Generating Mathematics Exercises by Computer

Ana Paula Tomás, Pedro Vasconcelos

Technical Report Series: DCC-01-06

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

Universidade do Porto

Rua do Campo Alegre, 823 4150 Porto, Portugal

Tel: +351+2+6078830 – Fax: +351+2+6003654

http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html

Generating Mathematics Exercises by Computer

Ana Paula Tomás, Pedro Vasconcelos*

DCC & LIACC, Univ. do Porto

R. do Campo Alegre, 823

4150-180 Porto, Portugal

apt,pbv@ncc.up.pt

July 2001

Abstract

We present ongoing research on the automatic generation of exercises for learning mathematics, in particular high-school algebra. The emphasis is on working backwards from the intended solution of the problem to obtain a sequence of steps leading to that solution. This approach not only avoids generating unsolvable exercises but also allows writing explanatory text for each resolution step. Futhermore, we show how the difficulty level of the exercises can easily be fine-tuned. We discuss our prototype implementation as a CLP(FD) program and a graphical interface in Tcl/Tk. Finally, we propose directions for further research.

1 Motivation

A considerable amount of mathematics teaching resources is available on the Internet. They range from dictionaries and reference material, such as lecture notes [15], to web based environments designed to support virtual classes [2, 4, 17] but that may be also used in the classroom.

Commercial mathematical software, such as Maple [9], Mathematica [11], MATLAB [13], Geometer's Sketchpad [5], just to name a few, is often used as a technology tool to engage in mathematics explorations [7].

The focus of our work is not on problem solving in this broader sense of exploration, but rather on those dull exercises that students have to do for consolidation of concepts and practice of skills and routines.

Some students often work on the same exercise for several times until they have *learned* it by simply memorizing a particular solution rather than by understanding the steps to obtain the solution. This does not help develop students' self-confidence to approach new problems and is certainly one of the reasons for a lack of success in mathematics. Hence, one disadvantage of classical textbooks and some existing online course-wares and exercises systems is that the proposed problems are fixed, or at best generated with slight random variations of some parameters.

This paper reports on our ongoing research that aims at developing an application for automatically generating online examples and exercises for teaching and learning a topic in mathematics. As a major goal, the system should provide explanations that may help students improve their ability to express coherently in the language of mathematics, as suggested by the examples in Section 2.

We investigate an alternative methodology for formulating the exercises whose fundamental idea is to try to mimic what teachers usually do — *to proceed from an intended solution to the statement of the problem.* In this way, we 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

*The work presented here has been partially supported by funds granted to LIACC through the *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia* and *Programa POSI*.

number do. Often, the rules teachers and students employ for solving algebraic exercises can be made into an algorithm.

In the following section, we discuss strengths and weaknesses of our first attempt to implement such a system using a computer algebra environment. The need for a more declarative framework, led us to investigate the use of Constraint Logic Programming (CLP). Aspects of a prototype implementation are presented in Section 4. As we shall see, possible restrictions on the form of generated exercises can be encoded naturally as constraints. This is of great importance for educational purposes since the system must be parametrized to easily cater for different curricula. 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. In Section 5, we briefly address interface issues, concluding with directions for future work.

2 Revisiting Rote Exercises

Some grounding for our current work was provided by our previous experience in developing interactive course-ware using the Maple system. That work involved designing Maple worksheets to present some specific topic in mathematics. Besides some concise notes on the addressed issue, such worksheets typically include pointers to other worksheets 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.

In this section, we look at some of the strengths and weaknesses of that approach, going through examples picked up from such worksheets.

Example 1 *We consider the problem of finding the rational roots of a polynomial with rational coefficients by using the fact that any polynomial* $\sum_{i=0}^{n} a_i x^i$, *with* $a_i \in \mathbb{Z}$ *and* $a_0 \neq 0$, *has root* p/q *only if* p *and* q *are divisors of* a_0 *and* a_n, *respectively. Output from two implemented procedures* `problem_roots(degree)` *and* `example_roots(degree,explain)` *is shown in Fig. 1. To shorten the pictures we have actually adapted such output, by typesetting some of the formulas. Maple V uses a two dimensional character based formatter for pretty printing mathematical formulas, which does not enable us to display them in a single line*[1]. *We shall further discuss issues related to the interface in Section 5.*

As other computer algebra systems, Maple supports polynomial expressions and, consequently, implementing these procedures in Maple is quite simple. Nevertheless, for educational purposes, we need to have some control on the generated polynomial expressions. Rather than just making use of the builtin Maple procedure to generate random polynomials, the computation is driven by the selection of the set of roots, which may be rational and (conjugated) irrational numbers. Factors with no real roots are obtained by adding appropriate constants to quadratic polynomial expressions with real roots. Graphically, this means shifting the parabola upwards in order to remove any intersection with the horizontal axis.

