Automatic Generation and Delivery of Multiple-Choice Math Quizzes*

Ana Paula Tomás[1] and José Paulo Leal[2]

[1] DCC & CMUP, Faculdade de Ciências, Universidade do Porto, Portugal
[2] DCC & CRACS-INESC TEC, Faculdade de Ciências, Universidade do Porto
{apt,zp}@dcc.fc.up.pt

Abstract. We present an application of constraint logic programming to create multiple-choice questions for math quizzes. Constraints are used for the configuration of the generator, giving the user some flexibility to customize the forms of the expressions arising in the exercises. Constraints are also used to control the application of the buggy rules in the derivation of plausible wrong solutions to the quiz questions. We developed a prototype based on the core system of AGILMAT [18]. For delivering math quizzes to students, we used an automatic evaluation feature of Mooshak [8] that was improved to handle math expressions. The communication between the two systems - AgilmatQuiz and Mooshak - relies on a specially designed LaTeX based quiz format. This tool is being used at our institution to create quizzes to support assessment in a PreCalculus course for first year undergraduate students.

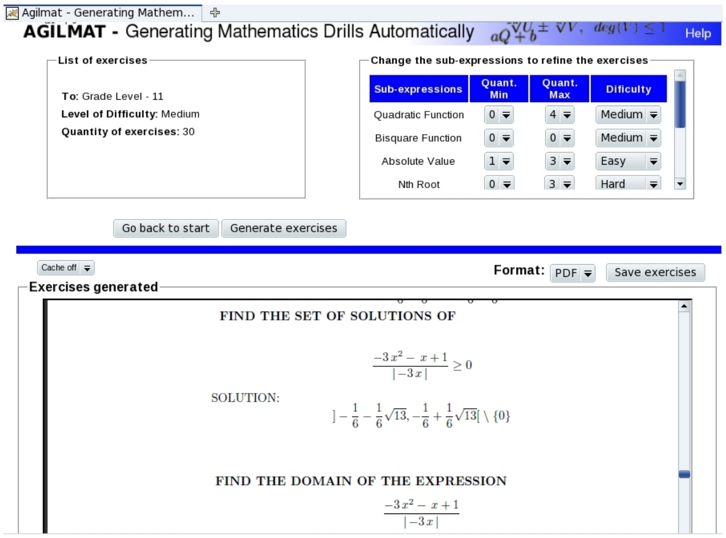
1 Introduction

Mathematics assessment should help both student and teacher to understand what the student knows, and to identify areas in which the student needs improvement [14]. As a diagnostic means for assessing conceptual understanding and procedural fluency, multiple-choice tests are quite popular. They can be given and graded at low cost, in contrast to tests with open-answer questions. Nevertheless, their construction remains a time-consuming task. It can get easier when some authoring tool and a bank of questions can be used to produce the tests. There exist hundreds of exercise assistants. The use of a bank of questions, or of templates with parameters that can be randomly instantiated to create variants of the exercises is the most common approach in the design of systems that provide either interactive drills or multiple-choice tests for mathematics (e.g., [11,12,15,20,23,25]). The use of collections of semantically annotated mathematical learning objects is a trend [23,24]. Very often, the exercise systems provide worked out solutions for the drills or automatic feedback that is somehow hardwired to the problem model, even if encoded as a solution graph.

* Research funded in part by the ERDF/COMPETE Programme and by the Portuguese Government through the FCT - Fundação para a Ciência e Tecnologia under the projects PEst-C/MAT/UI0144/2011 and FCOMP-01-0124-FEDER-022701.

When the intended answer is a mathematical expression, some systems give automatic feedback by making use of computer algebra systems or specific domain reasoners to diagnose the student answer [5,12,23] or to give hints [1,5,21]. This is useful for formative assessment (i.e., for self-assessment or assessment that is not directly contributing to the student grade). A similar feedback may be given for interactive multiple-choice questions, based on the analysis of individual responses and on the particular exercise model [5,12].

In [17], we proposed a novel approach for creating the drills, and adopted it for developing the AGILMAT prototype[1]. Instead of fixing the template of the parametric expression that is included in the question generator model [11], we focused on the algebraic procedures students know or learn in order to abstract and restrict the expressions in the questions. For that purpose, we tried to understand how the curricula contents condition the drills. This approach is feasible when we consider routine exercises about some topics, and their one-line solutions [17] or step-by-step solutions [1]. In AGILMAT, the expressions arising in drills are specified by constrained grammars and refined by some default profiles and possible user options (see Fig. 1). This is a distinguishing feature



AGILMAT - Generating Mathematics Drills Automatically

List of exercises

To: Grade Level - 11
Level of Difficulty: Medium
Quantity of exercises: 30

Change the sub-expressions to refine the exercises

Sub-expressions	Quant. Min	Quant. Max	Dificulty
Quadratic Function	0	4	Medium
Bisquare Function	0	0	Medium
Absolute Value	1	3	Easy
Nth Root	0	3	Hard

Format: PDF

Exercises generated

FIND THE SET OF SOLUTIONS OF

$$\frac{-3x^2-x+1}{|-3x|} \geq 0$$

SOLUTION: $]-\frac{1}{6}-\frac{1}{6}\sqrt{13}, -\frac{1}{6}+\frac{1}{6}\sqrt{13}[\setminus \{0\}$

FIND THE DOMAIN OF THE EXPRESSION

$$\frac{-3x^2-x+1}{|-3x|}$$

Fig. 1. The AGILMAT prototype available on the web. The notation $]a, b[$ and $[a, b[$ is employed instead of (a, b) and $[a, b)$ to represent intervals of real numbers

