A Web Application for Mathematics Education*

Ana Paula Tomás[1,2], José Paulo Leal[1,2], and Marcos Aurélio Domingues[2]

[1] Departamento de Ciência de Computadores, Faculdade de Ciências, Universidade do Porto, Portugal
{apt,zp}@dcc.fc.up.pt
[2] Laboratório de Inteligência Artificial e Ciência de Computadores, Universidade do Porto, Portugal
marcos@liacc.up.pt

Abstract. AGILMAT is a web application designed to help students learn Mathematics, with focus on high-school algebra and calculus drills. A modular and extensible architecture and a wizard-based configuration interface decoupled from the system core are major design features of AGILMAT. The drill expressions are specified by grammars and constraints imposed by default profiles and user options, so that AGILMAT may support distinct learning levels and stages. The core system uses symbolic manipulation and automated reasoning to provide correct answers for the drills. The paper shows how AGILMAT may be used to create and customize drills automatically.

1 Introduction

Improving proficiency in mathematics stands at the top of educational priorities in Portugal, as well as in many countries. Although it is a fact that some students lack mathematical skills, the main reason for the lack of success in mathematics is that too often students memorize how to work out a specific drill but do not grasp the concepts underlying the solution steps. Some students even prefer quick practical rules with no mathematical basis, such as *"to see if* $|x| > b$ *should be rewritten as* $x < -b \vee x > b$ *or as* $x < -b \wedge x > b$*, rotate* $>$ *by* $90°$ *clockwise"*.

Over the last three decades applications of computers in mathematics education as a supplement to regular class or as a primary means of instruction became more and more widespread [1,2,11]. The fast growth of Internet has also fostered a significant breakthrough in computer assisted learning. Sophisticated web-based learning environments are being developed also for mathematics education, some offering authoring tools for creating courseware, assignments and exams, some being used for training, assessment and contests [5,8,15,18]. Nevertheless, designing courseware material for e-learning is still quite time-consuming, even when instructors may count on e-learning authoring tools [5,15]. Advances in computer technology shall therefore be exploited to develop really re-usable and customizable contents, for personalized e-learning.

* Partially supported by AGILMAT (POSI/CHS/48565/2002), funded by Fundação para a Ciência e Tecnologia, under POSI, co-funded by EU/FEDER.

H. Leung et al. (Eds.): ICWL 2007, LNCS 4823, pp. 380–391, 2008.

AGILMAT project (`www.ncc.up.pt/AGILMAT/`) aims to develop tools to automatically create and solve mathematics drills, for computer aided training or assessment. Drill and practice, together with exposition, are still the main modes of instruction in Mathematics. Nevertheless, very often drills created by online exercise systems lack pedagogical interest, as they tend to be too hard-wired and just correspond to randomly generated instances of the same problem template [18].

In contrast to the most usual practice, the analysis of algebraic procedures that students should learn and apply in a given curriculum plays a central role in AGILMAT. In particular, for the abstraction of problem templates and their formal description, we try to understand how these algorithms condition the drills that may be created and solved automatically [16]. The expressions that arise in the exercises are specified by constrained grammars and refined by default profiles and user options.

AGILMAT supports different initial settings by parametrizing the interface, generator and solver. The user is given the possibility to further refine some of these parameters so as to customize the exercises. The released prototype[1] allows users to create hundreds of examples and their one-line solutions, about univariate functions. Drills currently supported are directed to high school students and mathematics teachers (grade levels 10 to 12). We may explore the advantages of this tool as a means for learning fundamental notions. For example, by making students understand and justify why the one-line solutions produced by the system are correct. The possibility of using AGILMAT for this purpose was validated in informal demos of the system to high-school students and teachers.

AGILMAT implementation integrates different technologies and programming languages. Pursuing the line of [16,17], the core system is implemented in a Prolog-based constraint logic programming language [10,13,14] to support constraints on the exercises.

The framework that supports AGILMAT's interface is described in [3,6]. It is loosely inspired in the model-view-controller (MVC) pattern and implemented on a Java servlet container (J2EE Web Application, using the Apache Tomcat Servlet Container, version 5.5). A main feature is the fact that it can be extended without requiring code programming. The hot spots of such framework are XML configuration files to define the interface data, how this data is mapped into the system's commands, and how commands output and the interaction state is mapped into web formatting languages. With this approach the web interface is kept separated from the system it controls, it is easy to define and modify, and is able to capture enough domain knowledge to be a real advantage for the novice or sporadic user.