As we shall see, the following example is not so trivial and, moreover, it is at the heart of the solving procedure for several other problems, as for instance defining inverse mappings, finding intervals where some function is monotonous and sketching a function graph.

Example 2 *While learning introductory calculus, students are often asked to characterize the domain, zeros and sign variation of a given real-valued function. In this example, we consider the case of rational functions, which are defined by a quotient of two polinomial expressions, as the ones shown below.*

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

Figs. 3 and 2 contain output from the designed programs, in a compressed form, as we did before for Fig. 1.

```
> example_roots(5,true):
```

COMPUTE THE SET OF RATIONAL ROOTS OF THE POLYNOMIAL

$$x^5 + 2\,x^4 + \frac{3}{2}\,x^3 + \frac{1}{2}\,x^2 + \frac{1}{16}\,x$$

SOLUTION: Since not all the coefficients are integers, we write the given polynomial as the product of $1/16$ by $16\,x^5\ +\ 32\,x^4\ +\ 24\,x^3\ +\ 8\,x^2\ +\ x$. Being a0 = 0, we have p(0)=0, thus 0 is a root. If we write this polynomial as

$$x\,(16\,x^4 + 32\,x^3 + 24\,x^2 + 8\,x + 1)$$

the remaining roots are the ones of $16\,x^4 + 32\,x^3 + 24\,x^2 + 8\,x + 1$, which does not have 0 as a root. By the theorem, the rational roots of $16\,x^4 + 32\,x^3 + 24\,x^2 + 8\,x + 1$ must be either of the form p/q or -p/q, with gcd(p,q)=1, where p belongs to the set of positive divisors of 1, that is to $\{1\}$ and q to the set of positive divisors of 16, that is to $\{1, 2, 4, 8, 16\}$. Therefore, the possible rational roots of such polynomial are

$$1,\,-1,\,\frac{1}{2},\,\frac{-1}{2},\,\frac{1}{4},\,\frac{-1}{4},\,\frac{1}{8},\,\frac{-1}{8},\,\frac{1}{16},\,\frac{-1}{16}$$

but only -1/2 is a root. The set of rationals roots of the given polynomial is $\{0,\,\frac{-1}{2}\}$.

```
> problem_roots(4):
```

COMPUTE THE SET OF RATIONAL ROOTS OF THE POLYNOMIAL

$$2x^4 + 4x^3 - 22x^2 - 24x + 72$$

ANSWER > {1,-3};
p(1) = 32. Thus, 1 is not a root.

Figure 1: Computing the rational roots of a given polynomial expression

CHARACTERIZE THE SIGN OF THE FUNCTION f DEFINED BY

$$f(x) = \frac{x^2(x+3)^3}{x^2+2x+1}$$

SOLUTION: We have to analyse the sign of the expressions in the numerator and denominator.
The expression x^2 has an even exponent, thus $x^2 \geq 0$.

The expression $(x+3)^3$ has an odd exponent, thus it has the same sign as $x+3$. It has negative value for $x < -3$ and positive value for $x > -3$ (notice that the graph of $x \to x+3$ is an ascendant straight line).

The expression x^2+2x+1 has nonnegative value for all real numbers (notice that the graph of $x \to x^2+2x+1$ is a concave up parabola).

In sum, we have

$$\begin{bmatrix} x^2 & + & + & + & + & \texttt{zero(0)} & + \\ (x+3)^3 & - & \texttt{zero(-3)} & + & + & + & + \\ x^2+2x+1 & + & + & \texttt{zero(-1)} & + & + & + \end{bmatrix}$$

and, therefore, we conclude that the sign variation of f is

[- zero(-3) + undef(-1) + zero(0) +]

Figure 2: Finding the sign function of a given rational function.

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

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

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

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

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.

Figure 3: Finding the domain of a rational function.

As suggested by these examples, further research work should be done to improve the linguistic quality of the output explanations. At present, almost no use is made of global context information, which renders some of them too repetitive.

Besides the need for a more flexible input/output interface, two other remarks have played a major role in our decision to adopt a different platform. The first one concerns the algebraic manipulations performed by Maple to simplify the expressions being operated. This feature appears as a great advantage but it is also as a serious drawback for our intended usage of the system. Actually, it may introduce some inconsistencies in the output explanations. In this case, in order to prevent the generated rational functions from being simplified, we had to put stronger conditions on their types, disallowing repetitions of factors and requiring that the involved polynomials have just integer coefficients.

