

Generating Mathematics Drills Automatically*

Ana Paula Tomás, José Paulo Leal, and Pedro Vasconcelos

DCC-FC & LIACC, Universidade do Porto,
R. do Campo Alegre, 823, 4150-180 Porto, Portugal
{apt,zp,pbv}@ncc.up.pt

Keywords: Automatic Generation of Drills, Computer-based training, Constraint Programming

Topics: AI Foundations and Knowledge Representation, Constraint Satisfaction, Planning and Scheduling, AI in Education and Intelligent Tutoring Systems.

Abstract. We propose a methodology for designing online exercises systems with special focus on applications to Mathematics education. The major goal is to develop a web-based environment that make available exercises and solutions to students (and teachers). Ongoing study aims at investigating whether Constraint Logic Programming frameworks are adequate to implement such a system. Encouraging results are reported in this paper, indicating that these languages have the right expressiveness to encode control on the system in an elegant way.

1 Motivation

Not all students have high mathematical skills but surely one of the reasons for the lack of success in mathematics is that too often students merely memorize how to solve some exercises, instead of trying to understand the fundamental concepts and results. Hence, a possible drawback of classical textbooks and some existing online course-ware and exercise systems is that the proposed problems are quite pre-defined, either fixed or at best randomly generated instances of the same problem template [4, 5].

Rather than to reproduce the classical textbooks, advances in the computer technology and the Internet should be exploited to develop really interactive and re-usable contents. Quite sophisticated web-based learning environments are being developed. For example, ActiveMath [14], that is a second-generation interactive textbook project offering user-adaptiveness and reusability by employing an XML-based representation of mathematical knowledge and Artificial Intelligence techniques. Alike [4], it supports exploratory learning through communication with mathematical systems.

Commercial mathematical systems, as *Geometer's Sketchpad* [6], *Maple* [11] and *Mathematica* [13], just to name a few, are often used as mathematics tools

* Work partially supported by funds granted to LIACC through *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia* and *Programa POSI*.

for exploratory learning [8, 15], enabling the students to try their own examples. Some already offer access to their applications through web browsers [13]. The focus of this paper is not on problem solving in the broader sense of exploration, but rather on the repetitive drills students have to do for consolidation of concepts and practice of algebraic procedures. For constructive learning to be effective, students need self-confidence and also basic knowledge.

Lots of mathematics teaching resources are spread over the Web, namely web-based systems for computer aided training and/or assessment, with authoring facilities for teachers to create question files, for example, for homework and assignments (e.g. [1, 5, 9, 14]). This requires non-negligible effort from the teacher, specially to generate problem instances that are not immediately recognized as simple variants of a few basic expressions. In fact, for all the systems we know, the exercises are not generic enough and the user can almost anticipate the form of the next instance of the problem, after a while.

This paper reports on our experiences in using a computer algebra system (`Maple`) and Constraint Logic Programming (CLP) frameworks to automatically generate online examples and exercises for teaching and learning a topic in mathematics. Our final goal is to develop a system that dynamically computes a wide range of examples that really look different for students, despite they naturally obey some given specification. Moreover, the system shall optionally provide explanations that may help students improve their ability to express coherently in mathematical language.

Our approach has many different potentials that include user-adaptiveness, easy definition of several curricula, and possible integration in intelligent tutoring systems. The methodology we adopt to characterize an application domain mimics the one teachers usually follow when they try to formulate basic problems in some context. Firstly we have to define and represent the forms of exercises that may be solved by the procedures that students shall learn. Secondly, the exercises must have pedagogical interest, so that we must have some idea of their solutions. The best strategy is then to proceed from an intended solution and/or solving procedure to formulate a problem instance. In this way, we may also ensure that the generated problems are solvable by the computational system (and, hopefully, by the student at a given level), thus avoiding undecidability issues. Although not all the topics taught in mathematics at high school allow such an automatic treatment, a considerable number do. Many of the questions that students have to work out in Mathematics courses may successfully be solved by algebraic procedures. If these procedures are also implemented, the generated problems can be completely solved and the solving steps explained, in contrast to systems that implement deduction schema, as theorem provers.

