Designing a Solver for Arithmetic Constraints to Support Education in Mathematics*

Ana Paula Tomás, Nelma Moreira, and Nuno Pereira

DCC-FC & LIACC
University of Porto, Portugal
\{apt,nam,nfp\}@dcc.fc.up.pt

Abstract. We present a conditional rewrite system for arithmetic and membership univariate constraints over real numbers, designed for computer assisted learning (CAL) in elementary math. Two fundamental principles guided the design of the proposed rewrite rules: *cognitive fidelity* (emulating steps students should take) and *correctness*, aiming that step-by-step solutions to problems look like ones carried out by students. In order to gain more flexibility to modify rules, add new ones and customize solvers, the rules are written in a specification language and then compiled to Prolog. The rewrite system is complete for a relevant subset of problems found in high-school math textbooks.

1 Introduction

To understand what people do when they do mathematics and write programs emulating that process is a continuous research topic in Artificial Intelligence, Automated Reasoning, and Symbolic Computation [3, 8]. Computer Mathematics is by now an established, although developing, subject. The challenge is to make the systems, including Computer Algebra systems and Proof Assistants, more (mathematician-)friendly [1]. Symbolic computation systems, like the commercial packages Maple and Mathematica, are widely used, though they can produce unexpected or wrong answers [1, 2, 5]. Nevertheless, in order to reduce the effort of writing solvers, some web-based learning environments and e-learning authoring tools support (unsafe) interaction with them [7, 10]. Those packages were not developed specifically for education, which makes it difficult to get them generate step-by-step solutions that are *cognitive faithful*, i.e. that emulate the steps a student should take. In [2] a discussion about design criteria of software for mathematics education is given. AGILMAT – *Automatic Generation of Interactive Drills for Mathematics Learning* (`www.ncc.up.pt/AGILMAT/`) – aims at the design and implementation of a system to automatically create and solve math exercises, continuing research work reported in [9]. There, we introduced a prototype, called DEMOMATH, that also

* Partially funded by FCT and POSI, co-financed by EC fund FEDER, under project AGILMAT (contract POSI/CHS/48565/2002).

Please use the following format when citing this chapter:
Tomas, Ana Paula, Moreira, Nelma, Pereira, Nuno, 2006, in IFIP International Federation for Information Processing, Volume 204, Artificial Intelligence Applications and Innovations, eds. Maglogiannis, I., Karpouzis, K., Bramer, M., (Boston: Springer), pp. 433–441

yields one-line solutions for some exercises. Its solver is fairly ad-hoc, and cannot be easily adapted to present step-by-step solutions with pedagogic interest, which motivated our current work. We propose a conditional rewrite system for arithmetic and membership univariate constraints over real numbers. To gain flexibility, the rules are written in a specification language and then compiled to Prolog. In the next section we recall basic notions of real-valued functions and give examples of problems we want to automate. In Section 3 we introduce our representation for problems and constraints and show how to convert membership to arithmetic constraints, and reciprocally. Section 4 is devoted to the presentation of the proposed rewriting system, which was designed to be complete for the problems that can be solved by analyzing the sign variation of functions created by DEMOMATH.

2 Some Mathematical Background and Examples

We start with some notions about real-valued functions. $\mathbb{R}$ stands for the set of the real numbers, a, b, c, k for real constants, f, g, h for generic real-valued functions over $\mathbb{R}$, and x, y, z for real valued variables. As usual, $\mathcal{D}_f$ is the domain of the function f, and its image (a.k.a., range) is $f(\mathcal{D}_f) = \{f(x) : x \in \mathcal{D}_f\}$. We represent the restriction of f to $D \subseteq \mathcal{D}_f$ by $f|_D$ and the inverse function by f^{-1}, if it exists. If f is strictly monotonic over D, then $f|_D$ is invertible. The following table shows the basic functions studied in math at high school and some of their properties, if we exclude the trigonometric functions and generic polynomial functions $pol_{a_n,\ldots,a_0} : x \mapsto \sum_{i=0}^{n} a_i x^i$.