The second point is related to the need for 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.

3 A Case Study – Calculus for Pre-University Levels

In this section we revisit Example 2, looking at more general functions. In order to be able to abstract their possible forms, we have carried out a thorough analysis of Portuguese textbooks in mathematics for grades 10 to 12.

A wide range of the sampling examples is characterized by the grammar shown in Fig. 4. For prototyping, the trigonometric, exponential and logarithmic functions have been left out. The basic types $\texttt{pol}(X,[a,b])$, $\texttt{pol}(X,[a,b,c])$, $\texttt{expand}(N,x,\texttt{pol}(x,[a,b]))$, $\texttt{expand}(N,x,\texttt{pol}(x,[a,b,c]))$, $\texttt{pow}(N,X)$, $\texttt{rad}(N,X)$ and $\texttt{abs}(X)$ stand for $aX+b$, aX^2+bX+c, $ax^{N+1}+bx^N$, $ax^{N+2}+bx^{N+1}+cx^N$, X^N, $\sqrt[N]{X}$ and $|X|$, respectively.

Example 3 *It is not difficult to conclude that*

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

is of the form

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

which is defined as prodfact/prodfact expression by the grammar. Similarly,

$$\texttt{abs}\left(\frac{\texttt{expand}(7,y,\texttt{pol}(y,[-4,-4]))}{\texttt{pol}(\texttt{abs}(\texttt{pol}(y,[-2,4])),[-2,0])+\texttt{pol}(\texttt{abs}(\texttt{pol}(y,[3,3])),[4,2])}\right)$$

represents

$$\left|\frac{-4y^8-4y^7}{-2|-2y+4|+4|3y+3|+2}\right|$$

Basically, with this grammar we try to capture some of the expressions for which the computation of the domain and zeros just involves solving linear or quadratic equations, and possibly some equations of the form $aX^n+b=0$ or $a\sqrt[n]{X}+b=0$, with $n\geq 2$. Notice that by writing, for instance,

$$(\texttt{k*})^?\texttt{rad}(N,basic_{12})+(\texttt{k*})^?\texttt{rad}(N,basic_{12})$$

we really want to restrict N to be the same for both subterms, so that the grammar is not context-free. The abbreviated notation $(\texttt{k*})^?\texttt{rad}(N,basic_{12})$ stands for both $\texttt{k*rad}(N,basic_{12})$ and $\texttt{rad}(N,basic_{12})$.

[1] The latest version – Maple 7 – includes enhanced Web functionality, using WebEQ technology, which could be explored to overcome some of these interface difficulties.

