developed an approach to construct expert knowledge modules for a variety of ITSs, based on problem-solving strategies and rewriting and refinement steps. This paper describes our approach to developing expert knowledge modules, which we call domain reasoners. We show which components we need in our domain reasoners, and how we use domain reasoners in various ITSs. We will not describe how to design a full-blown environment for learning and practicing mathematics (such as the Digital Mathematical Environment (Drijvers, Boon, Doorman, Bokhove, & Tacoma, 2013)), nor how or when the calculated feedback is presented to a student (VanLehn, 2006), nor which kind of feedback is most effective (Bokhove & Drijvers, 2012).

We believe that explicitly describing the expert knowledge module concepts of an ITS can help designers and developers of ITSs, which often are complex software systems. Our approach can be applied in many domains, and the generality of the approach gives some guarantees about the consistency and completeness of the feedback provided.

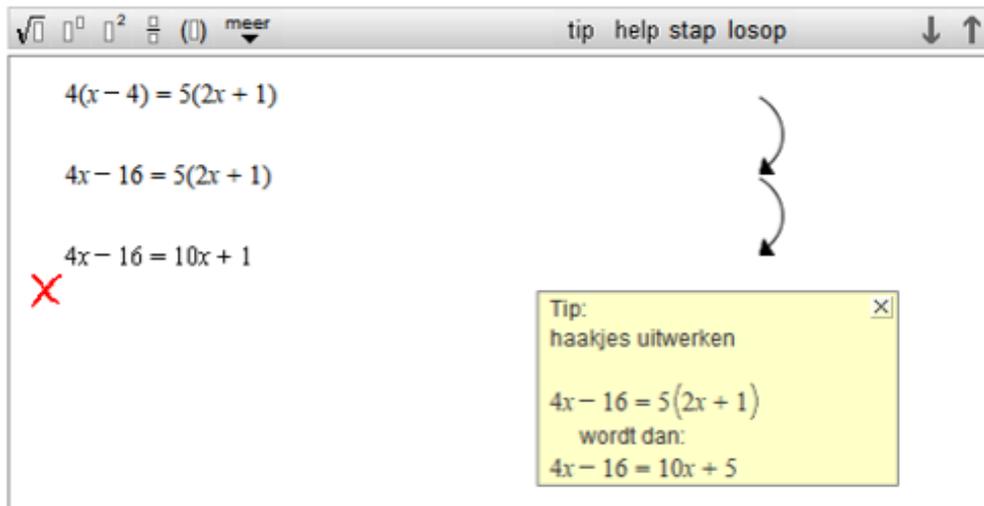


Figure 1: Inner-loop feedback, presented in the Digital Mathematics Environment

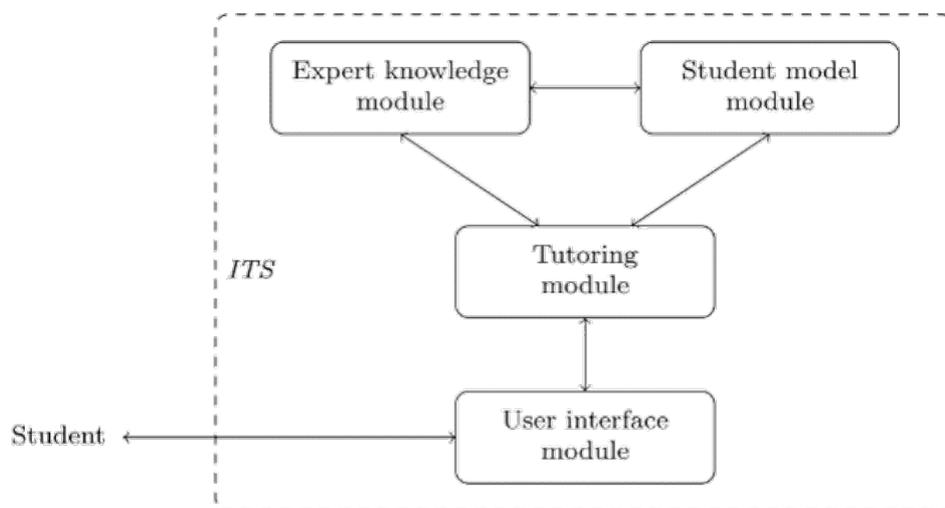


Figure 2: Classic structure of an ITS, decomposed into four components

In the remainder of the paper, we first introduce ITSs. We then show how we represent expert domain knowledge, and give many examples of ITSs in which we have used our approach. We conclude by expressing the paper's main points, and by identifying trends and challenges for the future.