This idea is also implicit in a recent work by Sangwin [16], although the emphasis there is on how to generate exercises that get the students to construct instances of mathematical objects with some properties. How to reduce the teachers' effort to prepare questions is not considered at all and, moreover, it is assumed that they have some expertise in writing computer programs. Actually, it is examined the application of AIM [9], which is an authoring system for

computer aided assessment that ultimately uses Maple to process the exercises but that counts on the teacher to program the exercises and in some situations their grade scheme. This is quite different from what we have in mind.

Although we expected that the use of computer algebra systems could highly reduced our implementation effort, our experience (with Maple) has shown that the algebraic simplifications may turn out troublesome. As we shall detail in the following section, some additional constraints shall be imposed, for instance, on the expressions that arise in the exercises, to avoid inconsistencies in the explanations that are produced.

This paper is intended to present the results we have achieved so far. To illustrate the main ideas we refer to a particular topic in Introductory Calculus, giving, in Section 3, a grammar that characterizes a vast sample of examples from some high school textbooks. This grammar shall still be extended to cover other functions taught. The interesting point is that we now may get specialized forms of the expressions almost for free, by adding further restrictions through constraints. This is of great importance for educational purposes since the system must be parametrized to easily cater for different curricula. In the following section, we discuss strengths and weaknesses of our first attempt to implement some programs to generate problems and examples using a computer algebra environment. The need for a more declarative framework, led us to investigate the use of CLP, that offers natural support for possible user-defined constraints on the expressions. Section 4 describes aspects of our prototype implementation and briefly address interface issues. Programs that have been developed as a testbed for some of the ideas may be download from <http://www.ncc.up.pt/~apt/demomath.html>.

2 Some Experiments Using a Computer Algebra System

In this section, we discuss some pros and cons of using Maple to develop interactive course-ware, which may be common to other computer algebra systems. Our previous work involved the design of Maple worksheets to present some specific topic in mathematics. Besides some concise notes on the addressed issue, such worksheets typically include pointers to other ones where the end-user student may find randomly generated examples and exercises to work on. They have been developed in the scope of a course on Computers in Education, some of them by undergraduate Computer Science students.

Some of the algebraic procedures that students learn are crucial to different problems. For example, in introductory calculus, the analysis of the sign variation, zeros and domain of real-valued functions is a basic tool to find intervals where a function is monotonic, to study concavity and convexity for twice differentiable functions and to sketch their graphs. But, it is quite easy to define functions for which no generic algorithm exists to compute accurately their zeros, as shown independently by Abel and Galois, about a century ago, for polynomials of degree greater than four. In the following example, we consider the generation of rational functions defined by a quotient of two polynomial

expressions so different as the following ones.

$$\frac{(-x - \sqrt{5})^5}{x^5 - 4x^4 + 6x^3} \quad \frac{(-x^3 - x^2)}{(x+2)^3} \quad \frac{(x+1)^4(x^5 + 3x^4 + 2x^3)}{(x^2+1)^2(-2x-2-\sqrt{7})}$$

Example 1. We shall present some output from one of our **Maple** programs that is helpful for the discussion. Some typesetting has in fact been done for space reasons.

```
> domains(true);
FIND THE DOMAIN OF THE FUNCTION f DEFINED BY
```

$$f(x) = \frac{(8x^2 + 14x - 15)(2x + 1)}{(4x^6 - x^5 - 5x^4)(3x^2 - 17x + 10)^2}$$

```
SOLUTION: Being f a rational function, it is defined for all
real numbers except the zeros of the denominator of its expression.
We have
```