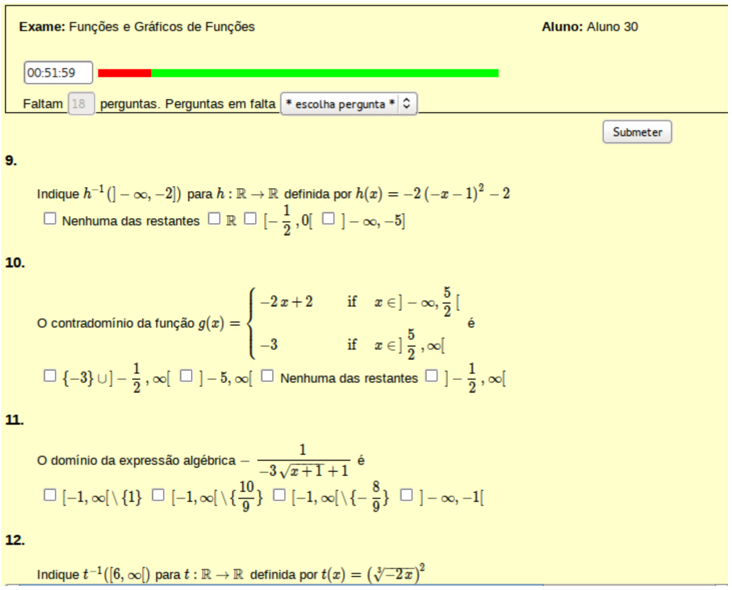
and an advantage of our work, making possible the generation of a large number of (non-trivial) examples. For a concise explanation on how this is done, please refer to section 2. A more detailed description can be found in [17,18]. Such

[1] http://www.dcc.fc.up.pt:8080/Agilmat

feature gives the user great flexibility to control the forms of the expressions. The system can produce automatically very different expressions or several expressions with the same pattern. The drills are always created and solved on the fly, if the cache option (see Fig. 1) is turned off.

1.1 AgilmatQuiz

In this paper, we describe AgilmatQuiz, the prototype we developed for producing multiple-choice questions for a Pre-Calculus course[2]. This course is being offered at our institution to the first year undergraduate students lacking the required mathematical background. AgilmatQuiz is based on an extension of the AGILMAT core system. Fig. 2 presents a screenshot of an exercise sheet, where questions 9 to 12, among some others, were produced using AgilmatQuiz.



Exame: Funções e Gráficos de Funções **Aluno:** Aluno 30

00:51:59

Faltam 18 perguntas. Perguntas em falta * escolha pergunta *

9.

Indique $h^{-1}(]-\infty,-2])$ para $h:\mathbb{R}\to\mathbb{R}$ definida por $h(x) = -2\,(-x-1)^2 - 2$

☐ Nenhuma das restantes ☐ $\mathbb{R}$ ☐ $[-\frac{1}{2},0[$ ☐ $]-\infty,-5]$

10.

O contradomínio da função $g(x) = \begin{cases} -2\,x+2 & \text{if} \quad x \in]-\infty,\frac{5}{2}[\\ -3 & \text{if} \quad x \in]\frac{5}{2},\infty[\end{cases}$ é

☐ $\{-3\}\cup]-\frac{1}{2},\infty[$ ☐ $]-5,\infty[$ ☐ Nenhuma das restantes ☐ $]-\frac{1}{2},\infty[$

11.

O domínio da expressão algébrica $-\dfrac{1}{-3\sqrt{x+1}+1}$ é

☐ $[-1,\infty[\setminus\{1\}$ ☐ $[-1,\infty[\setminus\{\frac{10}{9}\}$ ☐ $[-1,\infty[\setminus\{-\frac{8}{9}\}$ ☐ $]-\infty,-1[$

12.

Indique $t^{-1}([6,\infty[)$ para $t:\mathbb{R}\to\mathbb{R}$ definida por $t(x) = (\sqrt[3]{-2x})^2$

Fig. 2. Multiple-choice exercises about the notions of reciprocal image (9), range (10) and domain (11) of real valued algebraic functions, created by our system to a quiz

New types of expressions and of exercises were introduced. In particular, we were asked to create exercises about disjunctions and conjunctions of simple linear constraints, an extension that was quite easy. We were asked to create exercises involving piecewise-defined functions. This raises some difficulties that

[2] Please access `http://www.dcc.fc.up.pt/~apt/Research/AgilmatQuiz.html` to see more examples of questions created by our system.

we discuss in the paper. The questions created automatically cover essentially algebraic functions taught at high-school, the notions of domain, range, reciprocal image, piecewise-defined functions and the solution of equations and inequalities. Some questions about other topics were written by colleagues from the Mathematics Department.

For delivering math quizzes to students, we used Mooshak[3], a web based competitive learning system. Mooshak was originally developed for managing programming contests over the Internet under the ACM International Collegiate Programming Contest rules [8]. Quiz delivery is one of the educational features it supports currently. For AgilmatQuiz, it was improved to handle math expressions. The quiz questions are structured into groups and written in a specially designed LaTeX based quiz format. LaTeX is widely used in academia and therefore it made easier also the collaboration of our colleagues who were creating some questions by hand.

The rest of the paper is structured as follows. In section 2, we describe the main lines of the approach followed in AGILMAT and for the quiz generation. In section 3, we address the main changes introduced in the AGILMAT core system to be able to produce quiz questions. In section 4, we explain how the Mooshak system supports quiz delivery and grading. Section 5 concludes the paper.

2 Creating Drills Using AGILMAT

In the AGILMAT core system there are two main modular components – the *expression generator* and *exercise generator and solver* – which were implemented using Prolog based constraint programming languages. The *expression generator* processes the user constraints and produces a file with expressions and their *types* (i.e., templates). The *exercise generator and solver* processes this file and produces exercises (according to a specification) and their solutions. This module makes use of several submodules that handle arithmetic, set operations and symbolic constraints (to solve inequations, disequations and equations), along the lines we described in [17]. It uses also some modules for computing limits and derivatives of functions, performing simplications, and obtaining the image of a function when applied to a set (or an upper bound on this image). In addition, it uses a module for converting the internal representations of the exercises to LaTeX, as well as their solutions.