INTELLIGENT TUTORING SYSTEMS

What is an Intelligent Tutoring System (ITS)? In his seminal paper, VanLehn (2006) explains that the behavior of such a system is structured around two loops. The outer loop concerns itself with solving one task after another. Feedback for the outer loop could, for example, suggest the next suitable task to solve. The inner loop considers the steps for solving one complex, multi-step task. Figure 1 gives an example of feedback at the inner loop: it combines feedback about correctness, high-level help ('remove parentheses'), and a bottom-out hint. In this paper, we only focus on feedback for the inner loop.

The four-component architecture shown in Figure 2 is the classic decomposition of an ITS in four parts (Nkambou, Bourdeau, & Psyché, 2010). The decomposition helps with assigning responsibilities: there are modules for user interaction, for pedagogical strategies (the tutoring

module: e.g., which hint facilities to offer), for modeling the current knowledge of a student, and for expressing expert domain knowledge. Although this conceptual architecture can be helpful for understanding the inner workings of ITSs, clear interfaces and communication protocols between the modules are missing, and in practice, one often finds monolithic systems instead of separate components.

Feedback and hints are generated by the expert knowledge module. Following Gogvadze (2011), we use the term domain reasoner for the part of the module that can ‘reason about problems’. This reasoning includes knowledge about the objects in a domain (e.g. expressions, equations), how these objects can be manipulated, and how to guide manipulation to reach a certain goal (Bundy, 1983).

For mathematical learning environments, computer algebra systems (CAS) can do part of the domain reasoner’s job. Such systems are powerful tools that are great at evaluating expressions. However, these tools have not been designed for providing feedback, and using built-in equality for comparing expressions can be very subtle. Hence, specialized domain reasoners have an advantage in generating feedback.

Narciss (2008) distinguishes the following widely used feedback types:

- knowledge of performance, e.g. percentage of tasks solved correctly;
- knowledge of result/response, e.g. correct or incorrect;
- knowledge of the correct response, which provides the correct answer;
- answer-until-correct feedback and multiple-try feedback, which provide extra opportunities after an incorrect answer;
- elaborated feedback, which provides additional information besides correctness and the correct answer.

Similar feedback types have been described by Shute (2008). Domain reasoners provide feedback services that are derived from the feedback types (Heeren & Jeuring, 2014). Intuitively, a feedback service is just a request-response communication pattern that exposes the capabilities of the domain reasoner. Services can correspond to the inner loop or the outer loop. Examples of service requests are: *Am I finished? Give me a next-step hint or worked-out solution. Is my step correct (step diagnosis)?* If yes: *does the step bring me closer to a solution?* If no: *is it a common mistake?*

EXPERT DOMAIN KNOWLEDGE

We use the IDEAS framework¹ for constructing domain reasoners: IDEAS (Interactive Domain-specific Exercise Assistants) is a generic, open-source software framework that can be used for expressing expert domain knowledge and for calculating automated feedback based on this knowledge. The framework is independent of the problem domain. Table 1 summarizes the main expert domain knowledge components, and how these components are used for calculating feedback. Certain feedback types require combinations of knowledge components, such as the checking procedure for steps (the ‘diagnose’ feedback service) that is used by the domain reasoner for feedback on the structure of hypothesis tests (Tacoma, Heeren, Jeuring, & Drijvers, 2019). The following sections discuss the knowledge components.