$$(4x^6 - x^5 - 5x^4)(3x^2 - 17x + 10)^2 = 0$$

```
if and only if  $4x^6 - x^5 - 5x^4 = 0$  or  $(3x^2 - 17x + 10)^2 = 0$ .
As concerns  $4x^6 - x^5 - 5x^4 = 0$ , we have
```

$$\begin{aligned} 4x^6 - x^5 - 5x^4 = 0 &\Leftrightarrow x^4(4x^2 - x - 5) = 0 \\ &\Leftrightarrow x = 0 \vee 4x^2 - x - 5 = 0 \end{aligned}$$

```
To solve  $4x^2 - x - 5 = 0$ , we apply the solving formula for polynomial
equations of degree 2, the roots being -1 and 5/4.
```

```
As concerns  $(3x^2 - 17x + 10)^2 = 0$ , we have
```

$$(3x^2 - 17x + 10)^2 = 0 \Leftrightarrow 3x^2 - 17x + 10 = 0$$

```
To solve  $3x^2 - 17x + 10 = 0$ , we apply the solving formula for
polynomial equations of degree 2, the roots being 2/3 and 5.
We conclude that all real numbers are in the domain of f, but
2/3, 0, -1, 5 and 5/4.
```

As other computer algebra systems, **Maple** supports polynomial expressions and thus it is easy to implement this procedure. The main issue is how to control the difficulty level of the problems. The idea is to abstract their form from the rules students could use to solve them. For educational purposes, we also need to have some control on the generated polynomial expressions, so that the exercise may have pedagogical interest. Instead of simply using the builtin **Maple** procedure to generate random polynomials, the computation of $f(x)$ was driven by the selection of the set of roots, which might be rational and (conjugated) irrational numbers. Factors with no real roots were obtained by adding appropriate constants to quadratic polynomial expressions with real roots to shift their representing parabolas upwards or downwards so that every intersection with the

horizontal axis is eliminated. Both the denominator and numerator are factored and the factors may be the following basic forms: $ax + b$, $ax^2 + bx + c$, $(ax + b)^n$, $(ax^2 + bx + c)^n$ and $ax^{n+1} + bx^n$, $ax^{n+2} + bx^{n+1} + cx^n$, where $a, b, c \neq 0$. The idea is that the student has to know how to solve linear and quadratic equations, $X^n = 0$ and that $ax^{n+1} + bx^n = x^n(ax + b)$ and $ax^{n+2} + bx^{n+1} + cx^n = x^n(ax^2 + bx + c)$. In this phase, we discarded expressions as $(ax^{n+2} + bx^{n+1} + cx^n)^m$, because we did not think they are of great pedagogical interest.

An important point that deserves some further research is how to improve the linguistic quality of the output explanations. It is not trivial to obtain explanations in natural language by annotating the programs. In this example, almost no use was made of global context information, which renders the explanations fairly repetitive and, therefore, unnatural or pedagogically poor. This seems to apply also to intelligent tutoring systems. Besides the need for a more flexible input/output interface, three other remarks have played a major role in our decision to try a different platform. The first one concerns the algebraic manipulations that **Maple** automatically performs, which may result in unpredictable simplifications of the expressions being operated. This feature appears as a great advantage when compared to Logic Programming systems, but is surely a serious drawback for our intended usage of the system. Actually, it may introduce some puzzling inconsistencies in the output explanations. For instance, it is not possible to print $3(x^2 + 5)$ in **Maple** since it will naturally yield $3x^2 + 15$. By a similar reason we had better not ask the student to find the domain of a rational function defined by $f(x) = (x - 1)^2 / (x - 1)$ because that expression would be printed as $f(x) = x - 1$, and 1 hence belongs to domain of the latter but not of the former one. In Example 1, simplifications were avoided by further restricting the types of the generated rational functions $f(x)$ to disallow repetitions of factors (either in a product or quotient) and to require that the involved polynomials just have integer coefficients. Since we would like to cover more general expressions, this does not seem the right way to proceed. The second point is that we need declarativeness to help specify the possible form of the expressions. Finally, we would like the application to be well parametrized to cater for different curricula. For both these aspects, CLP seems to offer the right expressiveness to encode control on the system in an elegant way. The disadvantage is that we now have to implement symbolic processing of algebraic expressions to provide exact representations of the solutions, which hopefully have quite simple pre-defined forms. It is worth mentioning that CLP languages are rather adequate for symbolic processing, all one needs is either to spend sometime implementing a symbolic processor or to find and reuse some existing one. Difficulties have also appeared when we tried to combine different constraint solvers, since it is almost impossible to share variables between them in a natural way.