Each exercise produced by the AGILMAT prototype has a *question* where a function expression is required. The expressions are built from polynomial functions, the absolute value function $x \rightarrow |x|$, and the power and radix functions $x \rightarrow x^n$ and $x \rightarrow \sqrt[n]{x}$, possibly using composition, addition, product and quotient operations. In the implementation of the expression generator, we follow the grammar proposed in [17] for the expressions. This grammar characterizes a wide range of algebraic expressions found in high school textbooks and whose zeros can be exactly computed by an algorithm students learn. To illustrate the main ideas, we present a fragment in Fig. 3. The category *sumexpr* denotes

[3] `http://mooshak.dcc.fc.up.pt`

$$\begin{array}{lll}
\mathit{prodexpr} & \longrightarrow & \mathit{factor} \mid \mathit{factor} * \mathit{prodexpr} \\
\mathit{factor} & \longrightarrow & \mathit{sumexpr} \mid \mathit{basic} \\
\mathit{basic} & \longrightarrow & \mathit{ipol}_2(\mathtt{x}) \mid \mathit{bisqr} \\
 & \longrightarrow & \mathit{fbasic} \mid \mathit{fpol}_1(\mathit{fbasic}) \mid \mathit{fpol}_1(\mathtt{x}) \\
\mathit{fbasic} & \longrightarrow & \mathtt{abs(}\mathit{basic}\mathtt{)} \mid \mathtt{pow(}\mathit{basic}, N\mathtt{)} \mid \mathtt{rad(}\mathit{basic}, N\mathtt{)} \\
\mathit{ipol}_2(T) & \longrightarrow & \mathtt{pol(}T\mathtt{,[}a,b,c\mathtt{])},\ \ abc \neq 0 \\
\mathit{fpol}_1(T) & \longrightarrow & \mathtt{pol(}T\mathtt{,[}a,b\mathtt{])},\ \ a \neq 0 \\
\mathit{bisqr} & \longrightarrow & \mathit{ipol}_2(\mathtt{pow(x,}N\mathtt{)}),\ \ N \geq 2
\end{array}$$

Fig. 3. An fragment of the grammar proposed in [17]

some particular forms of sum expressions. We can see that $\sqrt[3]{(5x-1)^2}$ and $\sqrt{(2x+3)^5}$ are expressions of the category *basic* and instances of $\sqrt[N]{(ax+b)^M}$. This is rewritten as $\mathtt{rad}(\mathtt{pow}(\mathtt{pol}(x,[a,b]),M),N)$ and the expressions of this form are characterized by the pattern `rad(N) o pow(M) o p1 o x`. Composition, denoted by $\circ$, is the main operation. The generation of expressions for the exercises is driven by the generation of their patterns.

Each pattern, called *type*, is represented by a Prolog term with finite domain variables. These variables bind the exponents and, hence, can be constrained to tailor the expressions to specific needs. For example, if we need exercises about the quadratic function, we can constrain the degree of the expression to be two. The exponents are instantiated when the system creates an instance of the expression. Below, we show two other examples of couples of patterns and expressions produced by our generator, their internal representations and usual typesetting.

```
rad(3)o(abs o p1 o pow(2)o p1 o x + pow(2)o p1 o x)
rad(abs(pol(pow(pol(x,[3,-4]),2),[-4,-2])),3) + pow(pol(x,[1,-3]),2)
```

$$\sqrt[3]{|-4(3x-4)^2-2|+(x-3)^2}$$

```
pow(7)o ip(1)o(p1 o x/p1 o x)
pow(pol(pol(y,[-2,-1])/pol(y,[-3,4]),[-2,3]),7)
```

$$\left(-2\frac{-2y-1}{-3y+4}+3\right)^7$$

Here, `ip(1)` is the internal pattern of expressions of the form $\mathtt{pol(}T\mathtt{,}[a,b]\mathtt{)}$, with a and b non-null, whereas, `p1` corresponds to $\mathtt{pol(}T\mathtt{,}[a,b]\mathtt{)}$, with b unrestricted. The variable in the expression is not relevant for the template and is passed as a parameter to the generator.

Each type can be used by the system to generate a single or several exercises of the same type. In this way, the difficulty level of the distinct versions would be similar as they are instances of the same template (only coefficients change). This can be important for grading. Nevertheless, for self-assessment or practice, drills with very different expressions favour the learning of concepts and properties instead of focusing on very specific methods for particular instances.

By setting parameters of the generator, the user can refine the types created. Some parameters are used to define constraints on the number of occurrences of each primitive function and of some categories, and also to enforce constraints on the difficulty level of the expressions. The latter is modeled as a weighted sum of the difficulty rates assigned to the primitive functions and some distinguished forms of constructs.

The prototype available on the web (see Fig. 1) allows some customization by default profiles but also by parameters that can be refined by the user. For further details, please refer to [17,18]. However, this is fairly less than what a user that knows CLP can do by interacting directly at the low level. The interface was kept simple because a preliminary version where users could tune several finer parameters was found too complex by a focus group of teachers.

