CS365: Homework 2

(Computer-Aided Instructions)

Automatically Generating Algebra Problems

(Sumit Gulwani, Sriram Rajamani, Rohit Singh)

SUMMARY

Goal:

In this paper, another aspect of education namely 'problem generation' has been automated i.e. given an algebraic proof problem *p* involving equalities (of the form LHS = RHS), how to automatically generate the problems 'similar' to *p*, by allowing the user to define the notion of 'similar' by fine-tuning. The focus is on providing personalization to a student trying to learn a particular concept where a student interacts with the computer to generate the problems which are of interest to him/her and according to his/her notion of similarity [2]. The methodology presented generates unique, fresh and interesting problems as the user wants.

Importance of Research problem:

1) For a teacher, giving the *fresh* problems to students that uses same set of concepts and same level of difficulty is a tedious task. This paper directly targets this problem. [3]
2) Although there are many online sites that provide exercise material, but they provide only a fixed set of exercises which doesn't provide enough 'personalization' to a learner trying to grasp a concept. By allowing a user to interact with computer and generate the problems according to his/her idea of similarity, user gets to solve fresh, interesting problems of same difficulty level and same concepts. This is one of the central focuses of this paper. [3]

Previous Work on this problem:

There were two approaches to generate similar problems:

a) Flexibility provided for instantiating parameters of a problem with random constants. But this was done only for the constant in problem and hence it doesn't generate interesting problems.
b) Certain features of the problem domain are provided as hard-coded options and users are able to choose among these options and generate problems. Some examples of feature include for example in quadratic domain: "simple factorable", "difficult factorable". However this approach is limited to only simple algebraic domains like counting, or linear and quadratic equation solving. [3]

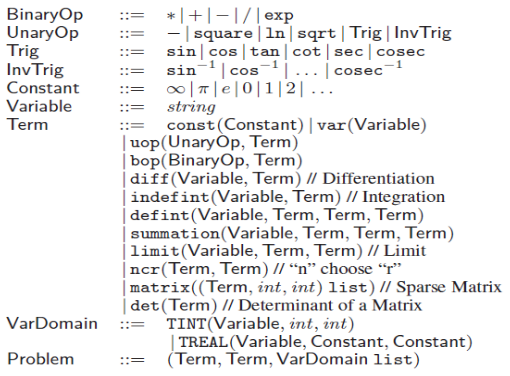
Main Contributions of work presented in this paper:

a) The methodology presented is applicable to large sub-fields of algebra.
b) The user can interactively fine tune the notion of similarity according to his/her tastes and needs and generate problems.

Brief Overview of the methodology:

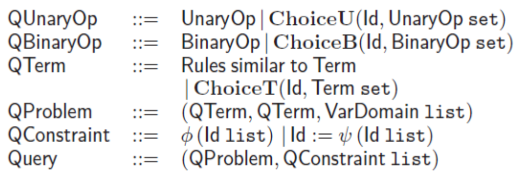
A Query language (fig. 1 and fig. 2 [3]) has been defined using which the problems are represented and generated by the following methodology.

1) Query Generation:
   Given a problem *p* it is associated with an abstract set of problems (denoted by [[Q]]). This step generates the abstract space [[Q]] by recursively defined notion (using tree structure) of a 'term', a 'QProblem', variables domain and query constraints (like denominator! =0 etc.). Some query constraints are kept as default like if two numbers are divided then their gcd = 1 is kept as constraint besides many other. Further, it has some set of operations (like Unary op., binary op. etc.) predefined that are abstractly placed in tree nodes and by replacing the choice nodes for some value in set, a large [[Q]] space is generated. A sample problem generated is in figure 3 [3]