f	$\mathcal{D}_f$	$f(\mathcal{D}_f)$	Behavior in $\mathcal{D}_f$	Inverse function
$id : x \mapsto x$	$\mathbb{R}$	$\mathbb{R}$	strictly increases, odd	$id^{-1} = id$
$c_k : x \mapsto k$	$\mathbb{R}$	$\{k\}$	constant, even	—
$p_k : x \mapsto kx,\ k \neq 0$	$\mathbb{R}$	$\mathbb{R}$	strictly increases if $k > 0$ strictly decreases if $k < 0$ odd	$p_k^{-1} : x \mapsto \frac{1}{k}x$
$pol_{a,b} : x \mapsto ax + b$	$\mathbb{R}$	$\mathbb{R}$	strictly increases if $a > 0$ strictly decreases if $a < 0$ odd if $b = 0$	$pol_{a,b}^{-1} : x \mapsto \frac{1}{a}x - \frac{b}{a}$
$pow_{2n+1} : x \mapsto x^{2n+1}$	$\mathbb{R}$	$\mathbb{R}$	strictly increases, odd	$pow_{2n+1}^{-1} = rad_{2n+1}$
$pow_{2n} : x \mapsto x^{2n}$	$\mathbb{R}$	$\mathbb{R}_0^+$	symmetric w.r.t. $x = 0$ $pow_{2n}\|_{\mathbb{R}_0^+}$ strictly increases even	$(pow_{2n}\|_{\mathbb{R}_0^+})^{-1} = rad_{2n}$
$rad_{2n+1} : x \mapsto \sqrt[2n+1]{x}$	$\mathbb{R}$	$\mathbb{R}$	strictly increases, odd	$rad_{2n+1}^{-1} = pow_{2n+1}$
$rad_{2n} : x \mapsto \sqrt[2n]{x}$	$\mathbb{R}_0^+$	$\mathbb{R}_0^+$	strictly increases	$rad_{2n}^{-1} = pow_{2n}\|_{\mathbb{R}_0^+}$
$abs : x \mapsto \|x\|$	$\mathbb{R}$	$\mathbb{R}_0^+$	symmetric w.r.t. $x = 0$ $abs\|_{\mathbb{R}_0^+}$ strictly increases even	$(abs\|_{\mathbb{R}_0^+})^{-1} = id\|_{\mathbb{R}_0^+}$
$exp_a : x \mapsto a^x$	$\mathbb{R}$	$\mathbb{R}^+$	strictly increases if $a > 1$ strictly decreases if $0 < a < 1$	$exp_a^{-1} = log_a$
$log_a : x \mapsto \log_a x$	$\mathbb{R}^+$	$\mathbb{R}$	strictly increases if $a > 1$ strictly decreases if $0 < a < 1$	$log_a^{-1} = exp_a$

Composition, sum, difference, product and quotient of functions are represented by $\circ$, $+$, $-$, $\times$ and $/$. We have $\mathcal{D}_{f\circ g} = \mathcal{D}_g \cap \{x : g(x) \in \mathcal{D}_f\}$, $\mathcal{D}_{f\odot g} = \mathcal{D}_f \cap \mathcal{D}_g$ for $\odot \in \{+, -, \times\}$ and $\mathcal{D}_{f/g} = \mathcal{D}_f \cap \mathcal{D}_g \setminus \{x : g(x) = 0\}$. A piecewise function f is of form $(f_i, D_i)_{i=1}^n$, with $n \geq 2$, being $f(x)$ given by $f_i(x)$ if $x \in D_i$.

Drills and practice We now give some examples of exercises we are interested in automating. The first ones are from a high school math textbook (grade 11). To create the two last ones and get their solution we have used AGILMAT (available at `www.ncc.up.pt:8080/Agilmat/`).

- Find the domain of $f(x) = \frac{\sqrt[3]{1-x^2}}{4-x+\sqrt{x+2}}$.
- Express $g(x) = |x-1| + |x+1| + x$ without using the absolute value function. Solve $g(x) < |x+3|$.
- Study the sign variation of $-\sqrt{\frac{2|2x^2+2x|}{-3x-1}} - 1$ for $x \in \mathbb{R}$. (Solution: Negative in $]-\infty, -\frac{1}{3}] \cup \{0\}$ (the domain of the expression)).
- Solve $\left(2\left|-x^2-2x+1\right|\right)^2 \left(2x^4 - x^2 - 3\right) \neq 0$ for $x \in \mathbb{R}$. (Solution: $]-\infty, \infty[\setminus \{-1-\sqrt{2}, -\frac{1}{2}\sqrt{6}, -1+\sqrt{2}, \frac{1}{2}\sqrt{6}\}$)

3 Constraints and Problems

We would like to solve problems that may involve arithmetic and membership constraints, because both types coexist in some math problems. We define *atomic* and *complex* constraints as follows.