¹ <http://hackage.haskell.org/package/ideas>

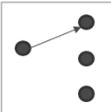
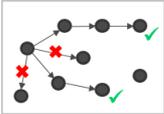
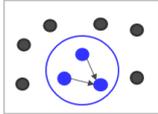
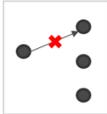
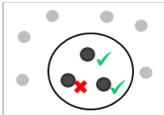
<i>component</i>		<i>used for</i>
rules		recognizing steps; suggesting possible next steps
problem-solving procedures		recognizing the solution strategy; detecting detours; providing next-step hints; providing worked-out examples
normal forms		recognizing steps; deciding whether in finished form or not; rewriting atypical expressions (e.g. $x+(-5)$ to $x-5$)
buggy rules		detecting common mistakes
constraints		checking properties or attributes; reporting violations

Table 1: Five knowledge components for expert domain knowledge, and how these are used

Rules

Rules specify the steps (manipulations) that are allowed: these steps can be rewriting steps (e.g. replacing $5 + 2$ by 7) or refinement steps (e.g. adding a line to a proof). For example, consider the rule for distributivity:

$$\forall abc . a(b + c) \rightarrow ab + ac$$

With this rule, the step $5(x + 2) \rightarrow 5x + 10$ can be taken towards reaching a solution. Rules can be used for recognizing steps, or for suggesting possible next steps.

When coding rules into software, it should be clear where the knowledge ends up. Preferably, the rule is programmed as a rewrite rule (Baader & Nipkow, 1997), which can be coded using a datatype-generic approach (Van Noort et al., 2010) as follows:

```
rule "distr" $ \a b c -> a * (b + c) :~> a * b + a * c
```

Observe the similarities between the rule's specification and its implementation. The explicit representation of the rewrite rule allows for further analysis and transformation, such as Knuth-Bendix completion for finding missing or conflicting rules (Knuth & Bendix, 1983), support for associative-commutative rewriting, rule inversion, automated testing, documentation, etc.

Problem-solving procedures

Whatever aspect of intelligence you attempt to model in a computer program, the same needs arise over and over again (Bundy, 1983): the need to have knowledge about the domain; the need to reason with that knowledge; the need for knowledge about how to direct or guide that reasoning. Problem-solving procedures describe sequences of rule applications that solve a particular task (Heeren & Jeuring, 2017), and thus address the third type of need. For instance, a common procedure for adding two fractions is: (1) find the lowest common denominator (LCD), (2) convert fractions to a form with

the LCD as denominator, (3) add the resulting fractions, and (4) simplify the result. We have developed a domain-specific language for specifying such procedures explicitly. This language provides a rich vocabulary for accurately specifying procedures, and introduces composition operators for combining simple procedures into complex composites. Examples of such operators are sequence, choice, repeat, try, prefer, and somewhere. With this language, the procedure for adding two fractions can be defined as:

FindLCD ; many(somewhere Convert) ; Add ; try Simplify

where FindLCD, Convert, Add, and Simplify are rules. An example of a step-wise derivation for this procedure is:

$$\frac{1}{2} + \frac{4}{5} \xrightarrow{\text{FindLCD}} \frac{1}{2} + \frac{4}{5} \xrightarrow{\text{Convert}} \frac{5}{10} + \frac{4}{5} \xrightarrow{\text{Convert}} \frac{5}{10} + \frac{8}{10} \xrightarrow{\text{Add}} \frac{13}{10} \xrightarrow{\text{Simplify}} 1 \frac{3}{10}$$

The first step finds 10 as LCD, which is used in the conversion steps that follow. Problem-solving procedures can be used for feed forward (providing next-step hints and worked-out examples), or to recognize the approach followed by a student and to detect possible detours. Problem-solving procedures help with following the steps of a model solution conform the model-tracing paradigm (Anderson, Boyle, Corbett, & Lewis, 1990). The composite structure of procedures also allows for decomposing procedures into parts, and to tailor feedback based on this decomposition.

Normal forms (equivalence classes)

Normal forms define classes of expressions that are treated the same, and select one canonical element for such a class (Heeren & Jeuring, 2009): for example, $10 + 5x \approx 5x + 10 \approx 5x + 5 \cdot 2$, for which $5x + 10$ is usually considered the standard notation. In mathematics, equivalences concerning associativity, commutativity, basic calculations, simplifications, etc. are often implicitly assumed for relations dealing with which expressions are equal, equivalent, similar, indistinguishable, etc. Equivalence classes make the granularity (step size) of a task explicit (McCalla, Greer, Barrie, & Pospisil, 1992). For example, given the rule for distributivity, $5(x + 2)$ and $5x + 10$ should be distinguishable.