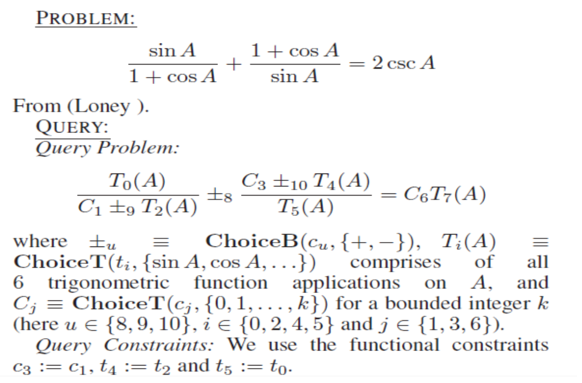
```
BinaryOp    ::=  * | + | − | / | exp
UnaryOp     ::=  − | square | ln | sqrt | Trig | InvTrig
Trig        ::=  sin | cos | tan | cot | sec | cosec
InvTrig     ::=  sin⁻¹ | cos⁻¹ | . . . | cosec⁻¹
Constant    ::=  ∞ | π | e | 0 | 1 | 2 | . . .
Variable    ::=  string
Term        ::=  const(Constant) | var(Variable)
                | uop(UnaryOp, Term)
                | bop(BinaryOp, Term)
                | diff(Variable, Term) // Differentiation
                | indefint(Variable, Term) // Integration
                | defint(Variable, Term, Term, Term)
                | summation(Variable, Term, Term, Term)
                | limit(Variable, Term, Term) // Limit
                | ncr(Term, Term) // "n" choose "r"
                | matrix((Term, int, int) list) // Sparse Matrix
                | det(Term) // Determinant of a Matrix
VarDomain   ::=  TINT(Variable, int, int)
                  | TREAL(Variable, Constant, Constant)
Problem     ::=  (Term, Term, VarDomain list)
```

Figure 1: Syntax of Algebra Proof Problems.



```
QUnaryOp     ::=  UnaryOp | ChoiceU(Id, UnaryOp set)
QBinaryOp    ::=  BinaryOp | ChoiceB(Id, BinaryOp set)
QTerm        ::=  Rules similar to Term
                  | ChoiceT(Id, Term set)
QProblem     ::=  (QTerm, QTerm, VarDomain list)
QConstraint  ::=  φ (Id list) | Id := ψ (Id list)
Query        ::=  (QProblem, QConstraint list)
```

Figure 2: Syntax of Query Language.



PROBLEM:

$$\frac{\sin A}{1+\cos A} + \frac{1+\cos A}{\sin A} = 2\csc A$$

From (Loney).

QUERY:

Query Problem:

$$\frac{T_0(A)}{C_1 \pm_9 T_2(A)} \pm_8 \frac{C_3 \pm_{10} T_4(A)}{T_5(A)} = C_6 T_7(A)$$

where $\pm_u \equiv \mathbf{ChoiceB}(c_u, \{+,-\})$, $T_i(A) \equiv \mathbf{ChoiceT}(t_i, \{\sin A, \cos A, \ldots\})$ comprises of all 6 trigonometric function applications on A, and $C_j \equiv \mathbf{ChoiceT}(c_j, \{0, 1, \ldots, k\})$ for a bounded integer k (here $u \in \{8, 9, 10\}$, $i \in \{0, 2, 4, 5\}$ and $j \in \{1, 3, 6\}$).

Query Constraints: We use the functional constraints $c_3 := c_1$, $t_4 := t_2$ and $t_5 := t_0$.

Figure 3: Example of a query problem

2) Query Execution:
By executing the query Q, a subset of 'valid', 'correct' and 'unique' problems is extracted using some methods. Also some Backtracking is done to reduce the size and time of problem generation.
3) Query Tuning:
'The user can remove some automatically generated constraints or improve the generalization by adding more choices in existing choice nodes or new choice nodes to the query' to generate the problems according to the interest and notion of similarity.

RESULTS: For the 5 algebra domain, the testing was done and a number of similar problems were generated. Some problems were unique and were not present in the text. Manual validation of each problem generated was done to ensure that the problems are indeed interesting and correct. The query execution required atmost one iteration to generate the valid and correct problems.

REFERENCES:

1. Computer Aided Learning: http://en.wikipedia.org/wiki/Computer-supported_collaborative_learning
2. Personalization Learning: http://en.wikipedia.org/wiki/Personalized_learning
3. Gulwani, S., Rajamani, S., Singh, R. 2012: Automatically Generating Algebra Problems, Association for the Advancement of Artificial Intelligence.
4. Singh, R., and Gulwani, S. 2012a. Learning semantic string transformations from examples. Very Large Databases (VLDB) 5.
5. Gulwani, S.; Korthikanti, V. A.; and Tiwari, A. 2011. Synthesizing geometry constructions. In Programming Language. Design and Implementation (PLDI), 50–61.