We shall now elaborate on abstract representations for the expressions, that we need to characterize the problem templates and to simplify the solving procedures. For that purpose, we give a grammar that characterizes a wide range of the function expressions that may be found in high school textbooks and whose

zeros can be exactly computed. This grammar extends the set of functions we considered in Example 1.

3 A Case Study – Calculus for Pre-University Levels

In order to be able to abstract the possible forms of function expressions, we have carried out a thorough analysis of Portuguese textbooks in mathematics for grades 10 to 12. As a result, we defined the grammar shown in Fig. 1.

For prototyping, the trigonometric, exponential and logarithmic functions have been left out. Basically, with this grammar we try to capture some of the expressions for which the computation of the domain and zeros mainly involves solving linear or quadratic equations ($ax+b=0$ or $ax^2+bx+c=0$), or equations of the form $aX^n+b=0$, $a\sqrt[n]{X}+b=0$, $X^n \pm Y^n=0$, $\sqrt[n]{X} \pm \sqrt[n]{Y}=0$, for $n \geq 2$, or $X/Y \pm Z/T=0$, with $\text{degree}(XT) \leq 2$ and $\text{degree}(YZ) \leq 2$, or even some case-based reasoning to get rid of the absolute value operators. We note that by writing, for instance, $(\mathbf{k}^*)^2 \text{rad}(N, \text{basic}_{12}) + (\mathbf{k}^*)^2 \text{rad}(N, \text{basic}_{12})$ we really want to restrict N to be the same for both subterms, so that the grammar is not context-free¹. We use $(\mathbf{k}^*)^2 \text{rad}(N, \text{basic}_{12})$ as an abbreviation for $\mathbf{k}^* \text{rad}(N, \text{basic}_{12})$ or $\text{rad}(N, \text{basic}_{12})$, and $*$ means product.

p1 o <i>TypeT</i>	pol($T, [a, b]$)	$aT + b$
p2 o <i>TypeT</i>	pol($T, [a, b, c]$)	$aT^2 + bT + c$
xip(1, N)	expand($N, x, \text{pol}(x, [a, b])$)	$ax^{N+1} + bx^N$
xip(2, N)	expand($N, x, \text{pol}(x, [a, b, c])$)	$ax^{N+2} + bx^{N+1} + cx^N$
pow(N) o <i>TypeT</i>	pow(T, N)	T^N
rad(N) o <i>TypeT</i>	rad(T, N)	$\sqrt[n]{T}$
abs o <i>TypeT</i>	abs(T)	$ T $
p2 o pow(N) o x instead of bisqr(N)	pol(pow(x, N), $[a, b, c]$) instead of bisqr	$ax^{2N} + bx^N + c$

The rightmost column of this table contains the the output expressions that correspond to the basic types, that are given in the first two columns. In our CLP programs the latter are used as the internal representation of these basic expressions, two levels of abstraction being used. It may be checked that

$$\frac{(8x^2 + 14x - 15)(2x + 1)}{(4x^6 - x^5 - 5x^4)(3x^2 - 17x + 10)^2}$$

is of the form

$$\frac{\text{pol}(x, [8, 14, -15]) * \text{pol}(x, [2, 1])}{\text{expand}(4, x, \text{pol}(x, [4, -1, 5])) * \text{pow}(2, \text{pol}(x, [3, -17, 10]))}$$

And, we may also conclude that $-2|-2y+4|+4|3y+3|+2$ belongs to *bsum* (i.e., basic sum expression), since it is given by

$$\text{pol}(\text{abs}(\text{pol}(y, [-2, 4])), [-2, 0]) + \text{pol}(\text{abs}(\text{pol}(y, [3, 3])), [4, 2])$$

¹ In fact, it is known that $\{0^n 10^n 10^n 1 \mid n \geq 1\}$ is not a context-free language.