Normal forms can be used for rewriting atypical expressions (e.g. replacing $x + (-5)$ by $x - 5$), and for deciding whether an expression is accepted as the final answer for a task or not (e.g. should $\sqrt{12}$ and $2\sqrt{3}$ be distinguishable). Normal forms can also be used for recognizing steps and rules.

Buggy rules

Buggy rules describe common mistakes and enable specialized feedback messages when detected. Consider, for example, a buggy rule for distribution:

$$\forall abc . a(b + c) \rightarrow ab + c$$

This buggy rule can be used for detecting the common mistake in $5(x + 3) \rightarrow 5x + 3$. More examples of buggy rules are the sign mistake in $5x = 2x + 3 \rightarrow 7x = 3$, and Hennecke's collection of 350 buggy rules for the fraction domain (Hennecke, 1999). Buggy rules are often associated with a sound rule.

Constraints

Constraints are based on the theory of learning from performance errors (Ohlsson, 1996) and can be used for checking properties and attributes, and reporting violations. Constraints have a relevance condition and a satisfaction condition: on violation (when relevant, but not satisfied), a special message can be reported. An example of a constraint is: if the equation is linear (relevance), then the

equation's right-hand side should not contain variable x (satisfaction). The corresponding constraint message would report that 'the equation is not yet solved because x appears on the right'.

EXAMPLES OF DOMAIN REASONERS

In this section we discuss examples of domain reasoners that have been developed with our approach, and their problem domains (see Figure 3). We will explain which knowledge components are used for generating feedback, and highlight some specifics of the problem domain.