The rest of the paper is structured as follows. In Section 2, we show how AGILMAT may be used to create and customize drills automatically. Section 3 presents its architecture in brief and some implementation details. In Section 4, we address ongoing work that aims at extending the prototype and discuss some

[1] The released version of AGILMAT is available at `www.ncc.up.pt:8080/Agilmat`.

of the theoretical problems inherent to mathematical knowledge management that we have to circumvent.

2 Generating Exercises

AGILMAT's front-end is a web application that collects a set of options that will control the generation of exercises. Figure 1 presents AGILMAT initial interface where the user can generate a set of exercises defining only a minimal set of options.

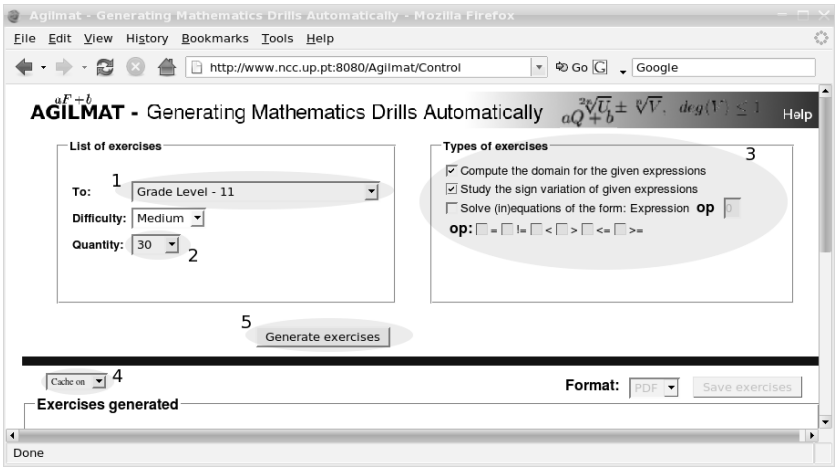

Fig. 1. Screen of the main interface of AGILMAT

The location of these options in the users interface is highlighted in Figure 1. Each one is numbered and its purpose is described below.

Option 1 - Defines a profile of exercises that will be generated, set to "Grade Level - 11" in the example. This makes the system initialize the range of each parameter of AGILMAT's back-end with suitable default values (defined by such profile). These values may correspond not only to grade levels but also to specific topics of the curricula. The advantage of this option is to allow novice users to generate exercises for their particular needs without having to set a large number of parameters.

Option 2 - Defines the maximum number of expressions that will be generated for the exercises, set to "30", in the figure.

Option 3 - Defines the types of exercises that will be generated. The user can use this option to choose the types of exercises that he/she does want, such as "Compute the domain for the given expressions", "Study the sign variation of given expressions" and "Solve (in)equations". Exercise sheets may

contain several different types of exercises. We set the option to "Compute the domain for the given expressions" and "Study the sign variation of given expressions".

Option 4 - Turns on and off the cache system. On "Cache on" the system gets exercises from the ones cached, if there are some. Otherwise, AGILMAT will generate fresh exercises.

Option 5 - Finally, if we press "Generate exercises", some exercises will be displayed in the bottom of the screen, as we show in Figure 2.