$$
\begin{array}{lcl}
\texttt{x} & \longrightarrow & \mathit{variable} \\
\texttt{k} & \longrightarrow & \mathit{constant} \\
\mathit{ipol}_1(X) & \longrightarrow & \texttt{pol(}X\texttt{,[}a,b\texttt{])},\ \ ab \neq 0 \\
\mathit{ipol}_2(X) & \longrightarrow & \texttt{pol(}X\texttt{,[}a,b,c\texttt{])},\ \ abc \neq 0 \\
\mathit{fpol}_1(X) & \longrightarrow & \texttt{pol(}X\texttt{,[}a,b\texttt{])},\ \ a \neq 0 \\
\mathit{bisqr} & \longrightarrow & \mathit{ipol}_2(\texttt{pow(}N\texttt{,x))},\ \ N \geq 2 \\
\mathit{xnp} & \longrightarrow & \texttt{expand(}N\texttt{,x,}\mathit{ipol}_2(\texttt{x}))\ \mid\ \texttt{expand(}N+1\texttt{,x,}\mathit{ipol}_1(\texttt{x})),\ \ N \geq 1 \\
\mathit{vxnp} & \longrightarrow & \mathit{xnp}\ \mid\ \texttt{k*}\mathit{vxnp}\ \mid\ \texttt{abs(}\mathit{vxnp}\texttt{)}\ \mid\ \texttt{pow(}N\texttt{,}\mathit{vxnp}\texttt{)}\ \mid\ \texttt{rad(}N\texttt{,}\mathit{vxnp}\texttt{)}, N \geq 2 \\
\mathit{fbasic} & \longrightarrow & \texttt{abs(}\mathit{basic}\texttt{)}\ \mid\ \texttt{pow(}N\texttt{,}\mathit{basic}\texttt{)}\ \mid\ \texttt{rad(}N\texttt{,}\mathit{basic}\texttt{)}, N \geq 2 \\
\mathit{basic} & \longrightarrow & \mathit{ipol}_2(\texttt{x})\ \mid\ \texttt{expand(1,x,}\mathit{ipol}_1(\texttt{x}))\ \mid\ \mathit{bisqr}\ \mid\ \mathit{fbasic} \\
 & \longrightarrow & \mathit{fpol}_1(\mathit{fbasic})\ \mid\ \mathit{fpol}_1(\texttt{x}) \\
\mathit{basic}_1 & \longrightarrow & \texttt{abs(}\mathit{basic}_1\texttt{)}\ \mid\ \mathit{fpol}_1(\texttt{abs(}\mathit{basic}_1\texttt{))}\ \mid\ \mathit{fpol}_1(\texttt{x}) \\
\mathit{basic}_2 & \longrightarrow & \mathit{fpol}_1(\texttt{abs(}\mathit{basic}_2\texttt{))}\ \mid\ \mathit{ipol}_2(\texttt{x})\ \mid\ \texttt{expand(1,x,}\mathit{ipol}_1(\texttt{x})) \\
 & \longrightarrow & \mathit{basic}_1\texttt{*}\mathit{basic}_1\ \mid\ \mathit{fpol}_1(\texttt{pow(2,}\mathit{basic}_1\texttt{))}\ \mid\ \texttt{pow(2,}\mathit{basic}_1\texttt{)} \\
 & \longrightarrow & \texttt{abs(}\mathit{basic}_2\texttt{)} \\
\mathit{basic}_{12} & \longrightarrow & \mathit{basic}_1\ \mid\ \mathit{basic}_2 \\
\mathit{quot}_{12k} & \longrightarrow & \texttt{k/}\mathit{basic}_{12}\ \mid\ \mathit{basic}_{12}\texttt{/k}\ \mid\ \mathit{basic}_{12}\texttt{/}\mathit{basic}_{12}\ \mid\ \texttt{abs(}\mathit{quot}_{12k}\texttt{)} \\
\mathit{vquot}_{12k} & \longrightarrow & \texttt{pow(}N,\mathit{vquot}_{12k}\texttt{)}\ \mid\ \texttt{rad(}N,\mathit{vquot}_{12k}\texttt{)}\ \mid\ \mathit{quot}_{12k} \\
\mathit{bsum} & \longrightarrow & \mathit{ipol}_1(\mathit{vquot}_{12k}) \\
 & \longrightarrow & (\texttt{k*})^?\texttt{rad(}N,\mathit{basic}_{12}\texttt{)} + (\texttt{k*})^?\texttt{rad(}N,\mathit{basic}_{12}\texttt{)} \\
 & \longrightarrow & (\texttt{k*})^?\texttt{pow(}N,\mathit{basic}_{12}\texttt{)} + (\texttt{k*})^?\texttt{pow(}N,\mathit{basic}_{12}\texttt{)} \\
 & \longrightarrow & (\texttt{k*})^?\texttt{pow(}N,\mathit{basic}_{12}\texttt{)} + (\texttt{k*})^?\texttt{pow(}2N,\mathit{basic}_1\texttt{)} \\
 & \longrightarrow & (\texttt{k*})^?\texttt{rad(}2N,\mathit{basic}_{12}\texttt{)} + (\texttt{k*})^?\texttt{rad(}N,\mathit{basic}_1\texttt{)} \\
 & \longrightarrow & (\texttt{k*})^?\texttt{rad(}2,\mathit{basic}_{12}\texttt{)} + (\texttt{k*})^?\mathit{basic}_1 \\
 & \longrightarrow & (\texttt{k*})^?\texttt{pow(}2,\mathit{basic}_1\texttt{)} + (\texttt{k*})^?\mathit{basic}_{12} \\
 & \longrightarrow & (\texttt{k*})^?\mathit{basic}_{12} + (\texttt{k*})^?\mathit{basic}_{12} \\
 & \longrightarrow & (\texttt{k*})^?\mathit{quot}_{12k} + (\texttt{k*})^?\mathit{basic}_{12},\ \ \text{subject to } \textbf{Condition} \\
 & \longrightarrow & (\texttt{k*})^?\mathit{quot}_{12k} + (\texttt{k*})^?\mathit{quot}_{12k},\ \ \text{subject to } \textbf{Condition} \\
\mathit{sumexpr} & \longrightarrow & \texttt{abs(}\mathit{sumexpr}\texttt{)}\ \mid\ \texttt{pow(}N,\mathit{sumexpr}\texttt{)}\ \mid\ \texttt{rad(}N,\mathit{sumexpr}\texttt{)}\ \mid\ \mathit{bsum} \\
\mathit{factor} & \longrightarrow & \mathit{sumexpr}\ \mid\ \mathit{vxnp}\ \mid\ \mathit{basic} \\
\mathit{prodexpr} & \longrightarrow & \mathit{factor}\texttt{*}\mathit{factor}\ \mid\ \mathit{factor}\texttt{*}\mathit{prodexpr} \\
 & \longrightarrow & \texttt{pow(}N,\mathit{prodsexpr}\texttt{)}\ \mid\ \texttt{rad(}N,\mathit{prodsexpr}\texttt{)}\ \mid\ \texttt{abs(}\mathit{prodsexpr}\texttt{)} \\
\mathit{prodfact} & \longrightarrow & \mathit{factor}\ \mid\ \mathit{prodsexpr} \\
\mathit{divexpr} & \longrightarrow & \mathit{prodfact}\texttt{/}\mathit{prodfact}\ \mid\ \texttt{k/}\mathit{prodfact}\ \mid\ \mathit{prodfact}\texttt{/k} \\
 & \longrightarrow & \texttt{pow(}N,\mathit{divexpr}\texttt{)}\ \mid\ \texttt{rad(}N,\mathit{divexpr}\texttt{)}\ \mid\ \texttt{abs(}\mathit{divexpr}\texttt{)} \\
\mathit{function} & \longrightarrow & (\texttt{k*})^?\mathit{prodfact}\ \mid\ (\texttt{k*})^?\mathit{divexpr}
\end{array}
$$