The generator is implemented in a Prolog based constraint programming language and runs on top of the SICStus Prolog system [26]. The constraints act on finite domain variables associated to the types. In general, the grammar rules were implemented by predicates of the form

category(Type,Degree,Rate,CountTypes,CountOps)

where Degree, Rate, CountTypes, CountOps are parameters used to constrain the generated Type. For example, for the *prodexpr* type, we can have:

```
prodstype(T,G,Rate,CTs,Ops) :-
  constrs(CTs,urestrs_factor),factorstype(T,G,Rate,CTs,Ops).
prodstype(Tb*T,G,Rate,CTs,Ops) :-
  rate_restr(prodstype,Rate,[RateB,RateT]),
  types_restr(prodstype,CTs,[CTsB,CTsT]),
  ops_restr(Ops,[1,OpsB,OpsT]), OpsT #=< OpsB,
  Gb #>= 1, Gt #>= 1, G #= Gt+Gb,
  constrs(CTsB,urestrs_factor),factorstype(Tb,Gb,RateB,CTsB,OpsB),
  prodstype(T,Gt,RateT,CTsT,OpsT).
```

where constrs/2, rate_restr/3 and ops_restr/2 impose new constraints on subtypes, rates and number of operators. Here, OpsT #=< OpsB is added to discard some symmetries. The user can define the rate value of the primitive functions (e.g., p1, p2, abs, rad(_), pow(_), ...) and of particular subexpressions (e.g., sums of radicals, quotients and products), through a predicate user_rate/2, used by rate_restr/3.

```
rate_restr(T,Rate,L) :- nonnegative(L), user_rate(T,Rt),
   sum([Rt|L],#=,Rate).
```

The parameter CountTypes is a list of finite domain variables, each one giving the number of occurrences of a given type and the user can define constraints on the values of these counters. Such constraints are imposed by constrs/2 and can involve a single variable (e.g., to specify its domain), or any subset of them, and are specified by predicates.

```
constrs(CTs,Functor) :- Goal =.. [Functor,CTsConstr], call(Goal),
   single_vars_low_up_constrs(CTsConstr,CTs),
   user_other_restrs(Functor,CTs).
```

The `Goal` is a user-defined predicate that instantiates `CTsConstr` to a list of terms of the form `I-[Low,Up]`. This list is passed to `single_vars_low_up_constrs/2` to add new constraints on the lower and upper bound values of the variable associated to key `I`. The definition of `user_other_restrs/2` can be less trivial, and may be used to state more complex constraints on subsets of the variables. Still, the configuration constraints are usually simple value or arithmetic constraints, lower and upper bound cardinality constraints or conditional constraints, such as $l \leq x_i \leq u$, $l \leq \sum_{i \in I_1} x_i \leq u$, and $\sum_{i \in I_1} x_i \geq l \Rightarrow \sum_{i \in I_2} x_i \leq u$, where x_i denotes an integer variable, usually a counter.

From the definition of `prodstype/5`, we can see that the underlying CSP model is not a classic model, with a fixed static collection of variables and constraints. This happens very often in configuration problems [4,7]. In our application, new variables and constraints are added during the execution (e.g., the ones corresponding to `RateB`, `RateT`, `OpsB`, `OpsT`, `Gt`, `Gb`, and some variables in the lists `CTsB` and `CTsT`). We are not using global constraints in our application, although domain filtering algorithms for open global cardinality constraints have been investigated [9,19], for some dynamic CSPs. Considering the underlying execution model and the fact that the CSP model is rather dynamic, that will result in a burden without any payoff.

The expression generator produces a file of instances of expressions and their types. A call to `examples(File,DegreeI,RateMin-RateMax,X,NumbInst)` yields a `File` of expressions in the variable `X`, with `NumbInst` instances of each type. The degree of the expressions is `DegreeI` (and can be undefined) and the difficulty level is within `RateMin-RateMax`, which are positive integers.

In AGILMAT, such a file can be passed to the *exercise generator and solver* to create a sheet of exercises and their one-line solutions. We developed specific solvers to be able to handle some nonlinear constraints (in a real-valued variable) and compute exact solutions. A numerical approximation would not be a correct answer usually. Our solvers perform symbolic manipulations and the rules applied emulate steps students may take.

3 Extensions for AgilmatQuiz

The major extensions carried out in this work involved the two main modules, and consisted of:

- the definition of new forms of expressions;
- the modification of the symbolic solver to produce plausible wrong answers;
- the definition of new exercises and strategies for choosing the wrong answers.

In the design of AGILMAT and of this extension, we kept in mind the basic principle that we do not need to support full generality in order to obtain an useful tool. This allows us to partially circumvent some inherent theoretical difficulties of this work, including the ones due to the undecidability of some computer algebra problems [2,10]. We observe that, very often, the topics focused on in the literature are comparatively very *well-behaved*, with well-known canonical forms and solving algorithms (e.g., elementary school arithmetics [22], operations with fractions, linear constraints in a single variable, systems of linear equations, and so forth).

Since the type of the expression acts as a template, the AGILMAT prototype can be used to compute one or more expressions of each type. This feature makes easier the creation of several instances of the same question. This is useful for obtaining multiple-choice tests of the same difficulty level, although this is not too important or even desirable when the tests are used for self-assessment. The separation of the generation of types (templates) from the generation of the instances of the expressions provides the flexibility we need to deal with these two cases. By enforcing constraints on the difficulty level of the types of expressions, we can control the expressions that occur in the questions. This feature is important for multiple-choice questions since, usually, the student must find the answer to the question in a short time.

Besides some simple adjustments, such as the ones needed to create questions about conjunctive and disjunctive constraints, we focused on the generation of expressions for piecewise-defined functions. This is more challenging than the generation of simple expressions, as we need to split the function domain and control the way the different branches fit or do not fit.

3.1 Creating Expressions for Piecewise-Defined Functions

Although we could look at this problem as a constraint problem, devising its solution could be tricky, because we also have to create expressions that are interesting from the pedagogical point of view. This means that the numbers arising in the expression and the breaking points cannot be too scary.