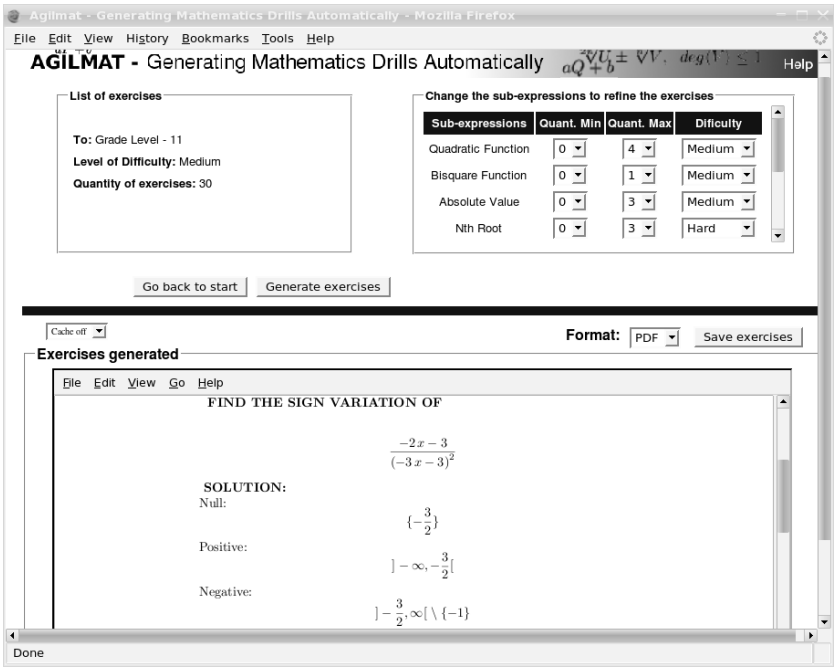

Fig. 2. Examples of exercises created by AGILMAT

2.1 Refining the Exercises

The user can refine exercise generation by setting different values for particular parameters of AGILMAT's back-end. After the first step, the interface header changes to present a comprehensive set of parameters that control variables in AGILMAT's back-end. The parameters, their range and default values depend on the basic profile defined previously. Figure 2 shows the parameters that would be made available under the conditions imposed by the selections made in the previous panel. If the user changes, for instance, his requirements on the number of occurrences of "Nth Root" (to be exactly 1) and "Quotient of Functions" (to be 0) and asks AGILMAT to generate exercises, then all the expressions will contain exactly one radical and no quotient, as we see in Figure 3.

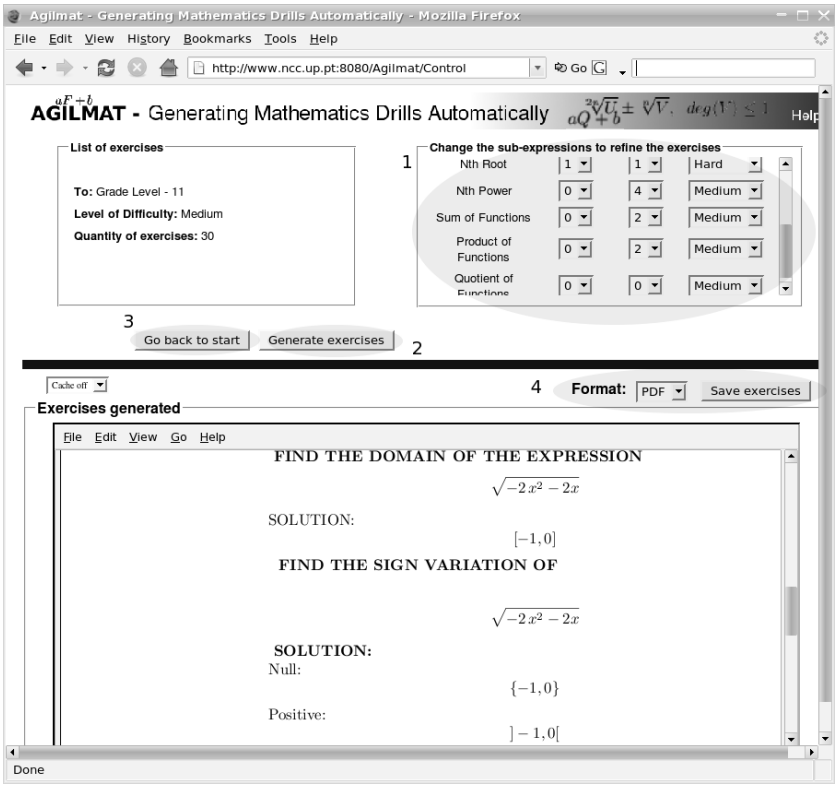

Fig. 3. Exercises generated by AGILMAT after a refinement of the number of occurrences of "Nth Root" and "Quotient of Functions", now set to 1 and 0

The highlighted options in Figure 3 have the following roles:

Option 1 - Allows the user to change the number of occurrences of a particular constructor (or sub-expression) that can arise in the exercises, as well as its difficulty level. Three difficulty levels are supported currently – easy, medium and hard – mapped internally to integer values or ranges, that are defined in the selected profile.