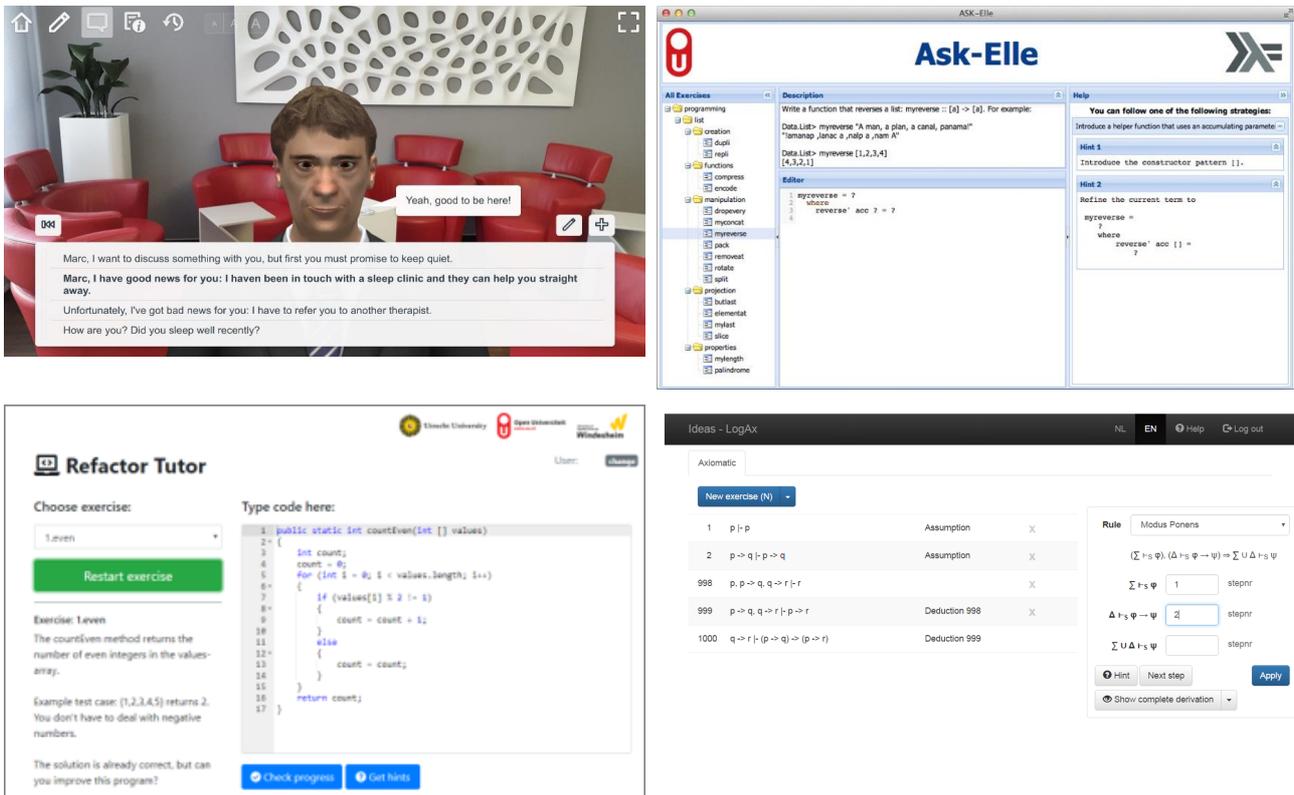


Figure 3: Examples of domain reasoners. Top row: Communicate! for training communication skills, and the functional programming tutor Ask-Elle. Bottom row: the Refactor Tutor for learning how to refactor imperative programs, and LogAx for constructing axiomatic proofs

Advise-Me. The Advise-Me assessment software analyses free-text input for math story problems (Heeren et al., 2018). The problems target mathematical competencies for setting up algebraic expressions and equations, and simplifying them. The software first extracts the mathematical expressions from answers, and then uses an analyzer that tries to recognize the solution steps (rules) and the high-level approach (problem-solving procedure). Normalizations are used for recognizing steps and buggy rules.

LogAx. LogAx is a tutor for constructing axiomatic proofs (Lodder, Heeren, & Jeurig, 2017), for example for proving that $q \rightarrow r \vdash (p \rightarrow q) \rightarrow (p \rightarrow r)$. Proofs are constructed in two directions: by formulating assumptions and combining these (forwards), or by working backwards from the goal. A directed acyclic multi-graph (DAM) is built that represents multiple proofs, and from this DAM a problem-solving procedure is generated. Feedback messages report subgoals and rules that can be applied. A student selects which rule she wants to apply and fills in a template for that rule. After an

unanticipated step by the student, the DAM and the problem-solving procedure are re-generated to facilitate giving feedback on subsequent student steps.

Ask-Elle. The functional programming tutor Ask-Elle lets students practice with defining small functions in Haskell (Gerdes, Heeren, Jeuring, & van Binsbergen, 2017). Holes can be used for unfinished parts in the program: a student can ask for hints on how to complete these holes. A problem-solving procedure is generated from multiple annotated model solutions: the procedure recognizes different solution approaches and can give feedback on this. The tutor relies on an extensive normalization procedure for recognizing many variations of the model solutions. For programs that cannot be recognized, constraints are used to test input-output correctness.

Refactor Tutor. This tutor lets students practice with refactoring small programs that are already functionally correct. The tutor uses a problem-solving strategy that imitates step-wise refactoring strategies by experts. Feedback is provided in a hint tree that can be expanded to see more detailed hints. Buggy rules capture common mistakes, such as logical errors in rewriting conditions. Similar to Ask-Elle, the tutor uses program normalizations for recognizing functionally equivalent programs, and constraints that perform input-output testing for detecting incorrect changes.

Communicate! The serious game Communicate! supports practicing interpersonal communication skills (Jeuring et al., 2015), for example between a health care professional and a patient. A virtual character is presented, and the player is offered a menu with sentences to choose from. Feedback is provided during the conversation (e.g. the flow of the conversation and emotions shown by the virtual character) and afterwards. Conversations can be scripted with a scenario editor. Scenarios are translated into problem-solving procedures.

CONCLUSION AND FUTURE WORK

In this paper we have presented our approach to automatically generating feedback in mathematical learning environments and Intelligent Tutoring Systems. To give better feedback, the approach is based on explicitly representing expert domain knowledge in such systems. We discussed which knowledge components we use for generating feedback, and explained that the step-size of a task is an often ignored, but very relevant aspect. Step-size can be controlled by defining normal forms, and using the hierarchical structure of problem-solving procedures.