We extended the generator to include a new type `piecewise`(L), where L is the list of types of the branches. For the corresponding expression, we use a similar notation except that each branch is identified by a term `br`(*Expr*, *DomExpr*). Below, we show an example of a type and an expression of that type. Actually, the expression is still a *partial expression* as the domain of each branch is a free variable, represented by the underline character (adopting the Prolog notation).

```
piecewise([abs o p1 o x,p1 o x]).
piecewise([br(abs(pol(x,[-2,-1])),_),br(pol(x,[-5,-4]),_)]).
```

$$\begin{cases} |-2x-1| & \text{if } x \in ? \\ -5x-4 & \text{if } x \in ? \end{cases}$$

We decided to fix the coefficients of the functions first and then fix the domain of each branch, taking into account the points where the functions meet (although it is possible that the branches do not meet for some other functions). This made the extension of the generator easier. Besides and more importantly, in this way we avoid cumbersome coefficients in the resulting expressions, and therefore obtain expressions that resemble the ones defined by teachers. The quality and variability of questions created by AgilmatQuiz was one of the issues that our colleagues appreciated.

For affine and quadratic functions, the candidate breakpoints can be computed easily by solving an equation. For some other functions, to be able to guarantee that the resulting function is continuous at a breakpoint, we often need to compute lateral limits and, quite likely, to replace some coefficients (e.g., of the constant branches). For instance, $f_k(x)$, defined below, is continuous iff the constant k is zero, since $\lim_{x\to 1^+} f_k(x) = 0$.

$$f_k(x) = \begin{cases} \frac{x-1}{\sqrt{x-1}} & \text{if } x > 1 \\ k & \text{if } x \leq 1 \end{cases}$$

For the example given above, the complete expression yielded by the system was the following one, which means that the two branches actually meet.

```
piecewise([abs o p1 o x,p1 o x]).
piecewise([br(abs(pol(x,[-2,-1])),[a(-(infty)),a(rat(-1,1))]),
           br(pol(x,[-5,-4]),[f(rat(-1,1)),a(infty)])]).
```

$$\begin{cases} |-2x-1| & \text{if } x \in]-\infty,-1[\\ -5x-4 & \text{if } x \in [-1,\infty[\end{cases}$$

It is possible that the system selects another breakpoint. With some probability, fixed in the implementation, the points where the functions meet would not be selected as breakpoints. In our current implementation, the search for breakpoints that may guarantee continuity is supported for affine and quadratic functions, for example, but we are not reasoning about limits. For the remaining functions, the implementation ensures that the domain of every branch is non-empty by defining breakpoints in the intersection of the domains of the primitive functions, preferentially.

In the implementation, the difficulty rate of a piecewise-defined function is determined by the difficulty rate of the branch that has the highest rate and the total number of branches. A constraint is imposed on the weight of each new branch, so that it does not exceed the half of the previous one.

3.2 Solutions and Distractors

For generating a set of plausible wrong answers (i.e., distractors) for a quiz question, we modified the symbolic solver and some of its submodules to include buggy algebraic rules that translate known common errors or misconceptions.

For that purpose, we augmented the signature of some of the predicates with a new argument that acts as a constrained variable. By imposing constraints on this variable we can restrict and track the number of wrong rules applied in the derivation of an answer. In this way, the same predicate can be used to compute the correct solution if we restrict the value of the variable to be zero. To explain better what we mean, we give a fragment of the initial implementation of the predicate `domain_expr(Expr,X,Dom,DomExpr)`, which determines the domain `DomExpr` of an expression `Expr` in the variable `X` when `X` could only take values in the subset `Dom` of the real numbers.

```
domain_expr(X,X,Dom,Dom) :- !.
domain_expr(pol(U,_L),X,Dom,Domf) :- !, domain_expr(U,X,Dom,Domf).
domain_expr(rad(U,N),X,Dom,Domf) :-  !,
   (even(N) ->
      (domain_expr(U,X,Dom,DomU),solve(DomU,U,geq,rat(0,1),X,Domf));
       domain_expr(U,X,Dom,Domf)).
```

The first rule is equivalent to saying that the identity function defined in the set `Dom` has domain `Dom`. The second rule basically says that the domain of the composition of a polynomial function and a function `U` of `X` is the domain of `U` in `Dom`. Finally, the third rule defines the domain of $\sqrt[N]{U}$ either as `DomU` if `N` is odd or as the solution set of $U \geq 0$ in `DomU` if `N` is even.

For the new version, `wrg_domain_expr(Expr,X,Dom,DomExpr,W)`, we added an extra argument `W`, that is a constraint variable, and add extra rules, which mimic frequent errors, known by experienced teachers.

```
wrg_domain_expr(X,X,Dom,Dom,W) :- {W = 0}.
wrg_domain_expr(X,X,Dom,[a(-infty),a(infty)],Wf) :- !,
   {Wf = 1},  Dom \= [a(-infty),a(infty)].
wrg_domain_expr(pol(U,_L),X,Dom,Domf,Wf) :-
   {Wf >= 0},  wrg_domain_expr(U,X,Dom,Domf,Wf).
wrg_domain_expr(pol(U,_L),X,Dom,Domf,Wf) :- !,
   {Wf = 1, Ok = 0},
   wrg_domain_expr(U,X,Dom,DomS,Ok),
   DomS \= [a(-infty),a(infty)], Domf = [a(-infty),a(infty)].
wrg_domain_expr(rad(U,N),X,Dom,Domf,Wf) :- even(N), !,
   {Wi >= 0, Wii >= 0, Wf = Wi+Wii},
    wrg_domain_expr(U,X,Dom,DomU,Wi),
   ( wrg_solve(DomU,U,geq,rat(0,1),X,Domf,Wii);
      ({Wii = 1}, Domf = DomU) ).
wrg_domain_expr(rad(U,_N),X,Dom,Domf,Wf) :- !, % N is odd
   {W >= 0, Wi >=0, Wf = W+Wi},
   wrg_domain_expr(U,X,Dom,Domi,W),
   ((({Wi=0}, Domf = Domi);
     ({Wi=1}, Domf = Dom, Domi \= Dom);
     ({Wi=1+Wii,Wii>=0}, wrg_solve(Domi,U,geq,rat(0,1),X,Domf,Wii))).
```

The constraint variable[4] is used to control the number of buggy derivation rules applied. If `W` is bounded to be zero, the predicate produces the correct answer, as before.