Option 2 - To generate exercises again, the user presses the button "Generate exercises".

Option 3 - The user can return to the initial interface by pressing the button "Go back to start".

Option 4 - Generated exercises can be saved in different formats by pressing "Save exercises". Formats currently supported are HyperText Markup Language (HTML), Portable Document Format (PDF) and PostScript (PS) and XML - Question & Test Interoperability (XML - QTI), although for the released version, the available format is restricted to PDF.

The user may go on redefining or refining the parameters. Within AGILMAT core system, the difficulty level is modeled as a sum of ranks given to the basic functions involved in the expressions and to their composition. The user may partially tune this level by changing the selections in the "Difficulty" menus (see Figure 3, option 1).

In Figure 4, we see another excerpt of an exercise sheet. This random example

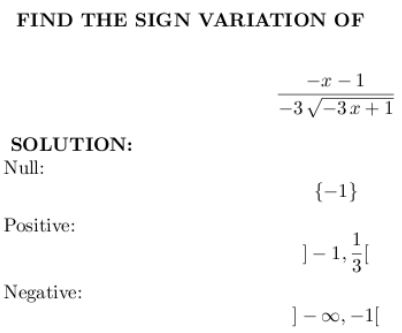

Fig. 4. Requiring one radical and one quotient

was created by setting the number of occurrences of "Quotient of Functions" to 1 and reducing its difficulty level and that of "Nth Root" to *easy*.

If the difficulty level of these constructs was the same as in Figure 3, the constraints imposed by the parameters would be inconsistent. The requirement of medium global cumulative difficulty could not be satisfied. AGILMAT would give a warning, informing the user that it could not create exercises that satisfy the given parameters.

Alternative interfaces, where the users could tune several finer parameters, were tested with a focus group of six high-school teachers before we decided for the current version. In particular, they did some experiments with another interface where they could control finer subcategories of expressions, as in our former testbed generator Demomath [16]. They found it too complex, not very intuitive and somewhat bewildering. Hence, we decided to keep some parameters fixed through default configuration profiles.

The default profile fixes the mapping of the difficulty ranks (easy, medium and hard) to integer values, which may be distinct from basic function to basic function. By changing this encoding, we may tune AGILMAT to particular needs, although the underlying model for estimating the difficulty level is rather simplistic.

The predefined parameters include also the ranges of exponents (of radicals and powers) and of coefficients that may occur in the expressions. In order to keep the numbers that may arise in the computations small, the coefficients of the created expressions are relatively small also (e.g., integers ranging over $[-5, 5]$).

As we see in the previous figures, the released prototype of AGILMAT, that is available in the World Wide Web, yields a one-line solution for each exercise. This may be very helpful to complement conventional class-based tuition, favoring retention of fundamental concepts or results.

It is important that students acquire or improve abstraction skills. Very often students do not really have to work out a solution with paper-and-pencil to grasp why the solution computed by AGILMAT is correct, as for example, when the problem is to "Find the set of solutions of $\frac{|-x-3|}{-x} \leq 0$", and AGILMAT output is $\{-3\} \cup]0, \infty[$. Students may be asked to justify why -3 is a solution and why x shall be positive otherwise.

The critical analysis of the one-line solutions output by AGILMAT may have a positive impact for the mastery of forms of reasoning. Although there is no formal user study, we exploited this feature during informal demos of AGILMAT to high school students and teachers. Their reactions were enough supportive and encouraging, although students' first reaction is usually to say that they are not especially keen on Mathematics.

As we can see in Figure 5, AGILMAT may be used to create more complex exercises. In this case, the solutions are not so immediate. To justify them, students need to work out the solutions steps using paper-and-pencil.