Designing domain reasoners with feedback services simplifies the construction of ITSs. Feedback services result in loosely-coupled, reusable software components. The design follows the ‘separation of concerns’ design principle and localizes expert domain knowledge. Feedback services can be derived from the popular feedback types that have been described in literature. The presented approach can be applied to a wide range of problem domains.

For the future, we observe the following trends and challenges:

- Literature reports that every one hour of instruction that uses an ITS takes 200–300 hours for authoring content (Murray, 2003). There are design trade-offs for building an ITS. For example, supporting only one task simplifies feedback generation compared to supporting a full class of problems (e.g. solving quadratic equations), but reduces reusability and maintainability. We believe that software technology can help with developing high-quality, reusable solutions.
- There is a trend towards data-driven intelligent tutoring systems (Koedinger, Brunskill, Baker, McLaughlin, & Stamper, 2013). These systems use AI techniques to generate feedback from collected data, and typically scale well. This trend raises questions about the need for explicit

- expert domain knowledge. Hybrid solutions that only use collected data when expert domain knowledge cannot provide feedback may combine the best of both worlds.
- There is a need for further adaptation and personalization, both for the inner loop and the outer loop. This requires models for mastery learning (e.g. techniques for Bayesian knowledge tracing), and more advanced techniques to use the information from such models in domain reasoners.
 - Designing tools with automated feedback for less-structured problem domains, such as software design and learning foreign languages, is challenging, especially compared to the structured domain of mathematics.

REFERENCES

- Anderson, J., Boyle, C., Corbett, A., & Lewis, M. (1990). Cognitive modeling and intelligent tutoring. *Artificial intelligence*, 42(1), 7–49. doi: 10.5555/103358.103550
- Baader, F., & Nipkow, T. (1997). *Term rewriting and all that*. Cambridge University Press. doi: 10.5555/280474
- Bokhove, C., & Drijvers, P. (2012). Effects of feedback in an online algebra intervention. *Technology, Knowledge and Learning*, 17(1-2), 43–59. doi: 10.1007/s10758-012-9191-8
- Bundy, A. (1983). *The computer modelling of mathematical reasoning*. Academic Press. doi: 10.5555/3720
- Drijvers, P., Boon, P., Doorman, M., Bokhove, C., & Tacoma, S. (2013). Digital design: RME principles for designing online tasks. In C. Margolinas (Ed.), *Proceedings of ICMI study 22: Task design in mathematics education* (p. 55-62).
- Gerdes, A., Heeren, B., Jeuring, J., & van Binsbergen, L. (2017). Ask-Elle: an adaptable programming tutor for Haskell giving automated feedback. *International Journal of AIED*, 27, 65–100. doi: 10.1007/s40593-015-0080-x
- Gogvadze, G. (2011). *ActiveMath - generation and reuse of interactive exercises using domain reasoners and automated tutorial strategies* (Unpublished doctoral dissertation). Universität des Saarlandes, Germany.
- Heeren, B., & Jeuring, J. (2009). Canonical forms in interactive exercise assistants. In J. Carette, L. Dixon, C. S. Coen, & S. M. Watt (Eds.), *Intelligent computer mathematics* (pp. 325–340). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-02614-0_27
- Heeren, B., & Jeuring, J. (2014). Feedback services for stepwise exercises. *Science of Computer Programming*, 88, 110–129. doi: 10.1016/j.scico.2014.02.021
- Heeren, B., & Jeuring, J. (2017). An extensible domain-specific language for describing problem-solving procedures. In E. André, R. Baker, X. Hu, M. M. T. Rodrigo, & B. du Boulay (Eds.), *Artificial intelligence in education* (pp. 77–89). Cham: Springer International Publishing. doi: 10.1007/978-3-319-61425-0_7
- Heeren, B., Jeuring, J., Sosnovsky, S., Drijvers, P., Boon, P., Tacoma, S., ... van Walree, F. (2018). Fine-grained cognitive assessment based on free-form input for math story problems. In V. Pammer-Schindler, M. Pérez-Sanagustín, H. Drachsler, R. Elferink, & M. Scheffel (Eds.), *Proceedings of European conference on technology enhanced learning (EC-TEL) 2018* (pp. 262–276). Springer International Publishing. doi: 10.1007/978-3-319-98572-5_20