$function \rightarrow (k^*)^? prodfact \mid (k^*)^? divexpr$
 $prodfact \rightarrow factor \mid prodsexpr$
 $divexpr \rightarrow prodfact/prodfact \mid k/prodfact \mid prodfact/k$
 $\rightarrow pow(N, divexpr) \mid rad(N, divexpr) \mid abs(divexpr)$
 $prodexpr \rightarrow factor*factor \mid factor*prodexpr$
 $\rightarrow pow(N, prodsexpr) \mid rad(N, prodsexpr) \mid abs(prodsexpr)$
 $factor \rightarrow sumexpr \mid vxip \mid basic$
 $sumexpr \rightarrow abs(sumexpr) \mid pow(N, sumexpr) \mid rad(N, sumexpr) \mid bsum$
 $bsum \rightarrow ipol_1(vquot_{12k})$
 $\rightarrow (k^*)^? rad(N, basic_{12}) + (k^*)^? rad(N, basic_{12})$
 $\rightarrow (k^*)^? pow(N, basic_{12}) + (k^*)^? pow(N, basic_{12})$
 $\rightarrow (k^*)^? pow(N, basic_{12}) + (k^*)^? pow(2N, basic_1)$
 $\rightarrow (k^*)^? rad(2N, basic_{12}) + (k^*)^? rad(N, basic_1)$
 $\rightarrow (k^*)^? rad(2, basic_{12}) + (k^*)^? basic_1$
 $\rightarrow (k^*)^? pow(2, basic_1) + (k^*)^? basic_{12}$
 $\rightarrow (k^*)^? basic_{12} + (k^*)^? basic_{12}$
 $\rightarrow (k^*)^? quot_{12k} + (k^*)^? basic_{12}, \text{ subject to Condition}$
 $\rightarrow (k^*)^? quot_{12k} + (k^*)^? quot_{12k}, \text{ subject to Condition}$
 $vquot_{12k} \rightarrow pow(N, vquot_{12k}) \mid rad(N, vquot_{12k}) \mid quot_{12k}$
 $quot_{12k} \rightarrow k/basic_{12} \mid basic_{12}/k \mid basic_{12}/basic_{12} \mid abs(quot_{12k})$
 $basic_{12} \rightarrow basic_1 \mid basic_2$
 $basic_2 \rightarrow fpol_1(abs(basic_2)) \mid ipol_2(x) \mid expand(1, x, ipol_1(x))$
 $\rightarrow basic_1*basic_1 \mid fpol_1(pow(2, basic_1)) \mid pow(2, basic_1)$
 $\rightarrow abs(basic_2)$
 $basic_1 \rightarrow abs(basic_1) \mid fpol_1(abs(basic_1)) \mid fpol_1(x)$
 $basic \rightarrow ipol_2(x) \mid expand(1, x, ipol_1(x)) \mid bisqr \mid fbasic$
 $\rightarrow fpol_1(fbasic) \mid fpol_1(x)$
 $fbasic \rightarrow abs(basic) \mid pow(N, basic) \mid rad(N, basic), N \geq 2$
 $vxip \rightarrow xip \mid k*vxip \mid abs(vxip) \mid pow(N, vxip) \mid rad(N, vxip), N \geq 2$
 $xip \rightarrow expand(N, x, ipol_2(x)) \mid expand(N+1, x, ipol_1(x)), N \geq 1$
 $bisqr \rightarrow ipol_2(pow(N, x)), N \geq 2$
 $fpol_1(T) \rightarrow pol(T, [a, b]), a \neq 0$
 $ipol_2(T) \rightarrow pol(T, [a, b, c]), abc \neq 0$
 $ipol_1(T) \rightarrow pol(T, [a, b]), ab \neq 0$
 $x \rightarrow variable$
 $k \rightarrow constant$