The *atomic arithmetic constraints* are either of the form $f(x) \trianglelefteq g(x)$ and $f(x) \trianglelefteq k$ with $\trianglelefteq \in \{=, \neq, >, <, \leq, \geq\}$, f and g are real valued functions on reals and k is a ground arithmetic-term. The *atomic membership constraints* are of form $f(x) \trianglelefteq S$ with $\trianglelefteq \in \{\in, \notin\}$ and S is a ground set-term. The conjunction and disjunction of a finite number of constraints in the variable x is *a (complex) constraint* $C(x)$.

We often write C instead of $C(x)$, since we will address only problems that involve a unique variable. We use $\trianglelefteq^{-1}$ to denote the inverse of the binary relation $\trianglelefteq$, for $\trianglelefteq \in \{=, \neq, \leq, \geq, >, <\}$. We inductively define *the domain of constraint* C (denoted by $\mathcal{D}_C$) by $\mathcal{D}_{f(x) \trianglelefteq g(x)} = \mathcal{D}_f \cap \mathcal{D}_g$, $\mathcal{D}_{f(x) \trianglelefteq k} = \mathcal{D}_{f(x) \trianglelefteq S} = \mathcal{D}_f$, $\mathcal{D}_{\wedge_{i=1}^n C_i} = \cap_{i=1}^n \mathcal{D}_{C_i}$ and $\mathcal{D}_{\vee_{i=1}^n C_i} = \cup_{i=1}^n \mathcal{D}_{C_i}$.

The *problem* P of finding all $x \in D$ that satisfy the constraint C is denoted by a tuple $\langle C, x, D\rangle$. A problem is in *solved form* iff it is $\langle id(x) \in D, x, D\rangle$ and D is then called *the solution set* of the problem. (For short, we shall write $\langle x \in D, x, D\rangle$ instead.)

3.1 Membership versus Arithmetic Constraints

It is important to be able to convert membership to arithmetic constraints and reciprocally. For that we define two representations for sets. A set is in a *standard form* if it is either $\emptyset$ or the union of a finite sequence $S_1, \ldots, S_n$ of non-empty intervals and/or finite sets of $\mathbb{R}$, that are pairwise disjoint and such that $\sup(S_i) \leq \inf(S_{i+1})$ for all $1 \leq i < n$ and if $\sup(S_i) = \inf(S_{i+1})$ then $\sup(S_i) \notin S_i$ and $\inf(S_{i+1}) \notin S_{i+1}$. The infimum and supremum of each set may be $-\infty$ and $+\infty$. A *constraining set* is a subset of $\mathbb{R}$ that may be written in standard form.

Although the *constraining sets* do not fully represent all subsets of $\mathbb{R}$, they cater for the most frequent types of sets that occur in math drills, if trigonometry is excluded. This standard form is like a picture of the set in the real axis.

Example 1. The set $([-3, -1[\cup\{2, 17\} \cup [8, 11[\cup]11, 14[) \setminus \{10\}$ is a constraining set and it standard form is $[-3, -1[\cup\{2\} \cup [8, 10[\cup]10, 11[\cup]11, 14[\cup\{17\}$.

We now introduce *the reduced normal form* which gives a more compact *arithmetic* representation of each constraining set, being thus relevant for CAL. The reduced normal form is unique. A constraining set is *in reduced normal form* (**rnf**) iff it is given in one of the following forms: $\mathbb{R}$, $\emptyset$, a finite non-empty set, $\cup_{i=1}^n S_i$, $\mathbb{R} \setminus S_{n+1}$, $(\cup_{i=1}^n S_i) \setminus S_{n+1}$, $((\cup_{i=1}^n S_i) \setminus S_{n+1}) \cup S_{n+2}$, or $(\cup_{i=1}^n S_i) \cup S_{n+2}$, for a finite sequence of non-empty and non-universal intervals $S_1, \ldots, S_n$ with $\sup(S_i) < \inf(S_{i+1})$, for $1 \leq i < n$ and S_{n+1}, S_{n+2} non-empty disjoint finite

sets such that $S_{n+1} \subset \cup_{i=1}^{n} S_i$ and $S_{n+2} \cap (S_i \cup \{\inf(S_i), \sup(S_i)\}) = \emptyset$, for every $i \leq n$.