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

Figure 4: A coarse characterization of functions that may appear in exercises

4 Generating Exercises in a CLP System

Constraint (Logic) Programming 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 and stage.

In order to test these ideas, we have developed a prototype of such a generator in SICStus Prolog [14] using CLP(FD) [3]. In this section, we assume the reader is familiar with Constraint Logic Programming languages, and in particular CLP(FD) (for an introduction and some references, see e.g. [10]).

We introduce type schemes with constrained finite domain variables to represent sets of expressions of the same form. Since the types mimic the grammar categories, we shall not give a precise description of such language. Instead we present some examples. Hence,

```
power(_)o ip(1)o(p1 o x/p1 o x)
```

denotes $\texttt{pow}(N, ipol_1(fpol_1(\texttt{x})/fpol_1(\texttt{x})))$, which, by the grammar, is a *sumexpr*.

$$\begin{array}{rcl} sumexpr & \Rightarrow & \texttt{pow}(N, sumexpr) \ \Rightarrow \texttt{pow}(N, bsum) \\ & \Rightarrow & \texttt{pow}(N, ipol_1(vquot_{12k})) \ \Rightarrow \texttt{pow}(N, ipol_1(quot_{12k})) \\ & \Rightarrow^\star & \texttt{pow}(N, ipol_1(fpol_1(\texttt{x})/fpol_1(\texttt{x}))) \end{array}$$

Here, `ip(1)` and `p1` replace $ipol_1$ and $fpol_1$, respectively. The operator `o` stands for function composition and may be declared as follows.

```
:- op(14, xfy, o ).
```

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)`. As we describe below in more detail, the parameters `Degree`, `Rate`, `CountTypes`, `CountOps` are used to constrain the resulting scheme `Type`. An instance of the expressions of a given `Type` is obtained by calling `expression(Type,X,Expr)`.

Before moving on to present some excerpts of the implemented program, we give some more examples.

Example 4 *For a better understanding of the meaning of schemes as*

```
xip(_,_)/(k*(power(2)o p1 o x) + abs o p2 o x)

rad(_)o(abs o p1 o power(2)o p1 o x+power(2)o p1 o x)
```

we give also an instance of the expressions they represent and display it in usual mathematical form. Notice that, while generating such instances, random values are given to the variables in the type scheme, so that they become instantiated.

```
xip(1,9)/(k*(power(2)o p1 o x) + abs o p2 o x)
expand(9,x,pol(x,[1,3]))/(-4*pow(2,pol(x,[4,-3]))+abs(pol(x,[-5,-2,-5])))
```

$$\frac{x^{10} + 3x^9}{-4(4x-3)^2 + |-5x^2 - 2x - 5|}$$

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

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

```
p1 o power(6)o abs o p1 o x*power(7)o rad(8)o p1 o x
pol(pow(6,abs(pol(y,[-2,-1]))),[3,-3])*pow(7,rad(8,pol(y,[-1,-5])))
```

$$(3|-2y-1|^6-3)(\sqrt[8]{-y-5})^7$$

```
p1 o abs o rad(6)o p1 o x*p2 o power(2)o x
pol(abs(rad(6,pol(x,[1,-5]))),[-3,-2])*pol(pow(2,x),[1,-5,2])
```

$$(-3\,|\,\sqrt[6]{x-5}\,|-2)\,(x^4-5x^2+2)$$

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

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

The basic constructs for the *basic* and *vxnp* grammar categories are distinguished as polynomial or non-polynomial functions, as shown below.

```
npftype(abs).
npftype(rad(_)).
npftype(power(_)).

pftype(p1).
pftype(p2).

% Which functions may T be composed with?
ctype_(T,x) :- pftype(T).
ctype_(p1,T o _) :- npftype(T).
ctype_(T,xip(_,_)) :- npftype(T).
ctype_(T,Tc o _) :- npftype(T), (pftype(Tc); (npftype(Tc), T \= Tc)).
```