Condition: Being either of the form $(k^*)^? A/B + (k^*)^? C$ with $degree(BC) \leq 2$ or of the form $(k^*)^? A/B + (k^*)^? C/D$ with $degree(AD) \leq 2$ and $degree(BC) \leq 2$.

Fig. 1. A coarse characterization of functions that may appear in exercises

To solve equations that involve *sum expressions* one may need to know how to solve $X^n \pm Y^n = 0$, $\sqrt[n]{X} \pm \sqrt[n]{Y} = 0$, for $n \geq 2$, or $X/Y \pm Z/T = 0$, with $\text{degree}(XT) \leq 2$ and $\text{degree}(YZ) \leq 2$. We notice that, in general we would not be able to solve the first two if instead of 0 we had a non-null constant k .

In the grammar, some categories have names that are indexed by 1, 2 or 12, because they result from the *basic* category when we restrict the degree to be 1, 2, or any of these two. As for *vquot*_{12k} and *quot*_{12k} the idea is that the numerator and denominator have degrees 1, 2, or 0. To avoid defining more grammar rules, the abbreviate notations *fpol*₁(T), *ipol*₂(T) and *ipol*₁(T) were introduced. For instance, *ipol*₂(*pow*(N, x)) rewrites to *pol*(*pow*(N, x), [a, b, c]) by applying the rule (scheme) for *ipol*₂(T).

4 Generating Exercises in a CLP System

CLP languages are quite convenient for naturally encoding requirements of the exercises. By imposing constraints on parameters of the problems generator, we may tune them and thus control the problems difficulty and adequacy for a certain curriculum, stage or user. In order to test these ideas, we have developed a prototype of such a generator in SICStus Prolog [18] using CLP(FD) [3]. In this section, we assume the reader is familiar with CLP languages, and in particular CLP(FD) (for an introduction and some references, see e.g. [12]). The constraint solver shall mainly be used to do consistency checking and to propagate constraints on the exponents and on the number of occurrences of some combinations of particular function types. So, the optimization facilities of the CLP systems shall not be utilized.

In the implementation, a higher level of abstraction for the types of expressions is considered, as shown in the leftmost column of the table given above. This is done by introducing type schemes with constrained finite domain variables to represent sets of expressions of the same form, that is to represent expression templates. They almost mimic the grammar categories, and the main idea is that the composition of functions (herein denoted by \circ) is the main operation to enable the definition of complex functions from the elementary ones. Hence, for example, *power*($_$) \circ *ip*(1) \circ (*p*1 \circ x /*p*1 \circ x) represents *pow*(*ipol*₁(*fpol*₁(x)/*fpol*₁(x)), N), which, by the grammar, is a *sumexpr*. The following expression is a particular instance of this type scheme

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

and has type *power*(7) \circ *ip*(1) \circ (*p*1 \circ x /*p*1 \circ x). Here, *ip*(1) and *p*1 replace *ipol*₁ and *fpol*₁, respectively. In general, the grammar rules are implemented by predicates of the form

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

the main one being *function*(Type, Degree, Rate, CountTypes, CountOps). The parameters *Degree*, *Rate*, *CountTypes*, *CountOps* are used to constrain the resulting scheme *Type*. This allows the impose constraints to control the difficulty

level or form of the generated expressions and to tackle user-defined constraints. For the moment, the overall rate is merely a sum of such rates.