Example 2. $\mathbf{rnf}([-3,-1[\cup\{2,17\}\cup[8,11[\cup]11,14[)\setminus\{10\}) = (([-3,-1[\cup[8,14[)\setminus\{10,11\}) \cup \{2,17\}$.

Let $\mathcal{S}^k_{\trianglelefteq}$ denote the set $\{x \in \mathbb{R} : x \trianglelefteq k\}$, for $k \in \mathbb{R}$ and $\trianglelefteq \in \{=, \neq, >, <, \leq, \geq\}$. E.g., $\mathcal{S}^{-3}_{\geq}$ is $[-3,+\infty[$, and $\mathcal{S}^{5}_{<}$ and $\mathcal{S}^{2}_{\neq}$ are $]-\infty,5[$ and $\mathbb{R}\setminus\{2\}$. To help transform membership constraints into arithmetic constraints we introduce τ_1 that writes sets given in reduced normal form in terms of $\mathcal{S}^k_{\trianglelefteq}$'s, for suitable k's and $\trianglelefteq$'s and is defined as follows.

The map τ_1 is given by: $\tau_1(\mathbb{R}) = \mathbb{R}$, $\tau_1(\emptyset) = \emptyset$, $\tau_1(\{a_1,\ldots,a_n\}) = \cup_{i=1}^{n} \mathcal{S}^{a_i}_{=}$, $\tau_1([a,+\infty[) = \mathcal{S}^a_{\geq}$, $\tau_1(]-\infty,a]) = \mathcal{S}^a_{\leq}$, $\tau_1(]a,+\infty[) = \mathcal{S}^a_{>}$, $\tau_1(]-\infty,a[) = \mathcal{S}^a_{<}$, $\tau_1([a,b]) = \mathcal{S}^a_{\geq} \cap \mathcal{S}^b_{\leq}$, $\tau_1([a,b[) = \mathcal{S}^a_{\geq} \cap \mathcal{S}^b_{<}$, $\tau_1(]a,b]) = \mathcal{S}^a_{>} \cap \mathcal{S}^b_{\leq}$, $\tau_1(]a,b[) = \mathcal{S}^a_{>} \cap \mathcal{S}^b_{<}$, for $a, b \in \mathbb{R}$, and, $\tau_1(\mathbb{R}\setminus\{a_1 \ldots, a_n\})) = \cap_{i=1}^{n} \mathcal{S}^{a_i}_{\neq}$, $\tau_1(A\setminus\{a_1 \ldots, a_n\}) = \tau_1(A) \cap (\cap_{i=1}^{n} \mathcal{S}^{a_i}_{\neq})$, for $A \neq \mathbb{R}$, and $\tau_1(\cup_{i=1}^{n} A_i) = \cup_{i=1}^{n} \tau_1(A_i)$.

This transformation τ_1 is quite convenient to convert $f(x) \in S$ into an arithmetic constraint, for $\emptyset \neq S \neq \mathbb{R}$.

The transformation τ_2 acts on membership constraints $f(x) \in S$, for S presented in terms of $\mathcal{S}^k_{\trianglelefteq}$'s, being inductively given by: $\tau_2(f(x) \in \mathbb{R}) = (f(x) \in \mathbb{R})$, $\tau_2(f(x) \in \emptyset) = (f(x) \in \emptyset)$, $\tau_2(f(x) \in \mathcal{S}^k_{\trianglelefteq}) = (f(x) \trianglelefteq k)$, $\tau_2(f(x) \in \cup_{i=1}^{n} S_i) = (\vee_{i=1}^{n} \tau_2(f(x) \in S_i))$ and $\tau_2(f(x) \in \cap_{i=1}^{n} S_i) = (\wedge_{i=1}^{n} \tau_2(f(x) \in S_i))$. Each of these reductions between different set representations was implemented in Prolog. We reused a module developed for DEMOMATH for operating constraining sets in standard form [9]. Union, intersection and set difference are translated by `cup`, `cap` and `setminus`. Some symbolic representations were introduced for $\mathcal{S}^k_{\trianglelefteq}$, e.g., `s(real)`, `s([])`, `s(K,eq)`, `s(K,lt)`, `s(K,leq)`. Exact arithmetic for a subset of $\mathbb{R}$ is supported also by a module defined for DEMOMATH, that uses CLP($\mathbb{Q}$) for some computations [6].