Then, the predicate defining the grammar category *basic* is written as follows.

```
basictype(xip(1,1),2,Rate,Ts,0) :-
   user_rate(p2,Rate),
   sum(Ts,#=,1), increment(p2,Ts).
basictype(p2 o power(N) o x,Dgr,Rate,Ts,2) :-
   user_rate(bisqr(N),Rate),
   sum(Ts,#=,1), increment(bisqr(N),Ts),
   degree(bisqr(N),Dgr).
basictype(T o Tc,Dgr,Rate,Ts,Ops) :- (pftype(T); npftype(T)),
   degree(T,DgrT), DgrC #>= 1, Dgr #= DgrT*DgrC,
   rate_restr(T,Rate,[RateC]),
   types_restr(T,Ts,[TsC]),
   ctype_(T,Tc),
   ops_restr(Ops,1,[OpsC]),
   ((Tc = x, DgrC #= 1, OpsC #= 0, RateC #= 0, sum(TsC,#=,0));
     basictype(Tc,DgrC,RateC,TsC,OpsC)).
```

`Rate` is a parameter that allows us to have some control on the application of each of the clauses defining a predicate. It must be instantiated or bounded when calling `function/5`. User-defined rates are assigned by `user_rate/2` to the primitive functions and to particular sub-expressions (as for example, sums of radicals, quotients and products). The overall rate is a sum of such rates.

`Ts` is a list of finite domain variables, each one giving the number of occurrences of a given type. These types include `p1`, `p2`, `abs`, `rad(_)`, `pow(_)`, `xip(_,_)`, `bisqr(_)`, but also more general ones as, for instance, `prodstype`, `divstype` and `sum`. The latter is related to the expressions identified by *sumtype* in the grammar. The idea is that the user may define constraints on the values of the counters in `Ts`. These constraints may involve a single variable (e.g., to specify its domain) or any subset of them. Calls to `constrs/2` result in imposing the user-defined constraints on `Ts` for the category identified by `urestr_`*name*.

The predicates `rate_restr/3` and `types_restr/3` propagate the constraints for rates and type counters, respectively. The same applies to `ops_restr/3`, which is used to count the number of operations (i.e., compositions, sums, products and quotients) in the expressions.

```
increment(T,Ts) :- type_index(T,I), element(I,Ts,Tsi), Tsi #= 1.

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

types_restr(T,Ts,Lsts) :- length(Ts,NTs), lists_vars(Lsts,NTs),
   (Lsts = [L] -> identify(T,L,Ts);
              (add_lists(Lsts,Sums), link_restrs(T,Sums,Ts))).
```

Here, `lists_vars/2` creates lists of domain variables of equal length, which is given by `NTs`. The linking constraints are then imposed by `identify/3` or `link_restrs/3`. The former unifies the variables in `Ts` with these in `L`, except the one related to type `T`. The latter defines equality constraints between the terms (sums) in `Sums` and the corresponding variables in `Ts`, that is, constraints of the form `Sumk #= Tsk`. Again, with the exception of the one related to type `T`, which is rather something like `Sumi+IncrTsi #= Tsi`, where `IncrTsi` is either 1 or any other increment clearly given by `T`.

The definition of the grammar category *vxnp* is adapted from that of *basic* in the following way.

```
vxnptype(xip(I,N),Dgr,Rate,Ts,0) :- user_rate(xip(I,N),Rate),
   sum(Ts,#=,1), increment(xip(I,N),Ts),
   degree(xip(I,N),Dgr),  Dgr #>= 3.
vxnptype(T o Tc,Dgr,Rate,Ts,Ops) :-  npftype(T),
   degree(T,DgrT),  DgrC #>= 1, Dgr #= DgrT*DgrC,
   rate_restr(T,Rate,[RateC]), types_restr(T,Ts,[TsC]),
   ops_restr(Ops,1,[OpsC]),
   ctype_(T,Tc), vxnptype(Tc,DgrC,RateC,TsC,OpsC).
```