Now, the second rule is buggy since it ignores the given domain and yields $\mathbb{R}$ as the answer (the representation of $\mathbb{R}$ is `[a(-infty),a(infty)]`). For instance, if the stem asks for "the domain of $f : \mathbb{R}_0^+ \to \mathbb{R}$ given by $f(x) = 3x + 1$", the correct answer is $\mathbb{R}_0^+$ and a derivation applying the third clause and then the second one produces $\mathbb{R}$.

The fourth rule is buggy for a similar reason, as it ignores the restrictions imposed both by `Dom` and the domain of `U`, and returns $\mathbb{R}$ as the answer. For instance, for $g : \mathbb{R} \to \mathbb{R}$ given by $g(x) = 5\sqrt{x-4} + 3$", the answer is $[4, \infty[$ but a derivation using the fourth clause yields $\mathbb{R}$.

In the fifth clause, the last branch is buggy and can produce a wrong answer (e.g., $\mathbb{R}$ for $\sqrt{x-4}$ instead of $[4, \infty[$). Finally, the last rule is buggy and produces a distractor if the solution set of $\texttt{U} \geq 0$ in `Domi` differs from the correct answer (e.g., for $\sqrt[3]{x-4}$, if `Dom` is $\mathbb{R}$, the correct answer is $\mathbb{R}$ and not $[4, \infty[$). If the problem asks for the solutions of, e.g., $\sqrt{1 + \sqrt[3]{x-4}} = 0$, the application of this buggy rule can lead to a wrong solution.

In our experiments, for finding distractors, we often bound `W` to 1 when the predicate is called. In this way, we try to focus on more plausible distractors, resulting from a single error, and discard options that can give clues for looking somehow more absurd. Depending on the results of the computations, when `W` is not 0, the predicate can produce the correct answer as if it were a wrong answer. The set of all wrong solutions, computed by backtracking, is filtered out afterwards to discard repetitions and the "solutions" that are equal to the correct one. In a multiple-choice question about domains, the distractors for the question are selected from this final list, and possibly the item "`None of the other ones`". With some probability, this item can replace the correct solution also.

In general, the exact comparison of solutions is not straightforward, and can be undecidable [2,10]. In the implementation, we defined a canonical form for some expressions and constants and for the restricted sets manipulated by the system (which are unions of a finite number of intervals and isolated points), as in our previous work [17]. Our solver is not complete, as a domain reasoner. For comparing more complex constants (clearly, not rational numbers), the system sometimes performs a numeric comparison, after evaluating the constants as floating point numbers. In practice, by limiting the number of flaws to 1, we obtain more plausible distractors that are helpful for identifying a student error or misconception. In addition, we reduce the risk of finding equivalent solutions that are not syntactically equal (e.g., $2\sqrt{7+4\sqrt{3}}$ and $4+2\sqrt{3}$), by avoiding many alternative derivations and unnecessary computations that may yield complex

[4] The AGILMAT solver used the CLP($\mathbb{Q}$) module for supporting computations with rational numbers. This is the reason for W being not treated as a finite domain variable, as that avoid some (re-)implementation effort. We plan to fix that in a future version of the prototype.

constants. On the other hand, some more powerful procedures available in current computer algebra systems (e.g., Mathematica, Maple, Maxima, etc, to name a few) are too advanced for high-school or undergraduate students. Hence, given that our system can produce a huge number of exercise instances, the system can discard the ones it cannot solve exactly. Nevertheless, this is an issue that requires further research to understand the limits and possible improvements of our approach although, it is know that, in general, canonical forms cannot exist [10].

It is worth mentioning that there are other e-learning tools that make use of buggy rules for creating exercises or for diagnosis [21,22]. SLOPERT, for instance, is a reasoner and diagnoser for symbolic differentiation, developed in Prolog, used as a domain reasoner by LeActiveMath and MathBridge [5,21]. It is enriched with buggy derivation rules, implemented as clause predicates as in AgilmatQuiz, each one being annotated as `buggy` (wrong) or `expert` (correct) rule. A parameter keeps track of the history of the derivation, but there is not the same support for imposing constraints on the number of buggy rules that can be applied in a derivation.

3.3 New Exercises and Strategies for Selecting Distractors

The set of exercises was also enriched and the corresponding solving procedures were implemented. As we observe above, we try to create exercises that make some sense. This means that we sometimes can exploit the relationship between some notions, e.g. range and reciprocal image, to obtain exercises that are pedagogically acceptable. For instance, for creating the exercises about the notion of reciprocal image $f^{-1}(D)$, the system selects D as a subset of the range of f, and so it first tries to compute that range.

Poorly written distractors for a multiple-choice question can invalidate the question. Finding good distractors can be difficult even for teachers. For distractor development, we tried to attend to the following rules: "use plausible distractors; avoid illogical distractors; incorporate common errors of students in distractors; use true statements that do not correctly answer the item" [6]. Since the choices we consider are either wrong solutions or the correct one, we interpret the last goal as the inclusion of solutions that overlap, when the answer is a set.