For every given constraining set S (s.t. $\emptyset \neq S \neq \mathbb{R}$) and function f, we shall write $\Gamma(f(x) \in S)$ as an abbreviation of $\tau_2(f(x) \in \tau_1(\mathbf{rnf}(S)))$. Clearly, $\tau_2(f(x) \in \tau_1(\mathbf{rnf}(S)))$ is an arithmetic constraint that is equivalent to $f(x) \in S$. Because we consider that $x \in S$ is simpler than $\Gamma(x \in S)$, we introduce yet another transformation $\tilde{\Gamma}$ defining it by $\tilde{\Gamma}(id(x) \in S) = (id(x) \in \mathbf{rnf}(S))$ and $\tilde{\Gamma}(f(x) \in S) = \Gamma(f(x) \in S)$, for $f \neq id$.

Proposition 1. *For all constraining sets S, $\langle f(x) \in S, x, D\rangle$ is equivalent to $\langle \tilde{\Gamma}(f(x) \in S), x, D\rangle$.*

Example 3. If $S = [-3,-1[\cup[8,11[\cup]11,+\infty[)$, we may rewrite, $\Gamma(f(x) \in S)$ as

$$\begin{aligned}\Gamma(f(x) \in S) &= \tau_2(f(x) \in \tau_1(([-3,-1[\cup[8,+\infty[) \setminus \{11\})) = \\ &= \tau_2(f(x) \in ((\mathcal{S}^{-3}_{\geq} \cap \mathcal{S}^{-1}_{<}) \cup \mathcal{S}^{8}_{\geq}) \cap \mathcal{S}^{11}_{\neq}) = \\ &= ((f(x) \geq -3 \wedge f(x) < -1) \vee f(x) \geq 8) \wedge f(x) \neq 11\end{aligned}$$

If f is $rad_3 \circ pol_{2,-7}$, i.e., $f(x) = \sqrt[3]{2x-7}$, for solving $\langle f(x) \in S, x, \mathbb{R}\rangle$, students transform the membership constraint to arithmetic constraints. Our solver does the same thing.

Each atomic constraint $C = (f(x) \trianglelefteqslant k)$ or $C = (f(x) \trianglelefteqslant S)$, is equivalent to $(f(x) \in \mathtt{ctrSet}(C))$, for $\mathtt{ctrSet}(C)$ given by $\mathtt{ctrSet}(f(x) \in S) = S$, $\mathtt{ctrSet}(f(x) \notin S) = \mathbb{R} \setminus S$ and $\mathtt{ctrSet}(f(x) \trianglelefteqslant k) = \mathcal{S}^k_{\trianglelefteqslant}$. So, $\mathtt{ctrSet}(C)$ is a constraining set that contains $f(x)$ if C holds. We also introduce a partial function $\mathtt{nf}$ that writes some constraints to a standard form: $\mathtt{nf}(f(x) \trianglelefteqslant \beta) = (\tilde{\Gamma}(f(x) \in \mathtt{ctrSet}(f(x) \trianglelefteqslant \beta)))$ for ground β and $\mathtt{nf}(\bigotimes_{i\in I}(f(x) \trianglelefteqslant_i \beta_i)) = (\tilde{\Gamma}(f(x) \in \tilde{\bigotimes}_{i\in I}\mathtt{ctrSet}(f(x) \trianglelefteqslant_i \beta_i)))$ for ground β_i. Here $\tilde{\vee} = \cup$ and $\tilde{\wedge} = \cap$.