Instances of the expressions of a given **Type** are obtained by a predicate `expression(Type,X,Expr)`. For each type scheme, we may generate several expressions of that type by repeatedly calling `expression/3`. Variations of the same example, in which the coefficients and exponents may change, can be easily found by forcing backtracking, in the CLP framework. Now, instead of saving all the constraint store on the exponents for later usage, the system would rather save either a particular instance of the type scheme or some pre-defined number of expressions that conform the scheme. To test the program and in particular to see how quickly it runs, we defined a predicate `examples/5` that obtains one exercise of each type for some given specifications, through backtracking. It is quite amazing how quickly it obtains a huge number of expressions. E.g., `examples(probs2,2,9,12,z)` writes expressions in the variable `z`, of degree 2 and difficulty level in `[9,12]` to the file `probs2`. The expressions of a given degree shall evaluate to polynomials of that degree when simplified to get rid of `abs` and `pow`, and do not contain quotients and radicals. Different algorithms may be implemented to define `expression/3`, which may even be specialized to the particular problem we have in mind. One possibility was described in Example 1, but we may also simply compute coefficients at random, though within a given range of pedagogical interest. Another possibility would be to use the program to generate several exercises which would later be filtered out, in view of the special application. If only partial consistency is enforced, we have to guarantee that the (random) labeling process eventually stops, when no solution exists. Our program currently implements committed-choice, preventing backtracking to the randomizer when a feasible value is found to the variable. In this way, the program may fail to find a solution even if one exists. This problem is not specific of CLP and other strategies could be devised to overcome it. The type scheme plays a crucial role not only during the generation phase but also to render the implementation of problem solvers easier. We are mainly using CLP(FD) to generate the expressions, which then naturally would have integer coefficients. We have also made some simple experiments in using constraint programming, namely CLP(R), to define and tackle some conditions on the final expressions. However, the results had almost no interest for educational purpose. Further details on the implementation may be found in [20]. The use of a constraint language helps simplify the implementation. As an example, `basic12` is just `Dgr in 1..2`, `basictype(T,Dgr,Rate,Ts,Ops)`, if `basic/5` implements the grammar category *fbasic*. Nevertheless, it is still not easy to implement a constrain-and-generate strategy in order to avoid introducing what can be seen as symmetries in types. Indeed, being `+` a commutative operator, `abs o p1 o x + abs o p2 o x` and `abs o p2 o x + abs o p1 o x` should be the same type. Some symmetries may be filtered out by propagating constraints on the number of operators.

In general, the system needs to support symbolic processing of algebraic expressions to provide exact representations of the solutions. Indeed, CLP(Q) [7]

could be used for finding the solutions, but expressions should have degree 1 and not involve the `abs` construct, so that they would be quite elementary. We have already implemented a prototype program to find the zeros and the domains of some of the expressions, in which computations involving rationals are performed by the CLP(Q) solver. Symbolic manipulation of irrational numbers is supported only for special forms, which for educational purposes are quite sufficient. As for domains, the system may need to exactly solve disequations and disjunctions, so that the CLP(Q) solver cannot be utilized to discard symbolic manipulation of constraints, even when no irrationals are involved.

We would like to obviate the need for students to learn a special syntax just for typing and reading formulas on the computer, unlike in `WebMathematica` [13]. The system already integrates a program to convert the internal representations of the mathematical expressions and solutions to LaTeX. Previously we have also considered using a prototype viewer/editor of MathML documents, written to Tcl/Tk [20]. With the purpose of illustrating the behaviour of the prototype system, we are developing a web interface using the `PiLLoW` package [2]. We are analysing also the integration of the system in a web-based environment. In particular we study the integration in Ganesh [10], although, so far, this distributed learning environment has been mainly used for Computer Science topics, with an emphasis on the automatic grading and correction of students exercises.