CHARACTERIZE THE MONOTONICITY OF

$$r(x) = -2\left(\frac{-2x+1}{2x^2 - x + 1}\right) - 2$$

SOLUTION:

r **is strictly increasing in**

$$]\frac{1}{2} - \frac{1}{2}\sqrt{2}, \frac{1}{2} + \frac{1}{2}\sqrt{2}[$$

r **is strictly decreasing in**

$$]-\infty, \frac{1}{2} - \frac{1}{2}\sqrt{2}[\cup]\frac{1}{2} + \frac{1}{2}\sqrt{2}, \infty[$$

Fig. 5. More complex exercises (grades 12)

3 Overview of AGILMAT's Architecture

The AGILMAT system is available to its users through a web interface. The AGILMAT system itself is a Constraint Logic Programing application, while its web interface is based on a web wizards framework developed as a Java web application. These two components are loosely coupled, thus providing a high degree of independence between them. The web wizards framework connects with the AGILMAT system through I/O streams, issuing Prolog queries and receiving its output. The web wizard generator uses XML files to configure the dependencies to the AGILMAT system. In particular, format conversions use

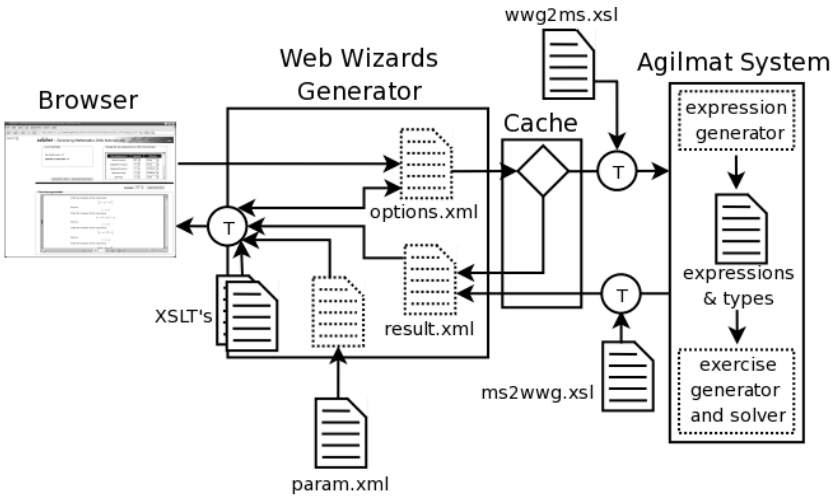

Fig. 6. AGILMAT system in the framework for developing web wizards

XSLT (W3C Recommendation, 1999). This setup is complemented by a cache system that avoids regenerating exercises in certain cases. Figure 6 contextualizes the AGILMAT system inside the framework for developing web wizards.

The web wizards generator, cache and AGILMAT system are represented as strong squares and rectangles, internal modules of the AGILMAT system are represented as dotted rectangles, points of transformation are represented as strong T labeled circles. A switcher is represented by strong rhombus. Physical files with XML documents are represented as strong file icons and XML documents in memory are represented as dotted file icons.

3.1 Web Wizards Generator

The state of the interaction with AGILMAT's web interface is managed by a set of *parameters* defined in the file `param.xml`, a valid document of a parameter definition language defined for this purpose. The parameter definition language includes such features as: composition of parameters, definition of default values as expression involving other parameters, dependencies between parameters, among others.

At each user request, parameters are converted into Prolog queries that are injected in the input stream to be interpreted and executed by Prolog engine. The document `wwg2ms.xsl` is used to map parameters into Prolog clauses that feed the AGILMAT's system. Although this conversion could be handled entirely on the framework's side, using XSLT transformations, we opted to keep this conversion fairly simple and develop a Prolog module on AGILMAT's system to process the parameters collected by the web interface.

The document `ms2wwg.xsl` is used to produce an HTML interface to display the current interaction state (namely the selected parameter values) and the exercises generated by the AGILMAT's system. For that purpose we had to convert the exercises and their solutions to XML formatting languages, which required the addition of a new Prolog module to serialize terms into an XML format, to the AGILMAT's system. In this case we could not have avoided doing this conversion on the Prolog process side since XLST cannot handle Prolog terms as input.

Using this XML representation, we can use an alternate `ms2wwg.xsl` to transform the exercises to different formats, such as the format XML - Question & Test Interoperability (XML - QTI) [9] with mathematical expressions represented in MathML. By now we expected to have a better support for QTI in the major e-learning platforms, such as Moodle [20], but unfortunately this proved not to be the case. To cope with this fact we are presently developing transformation for platform specific formats, such as Moodle XML [21].