- Hennecke, M. (1999). *Online diagnose in intelligenten mathematischen lehr-lern-systemen (in German)* (Unpublished doctoral dissertation). Hildesheim University. (Fortschritt-Berichte VDI Reihe 10, Informatik / Kommunikationstechnik; 605. Düsseldorf: VDI-Verlag)
- Jeuring, J., Grosfeld, F., Heeren, B., Hulsbergen, M., IJntema, R., Jonker, V., ... van Zeijts, H. (2015). Communicate! — a serious game for communication skills. In *EC-TEL 2015* (Vol. 9307, pp. 513–517). Springer. doi: 10.1007/978-3-319-24258-3_49
- Knuth, D. E., & Bendix, P. B. (1983). Simple word problems in universal algebras. In J. H. Siekmann & G. Wrightson (Eds.), *Automation of reasoning: 2: Classical papers on computational logic 1967–1970* (pp. 342–376). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-81955-1_23
- Koedinger, K., Brunskill, E., Baker, R., McLaughlin, E., & Stamper, J. (2013). New potentials for data-driven intelligent tutoring system development and optimization. *AI Magazine*, 34(3), 27–41. doi: 10.1609/aimag.v34i3.2484
- Lodder, J., Heeren, B., & Jeuring, J. (2017). Generating hints and feedback for Hilbert-style axiomatic proofs. In *SIGCSE 2017* (pp. 387–392). doi: 10.1145/3017680.3017736
- McCalla, G., Greer, J., Barrie, B., & Pospisil, P. (1992). Granularity hierarchies. *Computers & Mathematics with Applications*, 23(2), 363 - 375. doi: 10.1016/0898-1221(92)90148-B
- Murray, T. (2003). An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. In T. Murray, S. Blessing, & S. Ainsworth (Eds.), *Authoring tools for advanced technology learning environments* (p. 491-544). Springer. doi: 10.1007/978-94-017-0819-7_17
- Narciss, S. (2008). Feedback strategies for interactive learning tasks. *Handbook of research on educational communications and technology*, 125–144. doi: 10.4324/9780203880869.ch11
- Nkambou, R., Bourdeau, J., & Psyché, V. (2010). Building intelligent tutoring systems: An overview. In R. N. et al. (Ed.), *Advances in intelligent tutoring systems, SCI 308* (pp. 361–375). Springer. doi: 10.1007/978-3-642-14363-2_18
- Ohlsson, S. (1996). Learning from performance errors. *Psychological Review*, 103(2), 241–262. doi: 10.1037/0033-295X.103.2.241
- Shute, V. J. (2008). Focus on formative feedback. *Review of Educational Research*, 78(1), 153–189. doi: 10.3102/0034654307313795
- Tacoma, S., Heeren, B., Jeuring, J., & Drijvers, P. (2019). Automated feedback on the structure of hypothesis tests. In S. Isotani, E. Millán, A. Ogan, P. Hastings, B. McLaren, & R. Luckin (Eds.), *Artificial intelligence in education* (pp. 281–285). Cham: Springer International Publishing. doi: 10.1007/978-3-030-23207-8_52
- VanLehn, K. (2006). The behavior of tutoring systems. *Journal of AIED*, 16(3), 227–265. doi: 10.5555/1435351.1435353
- Van Noort, T., Rodriguez Yakushev, A., Holdermans, S., Jeuring, J., Heeren, B., & Magalhães, J. P. (2010). A lightweight approach to datatype-generic rewriting. *Journal of Functional Programming*, 20(3-4), 375–413. doi: 10.1017/S0956796810000183

DuEPublico

Duisburg-Essen Publications online

UNIVERSITÄT
DUISBURG
ESSEN

Offen im Denken

ub | universitäts
bibliothek

Published in: 14th International Conference on Technology in Mathematics Teaching 2019

This text is made available via DuEPublico, the institutional repository of the University of Duisburg-Essen. This version may eventually differ from another version distributed by a commercial publisher.

DOI: 10.17185/duepublico/71231

URN: urn:nbn:de:hbz:464-20200115-113049-3



This work may be used under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 License (CC BY-NC-ND 4.0)