4 Solving Problems

To design pedagogically relevant solvers we cannot manipulate problems and constraints in an arbitrary way. The *rewrite rules* we propose use some extra mathematical knowledge, e.g. about functions behavior, and, if applicable, transform a problem into an equivalent one, under some specific conditions. For instance, the rule BOUNDRANGE checks whether an atomic constraint is valid or inconsistent based on functions range. It states that: *for any generic functions f and g, $f \neq id$, and any ground set-term or arithmetic-term β, if $D \subseteq \mathcal{D}_{f\circ g}$ and $\mathcal{E}$ is such that $f(\mathcal{D}_f) \subseteq \mathcal{E}$ then*

$\langle (f \circ g)(x) \trianglelefteqslant \beta, x, D\rangle \rightarrow \langle x \in D, x, D\rangle$ **if** $\mathtt{ctrSet}((f \circ g)(x) \trianglelefteqslant \beta) \supseteq \mathcal{E}$;
$\langle (f \circ g)(x) \trianglelefteqslant \beta, x, D\rangle \rightarrow \langle x \in \emptyset, x, \emptyset\rangle$ **if** $\mathtt{ctrSet}((f \circ g)(x) \trianglelefteqslant \beta) \cap \mathcal{E} = \emptyset$;
$\langle (f \circ g)(x) \trianglelefteqslant \beta, x, D\rangle \rightarrow \langle \tilde{\Gamma}((f \circ g)(x) \in S \cap \mathcal{E}), x, D\rangle$
if $\trianglelefteqslant \notin \{=, \neq\}$, $\emptyset \neq S \cap \mathcal{E} \neq S$ *and* $S \not\supseteq \mathcal{E}$, *where* $S = \mathtt{ctrSet}((f \circ g)(x) \trianglelefteqslant \beta)$.

The rewrite rules look like $P \rightarrow P'$ *if condition* although some preconditions were stated in a global head. This kind of mathematical representation does not make clear the intended operational reading of each rule. Implicit meta-knowledge should be made explicit in order to be able to explain solution steps. Because of that, and to gain also more flexibility to modify rules, add new ones and customize solvers to different users or curricula, we developed a language for specification of rewrite rules. The corresponding formulation of BOUNDRANGE looks as follows. Relevant conditions for writing explanations are annotated with (#).

```
BOUNDRANGE(P)
  begin
    is_atomic(P:ctr), is_ground(P:ctr:rhs),
    subseteq(func_dom(P:ctr:lhs:func),P:dom),
    (#)P:ctr:lhs:func =? F ∘ G, !F =? id,
    E := (#)boundImage(F,func_dom(F)),
    S := (#)ctrSet(P:ctr)
    if (#)supseteq(S,E), (#)note("valid %", P:ctr)
      rewrite_to sfprob(P:var,inset,P:dom)
    elif (#)seteq(S cap E,s([])), (#)note("inconsistent %", P:ctr)
      rewrite_to sfprob(P:var,inset,s([]))
    else !inlist(P:ctr:op,[eq,neq]),
      (#)note("necessarily %", ctr(P:ctr:lhs:func,P:var,inset,E cap S)),
```

```
      !seteq((#)rnf(E cap S),S) rewrite_to
      prob(tgm(ctr(P:ctr:lhs:func,P:var,inset,E cap S)),P:var,P:dom)
    endif
  end
```

The specification language is a functional language with implicit types. Primitive (data)types are **boolean, real, set, function, constraint** and **problem**. All built-in constructs are typed and every rule definition must be type checked. Due to space limitations we can not describe its details in this paper. The definition of a rule consists of a name, a parameter (of type **problem**) and a sequence of conditions followed either by a nested **if_block** or by a **rewrite_to exp**, where **exp** corresponds to the resulting problem, if no condition is *false*. Atomic conditions will allow the specification and the verification of mathematical knowledge as relations between functions, sets and real numbers; equality of problems or constraints; properties of functions; transformations and computations, etc. Each rule is compiled to a Prolog predicate. The **if**-block is translated to an auxiliary predicate, whose clauses correspond to the branches of the **if**-block. A single branch may succeed. Besides defining the rewrite rules, we need to specify how they are applied for solving problems. For that we use the notion of strategy [4]. A trivial strategy is to try to apply all available rules until either a solved form or an upper bound on the number of steps (rule applications) is reached. But other strategies may be defined.