In the released version, exercises and their solutions are converted to a LaTeX representation that may be converted to different formats, such as: HyperText Markup Language (HTML), Portable Document Format (PDF) and PostScript (PS). The PDF file is embedded in the web interface. The released prototype is not yet using the document `ms2wwg.xsl` to convert exercises and their solutions to a XML representation. We hope to use this document in the next version of AGILMAT.

To improve the response time of the overall system with the web interface, we developed a cache system. When the cache is activated, the web wizards generator looks up for a system output previously generated with the same choice of parameters. Our experience showed that certain sets of choices, specially those selected in an early stage of the interaction, tend to be repeated since they are just default values of the initial screens. In these situations the cache system provides almost immediate feedback in these first attempts which encourages novice users to continue exploring the system's more complex features.

3.2 AGILMAT's Core System

In the AGILMAT system there are two main modules written in Prolog acting as filters: the *expression generator* processes the user constraints and produces an expressions and types file, the *exercise generator and solver* processes this file and produces exercises and theirs solutions. This last module is the control of the system and makes use of several libraries that handle arithmetic, set operations and symbolic constraints (to solve inequations, disequations and equations), along the lines of [16].

An interesting feature that distinguishes also AGILMAT from e-learning tools that use pools of exercises [5] is the fact that its expression generator may potentially yield arbitrarily complex expressions. In the current release, these expressions are characterized by the grammar proposed in [16]. This grammar was written to describe a non-trivial set of expressions whose zeros and domains can be exactly computed by the algebraic procedures that students should master

(up to the 12th grade). The constraints imposed by the selected profile and the user options restrict these generic forms further. The expression generator is a Prolog program that uses constraint programming to model such constraints as constraints on finite domain (integer) variables. Each generated expression belongs to some pattern (i.e., *type*). For instance, all the expressions of the following forms

$$\frac{a}{|\,by+c\,|} \quad \text{and} \quad \sqrt[n]{\frac{a}{(\,by+c\,)^m}}$$

would be of types `k / abs o p1 o x` and `rad(`n`) o (k / pow(`m`) o p1 o x)`, respectively, where `k`, `abs`, `p1`, `pow(`m`)` and `rad(`n`)` represent a constant function, the absolute value function, a polynomial function of degree 1 and the m^{th}-power and n^{th}-root functions, respectively. Further technical details may be found in [16].

The configuration parameters constrain these types. The constants arising in the expression – coefficients and exponents of radicals and powers – are also restricted by the values given in each configuration profile (defined in `param.xml`). The user cannot change their ranges directly in the interface, except by selecting a distinct default profile. As we mentioned already, a preliminary version of the interface, where users could tune several finer parameters, was tested with a focus group of high-school teachers. They found it too complex. Hence, we decided to keep parameters of this sort fixed through the default configuration profile.

The generator computes the type of the expressions and one (or more) expressions of each type. In this way, it is possible to produce several expressions with exactly the same type but distinct coefficients. This option is not yet available in web interface, although it is an interesting feature for applications in assessment, for instance.

All the modules supporting the solver are implemented in a Prolog-based constraint logic programming language. To guarantee the correctness of the computed answers and a consistent (safe) interaction with users, AGILMAT must have full control of the simplifications and rewritings performed. More often, web-based learning environments make use of available Computer Algebra Systems to reduce the effort of writing a solver, e.g., ActiveMath [7] and Wims [18], risking unexpected or wrong answers, as discussed in [1,4,11].

4 Some Ongoing Extensions and Discussion

We are addressing extensions of the core system to produce step-by-step solutions that resemble those worked out by human beings. Some initial results have been reported in [17]. This is a non-trivial design principle or goal common to some other mathematical tools for assisted learning, as MathXpert [1]. To understand and automate what people do when they do mathematics is a continuous research topic in Artificial Intelligence, Automated Reasoning, and Symbolic Computation [2,12].

When a student is using mathematical software for exploratory learning, it is reasonable that the system may output *don't know* or rather complicate formulas

as an answer. This is not acceptable when the system is *asking* the student to solve a problem that it has automatically generated. Both the computer and the student (if he/she has learned the topic in assessment) must know how to solve the exercise. For instance, the system shall not produce an ad-hoc polynomial of degree greater than four and ask the student to find its roots. Indeed, it is known that there exist no generic algorithm to solve that. In contrast, there are algorithms to compute the *rational* roots of any polynomial with rational coefficients, which, nevertheless students may not have learned. For this reason, we try to understand how the algorithms learners should master condition the drills that may be created and solved automatically.