In broad terms, the degree of an expression is the degree of the polynomial it reduces to, if we do not make simplifications that may cancel some terms. In this way, we assume that it is undefined, say ∞, for radicals and quotients. As regards sums, we naturally have $degree(X+Y) = \max(degree(X), degree(Y))$. The `degree/2` predicate imposes constraints on the exponents occuring in the generated expressions, as shown below.

```
degree(k,0).
degree(p1,1).
degree(p2,2).
degree(abs,1).
degree(power(N),N) :- exponent(N,2).
```

```
degree(rad(N),Dgr) :- undefined(Dgr), exponent(N,2).
degree(bisqr(N),Dgr) :- exponent(N,2), Dgr #= 2*N.
degree(xip(I,N),Dgr) :- I in 1..2, Dgr #= N+I, exponent(N,1).

exponent(N,Nmin) :- maxexponent(NMax), N in Nmin..NMax.
```

The use of a constraint language helps simplify the implementation. As an example, $basic_{12}$ is just `Dgr in 1..2, basictype(T,Dgr,Rate,Ts,Ops)`. 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.

In fact, being `+` a commutative operator, the following schemes could be viewed as the same one. The same may be said about `*` (the product).

```
abs o p1 o x + abs o p2 o x
abs o p2 o x + abs o p1 o x
```

To filter out symmetries we can, for example, introduce an interpretation of the relevant types in naturals, and impose some lexicographic ordering.

By counting the number of operators in the expression, we already get a partial solution, that is computationally less expensive. This is illustrated with `prodstype/6`, which implements the grammar category $prodfact$. The variable `MaxOps` sets an upper bound on the number of operators in each $factor$.

```
factorstype(T,Dgr,Rate,Ts,Ops) :-
  ((constrs(Ts,urestr_basic), basictype(T,Dgr,Rate,Ts,Ops));
   (constrs(Ts,urestr_vxnp), vxnptype(T,Dgr,Rate,Ts,Ops));
   (constrs(Ts,urestr_vsum), v_sumtype(T,Dgr,Rate,Ts,Ops))).

prodstype(T,Dgr,Rate,Ts,MaxOps,Ops) :-  Ops #=< MaxOps,
   constrs(Ts,urestr_factor),factorstype(T,Dgr,Rate,Ts,Ops).
prodstype(Tb*T,Dgr,Rate,Ts,MaxOps,Ops) :-
   DgrB #>= 1, DgrT #>= 1, Dgr #= DgrT+DgrB,
   rate_restr(prodstype,Rate,[RateB,RateT]),
   types_restr(prodstype,Ts,[TsB,TsT]),
   ops_restr(Ops,1,[OpsB,OpsT]), OpsB #=< MaxOps,
   constrs(TsB,urestr_factor),
   factorstype(Tb,DgrB,RateB,TsB,OpsB),
   prodstype(T,DgrT,RateT,TsT,OpsB,OpsT).
```

5 Pretty Printing of Mathematical Formulas

Any mathematical tutoring system requires some way of outputing text and mathematical formulas to the user. Although textual representations for mathematical expressions are quite common, for example, in programming languages, they are not suitable for a learning environment. The presentation and editing should be as close to the classical pencil-and-paper notation as possible. This is specially true for undergraduate students who are in fact learning to read and write the classical notation; they should not have to learn a special syntax just for typing and reading formulas on the computer.

Computer algebra systems like *Maple* and *Mathematica* do some basic two-dimensional formatting of the output presented to the user, but they have some limitations: the automatic re-ordering and simplification routines might modify the expression presented; they also have limited text formating possibilities (for example: mixing formulas and text in a single line, or formatting tables).

5.1 MathML representation

To overcome these dificulties, we chose a more flexible approach: generating a textual representation of the exercise using a subset of MathML.

MathML is an instance of the XML markup languages for encoding documents containing mathematical formulas [12]. MathML is designed not only to present maths on-screen (i.e.: on web documents) but also to support high-quality printing, rendering for non-graphical readers (for example: audio readers) and to serve as input/output language between math-aware software (for example: computer algebra systems).

Unfortunately, there are currently few web browsers that support MathML directly (one example: Amaya [1]), and none of the more common browsers support it "off-the-shelf". The situation is likely to improve in the future, either by the use of plug-in applications or updated browsers.

5.2 MathML viewer

Tag	Description
<math>	begin math expression
<mn>	numeric constant
<mi>	identifier
<mo>	operator
<mrow>	horizontal row
<msup>	superscripting
<mfrac>	vertical fraction
<msqrt>, <mroot>	radicals
<mfenced>	parentesis and delimiters

Figure 5: MathML markup tags supported by our viewer.