4.1 Cognitive faithful rewriting rules

We now present some of the rewrite rules, that contribute to the novelty of this work. The whole set is complete for a set of problems arising in high-school math curricula and that can be generated by DEMOMATH. The grammar that describes the arithmetic expressions involved in them is presented in [9]. For space reasons, we omit their formal definition, except for a few, presenting their aim instead. We start by REDUCEPROBDOMAIN, that says that solutions must be in $D \cap \mathcal{D}_C$. Then, we give four rules for handling complex constraints and the rules for atomic constraints, omitting BOUNDRANGE.

ReduceProbDomain To guarantee that solutions are in $D \cap \mathcal{D}_C$.
SplitConstraints To rewrite several top level conjuncts (or disjuncts).
AggregateNormalize To rewrite several atomic constraints $f(x) \trianglelefteq_i \beta_i$, that occur at top level, to a simpler form (may detect inconsistency/validity).
Conjunctive To rewrite a single conjunct at top level.
Disjunctive To rewrite a single disjunct at top level.
ArithNormalize To convert a single membership constraint to an arithmetic constraint if the latter is simpler.
DefRealValuedFunc To rewrite membership constraints $f(x) \in S$ or $f(x) \notin S$ for $S = \emptyset$ or $S = \mathbb{R}$, to solved form.
DomainAtomConstr To rewrite a constraint $(x \trianglelefteq \beta)$ to solved form.

ConstantFunc To rewrite a constraint involving the constant function to a constraint $f(x) \trianglelefteqslant k$ or to a solved form.

StrictMonotonic To rewrite $(f \circ g)(x) \trianglelefteqslant k$ to a simpler form when f is strictly monotonic.

AxialSymMonotonicBranch To rewrite $(f \circ g)(x) \trianglelefteqslant k$ to a simpler form when f is symmetric w.r.t. $x = a$, strictly monotonic on $\mathcal{D}_f^{\geq a}$ (i.e., the set of points in $\mathcal{D}_f$ that are greater than or equal to a).

We also introduce five specific rules **AffineTransf, Power, AbsoluteValue, Quadratic, Radix** to rewrite $(f \circ g)(x) \trianglelefteqslant k$ to a simpler form, when f is $pol_{a,b}$, pow_n, abs, $pol_{a,b,c}$ and rad_n. Although they are instances of the **StrictMonotonic** and **AxialSymMonotonicBranch**, these more advanced rules are best suited for handling generic functions, once students have already studied their behavior. For all but $pol_{a,b}$ and pow_{2n+1} (whose range is $\mathbb{R}$), conditions are imposed to disallow their application if it can be trivially deduced that the constraint is inconsistent (by using **BoundRange**). For example, **AbsoluteValue** is defined by $\langle (abs \circ f)(x) \trianglelefteqslant k, x, D\rangle \rightarrow \langle \tilde{\Gamma}(f(x) \in \mathcal{B}^k_{0,\trianglelefteqslant}), x, D\rangle$ **if** $k \in \mathbb{R}$ and $k \geq 0$, where $\mathcal{B}^{\delta}_{a,\trianglelefteqslant}$ is $\{x \in \mathbb{R} : |x - a| \trianglelefteqslant \delta\}$. That is, $\mathcal{B}^{\delta}_{a,\trianglelefteqslant} = S^{a+\delta}_{\trianglelefteqslant} \cup S^{a-\delta}_{\trianglelefteqslant^{-1}}$, if $\delta > 0$, and $\trianglelefteqslant \in \{\geq, >\}$, $\mathcal{B}^{\delta}_{a,\trianglelefteqslant} = S^{a+\delta}_{\trianglelefteqslant} \cap S^{a-\delta}_{\trianglelefteqslant^{-1}}$, if $\delta > 0$ and $\trianglelefteqslant \in \{\leq, <\}$, and so forth. The following rules handle constraints involving sum, product, difference and quotient of functions and also the piecewise function.

Piecewise To replace a constraint that involves a piecewise function f, given by $f \equiv (f_i, D_i)_{i=1}^n$, by a disjunctive constraint induced by the relevant branches f_i's.

ProductByConstant To rewrite $(c_k \times f)(x) \trianglelefteqslant k'$ to a simpler form.