Another guideline is to avoid or use sparingly "None of the above" [6]. However, the inclusion of this choice in some problems, either as distractor or correct answer, was a requirement from our colleagues (because the choices are shuffled by Mooshak, actually we use "None of the other ones" instead). This choice was intentionally used to make the guess of the correct answer by a simple analysis of the offered answers more difficult. We agree that this can be relevant for Mathematics. For instance, when the correct answer is a set, students may have to work out the solution if the other choices cannot be trivially discarded.

To prevent correct answers from being trivially guessed, the system tries to classify the distractors, for instance in terms of their intersection with the correct solution or the number of derivations that led to each one. This classification is sometimes used to introduce some bias on the selection procedure. In the implementation, the distractors are selected randomly for each problem, and the preference for distractors obtained by some rules can increase their weight. For instance, in problems about the reciprocal image $f^{-1}(D)$, a common error is the interpretation of f^{-1} as $1/f$. Distractors produced by the buggy rule that translates this misconception were given some higher weight. The number of times a distractor occurs is used in this case also.

The criteria for the selection of distractors from the computed list is an issue that deserves further investigation. In particular, a more accurate model for defining the preferences for some rules could be designed.

4 Quiz Delivery

In this section, we give further details on how the system is used to produce a quiz in the `mooshakquiz` style and how the quiz is delivered to students.

Mooshak Quizzes. Mooshak is a web based competitive learning system originally developed for managing programming contests [8]. It is used as an e-learning tool in several universities. Quiz delivery started as a complement for evaluation in programming courses, giving support for multiple-choice questions. It generates quizzes by randomly selecting questions and shuffling them and their items. Quizzes are graded automatically. Each correctly answered question adds its mark to the final grade. Incorrect answers are penalized so that a series of random answers to the quiz questions has an expected grade of zero. The system provides overall statistics per question. Quizzes are structured in groups. The number of questions in a group may be larger than the number of questions actually presented to students. A group may be regarded as a question bank. If this bank is created by AgilmatQuiz from a single template expression, the tests produced by selecting a single question from each group have similar difficulty level.

Quiz Format. Mooshak and AGILMAT use different formats. Mooshak uses its own XML based format to import and export data. AGILMAT uses LaTeX as output format. To make the two systems interoperable with each other the main issue was the definition of a common quiz format. A natural candidate for this role is the Question & Test Interoperability (QTI) standard. This approach would require some implementation effort on the AGILMAT side. Moreover, QTI with MathML would be inappropriate for humans. Teachers must be able to produce quizzes in the selected format, as certain exercise types needed for the PreCalculus course are not yet covered by AgilmatQuiz.

The final decision was to create a new LaTeX based format for quizzes – the `mooshakquiz` style. The quiz is defined as a document structured by LaTeX environments defining groups, questions and choices. These environments are configured by parameters, such as the number of questions extracted from each group (typically 1) or the logical values of a choice (true or false). Text in these environments may contain math expressions. A quiz in this format may be processed as a LaTeX document to produce an handout in PDF format, for instance.

On the AgilmatQuiz side, a predicate groups(p(*Pred*, *Args*, *Nq*)−*File*) defines the groups of exercises to create. Here, *Pred* is the name of the predicate that will generate the corresponding group using the expressions already saved in the file *File*. The definition looks like the following one.

```
groups(p(quizReciprocal,3)-p1)
groups(p(quizConjDisj,2,1)-p1).
groups(p(quizDomains,1)-rad1simple).
```

The sequence *Args* is optional and defines parameters to this predicate *Pred*. Finally, *Nq* defines the number of questions that will be selected from each group to a test. The system produces a LaTeX file in the mooshakquiz style, that is used for quiz delivery.