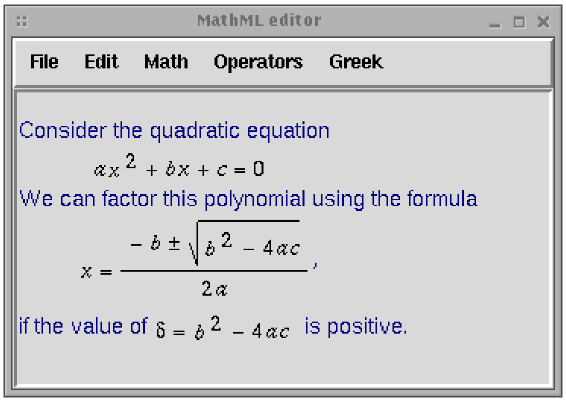

Figure 6: Example rendering from our MathML viewer.

To visualize the MathML document we use our own prototype viewer/editor, written using the Tcl/Tk toolkit [16]. This prototype viewer supports just a small subset of the *presentation* MathML, but this subset is all our MathML documents use (see figure 5). The quality of the layout is comparable to that obtained using *Maple*, with the advantage that text can be freely mixed with math formulas (see figure 6).

By choosing a subset language from MathML we gained a good framework for further increasing the expressiveness of our presentation language should the need arise in the future. Furthermore, when MathML is directly supported by web browsers, we can easily integrate our documents in web pages.

One further advantage is that our viewer is also a *graphical editor* where math expressions can be entered and modified in two-dimensional layout (unlike in Maple, for example, where expressions are always edited in textual form). In the future, we intend to allow the student to interact with the tutoring system (for example, to input trial solutions for the exercises) and this will avoid the need to learn a special syntax.

6 Conclusions and Further Work

We proposed a methodology for designing online exercises systems with special focus on applications to Mathematics education. Ongoing study aims at investigating whether Constraint Logic Programming frameworks are adequate to implement such a system. Preliminary, but encouraging, results were reported in this paper, indicating that these languages have the right expressiveness to encode control on the system in an elegant way. In order to reduce the implementation effort, we have also tried to develop a problem solver using CLP(Q) [6]. But, since we would like to handle representations of irrational numbers, the system shall include symbolic processing of algebraic expressions.

Future research will address how to integrate such a system in a web-based environment. The goal is to develop web pages that make exercises and solutions available to students (and teachers). In particular, we shall investigate a possible integration in the Ganesh [8] environment.

A further line of research involves studing the applicability to other domains, as for example, Linear Algebra and Introductory Calculus at undergraduate level.

References

[1] *Amaya, W3C's editor/browser*, W3 Consortium.
`http://www.w3c.org/Amaya`

[2] Bryc W., Pelikan S.: Online Exercises System, University of Cincinnati, 1996.
`http://math.uc.edu/onex/demo.html`

[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] Gang X.: WIMS – An Interactive Mathematics Server, Journal of Online Mathematics and its Applications, 1, MAA, 2001.
`http://wims.unice.fr`

[5] Geometer Sketchpad, Key Curriculum Press.
`http://www.keypress.com/sketchpad`

[6] Holzbaur C.: OFAI clp(q,r) Manual, Edition 1.3.3, Austrian Research Institute for Artificial Intelligence, Vienna, TR-95-09, 1995.

[7] Kent, P.: Computer-Assisted Problem Posing in Undergraduate Mathematics, Institute of Education, University of London 1996.
`http://metric.ma.ic.ac.uk/`

[8] 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`

[9] Maple, Waterloo Maple Corporate.
`http://www.maplesoft.com`

[10] Marriott K., and Stuckey P.: *Programming with Constraints – An Introduction*, The MIT Press, 1998.

[11] Mathematica, Wolfram Research Inc.
`http://www.wolfram.com/products/mathematica`.

[12] *Mathematical Markup Language (MathML) Version 2.0*, W3C Recomendation, February 2001.
`http://www.w3c.org/TR/2001/REC-MathML2-2001021`

[13] MATHLAB, The MATH WORKS Inc.
`http://www.mathworks.com`

[14] SICStus Prolog User Manual Release 3.8.6, SICS, Sweden, 2001.
`http://www.sics.se/isl/sicstus.html`

[15] S.O.S. MATHematics, Math Medics, LLC, USA, 2001.
`http://www.sosmath.com`

[16] The Tcl developer Xchange, Scriptics.
`http://www.scriptics.com`

[17] WeBWorK, University of Rochester.
`http://webwork.math.rochester.edu`