We are also considering extensions of the forms of expressions covered. In particular, we are investigating extensions of the types of numbers and sets supported. Supporting symbolic computation with generic real numbers raises theoretical difficulties. Many problems become undecidable. For instance, the computable real numbers represent a null measure set (within the real numbers) and there is no algorithm for deciding whether some real valued expression is zero. This means that from the computational point of view, there are some inherent difficulties that we have to circumvent. For educational purposes, we do not need to support full generality.

References

1. Beeson, M.: Design Principles of Mathpert: Software to support education in algebra and calculus. In: Kajler, N. (ed.) Computer-Human Interaction in Symbolic Computation, Texts and Monographs in Symbolic Computation, vol. XI, pp. 89–115. Springer, Heidelberg (1998)
2. Bundy, A.: The Computer Modelling of Mathematical Reasoning. Academic Press, London (1983)
3. Domingues, M.A., Leal, J.P.: Configuring Web Wizards in XML. In: Proc. of XATA2006, XML: Aplicações e Tecnologias Associadas, pp. 315–324 (2006)
4. Gottliebsen, H., Kelsey, T., Martin, U.: Hidden Verification for Computational Mathematics. J. Symbolic Computation 39, 539–567 (2005)
5. Isidro, R.O., Sousa Pinto, J., Batel Anjo, A.: SA3C - Platform of Evaluation System and Computer Assisted Learning. WEAS Transactions on Advances in Engineering Education 1:2, 1–6 (2005), `http://pmate.ua.pt:8081/pmate/`
6. Leal, J.P., Domingues, M.A.: Rapid development of web interfaces to heterogeneous systems. In: van Leeuwen, J., et al. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 716–725. Springer, Heidelberg (2007)
7. Melis, E., et al.: ActiveMath: A Generic and Adaptive Web-Based Learning Environment. International Journal of Artificial Intelligence in Education 12:4, 385–407 (2001), `http://www.activemath.org/`
8. FP6-Project "LeActiveMath": Language Enhanced User Adaptive, Interactive eLearning for Mathematics (2004/2006) `http://www.dfki.de/leactivemath/`
9. IMS QTI Specifications. IMS Global Learning Consortium, Inc. `www.imsglobal.org/question/index.html`
10. Marriott, K., Stuckey, P.: Programming with Constraints – An Introduction. MIT Press, Cambridge (1998)

11. Ravaglia, R., et al.: Successful Pedagogical Applications of Symbolic Computation. In: Kajler, N. (ed.) Computer-Human Interaction in Symbolic Computation, pp. 61–88. Springer, Heidelberg (1999)
12. Robinson, A., Voronkoy, A. (eds.): Handbook of Automated Reasoning. Elsevier Science, Amsterdam (2001)
13. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier Science, Amsterdam (2006)
14. SICStus Prolog User Manual (Release 3.12.0), SICS, Sweden (2004) http://www.sics.se/isl/sicstuswww/site/index.html
15. Sierra, J., et al.: A highly modular and extensible architecture for an integrated IMS-based authoring system: The <e-Aula> experience. Softw., Pract. Exper. 37:4, 441–461 (2007)
16. Tomás, A.P., Leal, J.P.: A CLP-Based Tool for Computer Aided Generation and Solving of Maths Exercises. In: Dahl, V., Wadler, P. (eds.) PADL 2003. LNCS, vol. 2562, pp. 223–240. Springer, Heidelberg (2002)
17. Tomás, A.P., Moreira, N., Pereira, N.: Designing a Solver for Arithmetic Constraints to Support Education in Mathematics. In: Proc. Artificial Intelligence Applications and Innovations (AIAI 2006). IFIP Series, vol. 204, pp. 433–441. Springer-Verlag, Heidelberg (2006)
18. Xiao, G.: On Public-Questions Tests, Univ. Nice Sophia-Antipolis, France (2004)
19. XSL Transformations (XSLT) W3C Recommendation (November 16, 1999), http://www.w3.org/TR/xslt
20. Moodle course management system. http://moodle.org/
21. Moodle XML. http://docs.moodle.org/en/Moodle_XML