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage{mooshakquiz}
\begin{document}
\begin{quizgroup}{3}
\begin{quizquestion} Find $\displaystyle t^{-1}(]1,\infty[)$
            for $\displaystyle t :\mathbb{R}\rightarrow\mathbb{R}$
            given by $ \displaystyle t(x) = -6\,x-1$ \newline
\begin{quizchoice}{false}$ \displaystyle ]-\infty,-3[ $  \end{quizchoice}
\begin{quizchoice}{true}$ \displaystyle ]-\infty,-\frac{1}{3}[ $
                                   \end{quizchoice}
\begin{quizchoice}{false}$ \displaystyle ]-\infty,-8[ $  \end{quizchoice}
\begin{quizchoice}{false}$ \displaystyle ]-\infty,0[ $  \end{quizchoice}
\end{quizquestion}
\begin{quizquestion} Find $\displaystyle t^{-1}(]4,\infty[)$
            for $\displaystyle t :\mathbb{R}\rightarrow\mathbb{R}$
            given by $ \displaystyle t(x) = 2\,(x-4)$ \newline
\begin{quizchoice}{false}$ \displaystyle ]-\infty,6[ $  \end{quizchoice}
\begin{quizchoice}{false}$ \displaystyle ]-1,\infty[ $  \end{quizchoice}
\begin{quizchoice}{false}$ \displaystyle ]-4,\infty[ $  \end{quizchoice}
\begin{quizchoice}{true}$ \displaystyle ]6,\infty[ $  \end{quizchoice}
\end{quizquestion}
...
\end{quizgroup}
\begin{quizgroup}{1} ... \end{quizgroup}
\begin{quizgroup}{1} ... \end{quizgroup}
\end{document}
```

Quiz Processing. Mooshak required minor changes to process quizzes. The function that imports quizzes in the mooshakquiz style converts the environment based structure of the document to XML, leaving text and math expressions unchanged. The document is then imported to the internal representation of Mooshak and processed as regular quiz, with text and math expressions inserted in HTML pages and presented on a web browser. Math expressions in LaTeX are converted on-the-fly in the browser using the MathJax [3] JavaScript display engine, which was crucial for a quick implementation.

5 Conclusions and Future Work

This work describes an approach for generating and delivering math quizzes using constraint logic programming. The main contribution is a novel approach for creating multiple-choice questions with a set of plausible wrong answers. We focused on a particular type of multiple-choice questions, but our approach could be exploited to support the generation of other types of questions or populate question repositories (if the output is written in some more standard exercise language). The work is an example of an application where the use of declarative languages was crucial for a rapid development of an useful tool. At the current stage, the generator and solver consist of about 7000 lines of code. But, it is not very easy to quantify the overall development effort of AgilmatQuiz, as it is an extension of AGILMAT. Our crude estimate is of about one month-person for the reported extension. We plan to improve the implementation and cover other topics. Constraint programming makes easier the re-usability and customization of the system. However, it would be interesting to study execution models where the propagation of constraints plays a more significant role in the program transformation. The prototype is currently used to support a remedial PreCalculus course for students entering the Faculty of Sciences at the University of Porto. A formal evaluation is planned. It would be already possible to draw some conclusions if we check whether experienced teachers can separate the exercises produced automatically from identical ones produced manually.

Acknowledgments. The authors would like to thank their colleagues responsible for the PreCalculus course and anonymous reviewers for helpful comments.

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, pp. 89–115. Springer, Heidelberg (1998)
2. Bradford, R., Davenport, J.H., Sangwin, C.J.: A Comparison of Equality in Computer Algebra and Correctness in Mathematical Pedagogy. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Calculemus/MKM 2009. LNCS (LNAI), vol. 5625, pp. 75–89. Springer, Heidelberg (2009)
3. Cervone, D.: MathJax – A Platform for Mathematics on the Web. Notices of the AMS 59, 312–316 (2012)
4. Faltings, B., Macho-Gonzalez, S.: Open Constraint Programming. Artificial Intelligence 161, 181–208 (2005)
5. Goguadze, G.: ActiveMath – Generation and Reuse of Interactive Exercises using Domain Reasoners and Automated Tutorial Strategies. PhD thesis, Saarland University (2011)
6. Haladyna, T.M., Downing, S.M.: A Taxonomy of Multiple-Choice Item-Writing Rules. Applied Measurement in Education 2, 37–50 (1989)
7. Junker, U.: Configuration. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, pp. 835–871. Elsevier (2006)

8. Leal, J.P., Silva, F.: Mooshak: a Web-based Multi-site Programming Contest System. Software – Practice and Experience 33, 567–581 (2003)
9. Maher, J.M.: Open Contractible Global Constraints. In: 21st International Joint Conf. on Artificial Intelligence, IJCAI 2009, pp. 578–583. Morgan Kaufmann Publishers, USA (2009)
10. Moses, J.: Algebraic Simplification: a Guide for the Perplexed. Communications of the ACM 14, 527–537 (1971)
11. Pinto, J.S., Oliveira, M.P., Anjo, A.B., Vieira Pais, S.I., Isidro, R.O., Silva, M.H.: TDmat-Mathematics Diagnosis Evaluation Test for Engineering Sciences Students. Int. J. Mathematical Education in Science and Technology 38, 283–299 (2007)
12. Sangwin, C.J., Grove, M.J.: STACK – Addressing the Needs of the "Neglected Learners". In: 1st WebAlt Conference and Exhibition, pp. 81–95 (2006)
13. Sangwin, C.: Computer Aided Assessment of Mathematics. Oxford University Press (2013)
14. Schoenfeld, A.H. (ed.): Assessing Mathematical Proficiency. Cambridge University Press (2007)
15. Snajder, J., Cupic, M., Basic, B.D., Petrovic, S.: Enthusiast: An Authoring Tool for Automatic Generation of Paper-and-Pencil Multiple-Choice Tests. In: ICL 2008, Villach, Austria (2008)
16. Sterling, L., Bundy, A., Byrd, L., O'Keefe, R., Silver, B.: Solving symbolic equations with Press. Journal of Symbolic Computation 7, 71–84 (1989)
17. 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)
18. Tomás, A.P., Leal, J.P., Domingues, M.: A Web Application for Mathematics Education. In: Leung, H., Li, F., Lau, R., Li, Q. (eds.) ICWL 2007. LNCS, vol. 4823, pp. 380–391. Springer, Heidelberg (2008)
19. van Hoeve, W.-J., Régin, J.-C.: Open Constraints in a Closed World. In: Beck, J.C., Smith, B.M. (eds.) CPAIOR 2006. LNCS, vol. 3990, pp. 244–257. Springer, Heidelberg (2006)
20. Xiao, G.: WIMS – An Interactive Mathematics Server. Journal of Online Mathematics and its Applications 1, MAA (2001)
21. Zinn, C.: Supporting Tutorial Feedback to Student Help Requests and Errors in Symbolic Differentiation. In: Ikeda, M., Ashley, K.D., Chan, T.-W. (eds.) ITS 2006. LNCS, vol. 4053, pp. 349–359. Springer, Heidelberg (2006)
22. Zinn, C.: Program Analysis and Manipulation to Reproduce Learners' Erroneous Reasoning. In: Albert, E. (ed.) LOPSTR 2012. LNCS, vol. 7844, pp. 228–243. Springer, Heidelberg (2013)
23. LeActiveMath: Language-Enhanced, User Adaptive, Interactive eLearning for Mathematics, EU project (2004–2006), `http://www.leactivemath.org/`
24. Math-Bridge: European Remedial Content for Mathematics, EU project (2009–2012), `http://www.math-bridge.org/`
25. PmatE – Mathematics Education Project. University of Aveiro, Portugal (1990), `http://pmate4.ua.pt/pmate/`
26. SICStus Prolog. SICS, Sweden, `http://www.sics.se`
27. STACK: System for Teaching and Assessment using a Computer algebra Kernel. University of Birmingham, UK, `http://www.stack.bham.ac.uk/`