5 Conclusions

We proposed a methodology for designing online exercises systems with special focus on applications to Mathematics education. The emphasis is on working backwards from the intended solution of the problem to obtain a sequence of steps leading to that solution. Prototype programs using CLP show that these languages have the right expressiveness to encode control on the system in an elegant way. The main drawback is that we cannot take complete advantage of CLP solvers to reduce the implementation effort. Indeed, we need to handle symbolic representations of some types of irrational numbers. Moreover, we also need symbolic processing of constraints, for example, to be able to find the domains of functions. Since the system must have great control on the solving procedure to be able to explain the solving steps, we think we would not benefit if we used other languages and platforms to realize the system.

References

1. Bryc W., Pelikan S.: Online Exercises System, University of Cincinnati, 1996. (<http://math.uc.edu/onex/demo.html>)
2. Cabeza D., Hermenegildo M., Varma S.: The `PiLLoW`/CIAO Library for INTERNET/WWW Programming using Computational Logic Systems. In *Proc. of the 1st Workshop on Logic Programming Tools for INTERNET Applications, JICSLP'96*, Bonn, 1996. (<http://www.clip.dia.fi.upm.es/miscdocs/pillow>)

3. Carlsson M., Ottosson G., Carlson B.: An Open-Ended Finite Domain Constraint Solver. In *Proceedings of PLILP'97*, LNCS 1292, 191-206, Springer-Verlag, 1997.
4. Cohen A. M., Cuypers H., Sterk H.: *Algebra Interactive*, Springer-Verlag, 1999.
5. Gang X.: WIMS – An Interactive Mathematics Server, Journal of Online Mathematics and its Applications, 1, MAA, 2001. (<http://wims.unice.fr>)
6. Geometer Sketchpad, Key Curriculum Press. (<http://www.keypress.com/sketchpad>)
7. Holzbaur C.: OFAI clp(q,r) Manual, Edition 1.3.3, Austrian Research Institute for Artificial Intelligence, Vienna, TR-95-09, 1995.
8. Kent, P.: Computer-Assisted Problem Posing in Undergraduate Mathematics, Institute of Education, University of London 1996. (<http://metric.ma.ic.ac.uk>)
9. Klai S., Kolokolnikov T., Van der Bergh, N.: Using Maple and the web to grade mathematics tests, International Workshop on Advanced Technologies, Palmerston North, New Zealand, December 2000 (allserv.rug.ac.be/~nvdbergh/aim/docs)
10. Leal J. P. and Moreira N.: Using matching for automatic assessment in computer science learning environments, *Proceedings of Web-based Learning Environments Conference*, 2000. (<http://www.ncc.up.pt/~zp/ganesh>)
11. Maple, Waterloo Maple Corporate. (<http://www.maplesoft.com>)
12. Marriott K., and Stuckey P.: *Programming with Constraints – An Introduction*, The MIT Press, 1998.
13. Mathematica, Wolfram Research Inc. (<http://www.wolfram.com/products/mathematica>)
14. Melis E. et al.: ActiveMath: A Generic and Adaptive Web-Based Learning Environment, *Artificial Intelligence in Education*, 12(4), 2001. (<http://www.mathweb.org/activemath>)
15. Moore L., Smith D. et al.: Connected Curriculum Project CCP, Duke University, 2001. (<http://www.math.duke.edu/education/ccp>)
16. Sangwin C.J.: New opportunities for encouraging higher level mathematical learning by creative use of emerging computer aided assessment. University of Birmingham, UK, May 2002. (www.mat.bham.ac.uk/C.J.Sangwin/)
17. Schrönert M. et al.: *GAP – Groups, Algorithms, and Programming*. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1995.
18. SICStus Prolog User Manual Release 3.8.6, SICS, Sweden, 2001. (<http://www.sics.se/is1/sicstus.html>)
19. WeBWorK, University of Rochester, 2001. (<http://webwork.math.rochester.edu>)
20. Tomás A. P., Vasconcelos P.: Generating Mathematics Exercises by Computer. Internal Report DCC-2001-6, DCC - FC & LIACC, University of Porto, 2001.