DiffSquare To factorize a difference of two squares $(pow_N \circ g - pow_M \circ h)(x) \trianglelefteqslant 0$, for N and M even.

NullProduct To simplify a constraint by applying the rules for null product and sign of a product.

FactMonotonic To simplify constraints $(f \circ g)(x) \trianglelefteqslant (f \circ h)(x)$ when $f \neq id$ is strictly monotonic. It is useful for solving $rad_n(X) \leq rad_n(Y)$ for instance.

FactOdd To simplify $(f \circ g)(x) \trianglelefteqslant ((-f) \circ h)(x)$ when $f \neq id$ is odd. It states that: $\langle (f \circ g)(x) \trianglelefteqslant ((-f) \circ h)(x), x, D\rangle \rightarrow \langle (f \circ g)(x) \trianglelefteqslant (f \circ (-h))(x), x, D\rangle$ **if** $f \neq id$ is an odd function and $g \neq h$.

ToHomQuotient To rewrite $(f/g)(x) \trianglelefteqslant k$ to $(f - c_k \times g)/g)(x) \trianglelefteqslant 0$.

DiffMono To rewrite $(f \circ g - f \circ h)(x) \trianglelefteqslant 0$ to $(f \circ g)(x) \trianglelefteqslant (f \circ h)(x)$ when f is strictly monotonic.

SignDiff To rewrite $f(x) \trianglelefteqslant g(x)$ to $(f - g)(x) \trianglelefteqslant 0$.

SumNull To simplify $(f + g)(x) = 0$ and $(f + g)(x) \neq 0$ when the ranges of f and g are both in $\mathbb{R}_0^-$ or $\mathbb{R}_0^+$.

SquarePol To simplify constraints of form $(rad_2 \circ f - g)(x) \trianglelefteqslant 0$ and $(rad_2 \circ f)(x) \trianglelefteqslant g(x)$.

We need the last rules to guarantee the solvers completeness for the expressions that DEMOMATH creates. The solvers will not support user-defined expressions

$f(x)$, unless they may be *recognized* by the system. Simple algebraic manipulations may be carried out to express f in terms of a different combination of primitive functions.

Applications of CAL to math education require a careful analysis of procedures that students usually apply to solve math drills to design generic solvers with pedagogic relevance. We claim that solvers based on the proposed rewrite rules set fulfills this requirement. The system is being implemented in Prolog.

References

1. H. Barendregt. Towards an Interactive Mathematical Proof Language. in F. Kamareddine (ed.), *Thirty Five Years of Automath*, Kluwer (2003) 25-36.
2. M. Beeson. Design Principles of Mathpert: Software to support education in algebra and calculus. In N. Kajler (ed.), *Computer-Human Interaction in Symbolic Computation*, Texts and Monographs in Symbolic Computation, Springer-Verlag (1998), 89-115.
3. A. Bundy. *The Computer Modelling of Mathematical Reasoning*. A. Press (1983).
4. H. Cirstea, C. Kirchner, L. Liquori, and B. Wack. Rewrite strategies in the rewriting calculus. *Electr. Notes Theor. Comput. Sci.*, 86:4 (2003).
5. H. Gottliebsen, T. Kelsey, U. Martin. Hidden Verification for Computational Mathematics. *J. Symbolic Computation* 39 (2005) 539-567.
6. C. Holzbaur. OFAI clp(q,r) Manual, Edition 1.3.3. Austrian Research Institute for Artificial Intelligence, TR-95-09, Vienna (1995).
7. E. Melis, E. Andrés, J. Büdenbender, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, C. Ullrich. ActiveMath: A Generic and Adaptive Web-Based Learning Environment, *Int. J. of AI in Education* 12:4 (2001) 385–407.
8. A. Robinson, A. Voronkoy (Eds). *Handbook of Automated Reasoning*, Elsevier Science (2001).
9. A. P. Tomás, J. P. Leal. A CLP-Based Tool for Computer Aided Generation and Solving of Maths Exercises. In V. Dahl, P. Wadler (Eds), *Practical Aspects of Declarative Languages, 5th Int. Symposium PADL 2003*, LNCS 2562, Springer-Verlag (2003) 223–240.
10. G. Xiao. WIMS – An Interactive Mathematics Server, *Journal of Online Mathematics and its Applications* 1, MAA (2001).