Summer Undergraduate Research Grant for Excellence, 2012
Indian Institue of Technology,Kanpur

Automatic Problem Generation in Logic

Ashudeep Singh[*]
ashudeep@iitk.ac.in
Computer Science And Engineering,
IIT Kanpur - 208016

Dr. Amey Karkare Dr. Subhajit Roy
{*karkare,subhajit*}*@cse.iitk.ac.in*
Computer Science And Engineering,
IIT Kanpur - 208016

12th July 2012

[*]Work Done along with Pulkit Jain (pulkitj@iitk.ac.in) during SURGE Programme 2012

ABSTRACT

With the increasing student to instructor ratio, it has become the need of the hour to automate the process of education. In this process, Intelligent Tutoring Systems have played a significant role. ITSs provide interfaces with features like solving problems, generating problems, generating hints, grading students' answers etc. In this project we try to make an attempt to develop such a system which can automatically generate problems for the course of "Introduction to Logic". We believe that such a system will be helpful for the instructors, who have to do the tedious job of creating new problems relating to the ones discussed in the class or that appeared in the previous examinations, as well as for the motivated students who are eager to solve more and more problems. We present a method to generate problems from a given problem and a method to generate multiple solutions for a given problem and utilize it to provide hints to a student while solving the problem. The solution generation can also be used along with problem generation where the instructor/student gets to personalize the problem, on the basis of rules to be used in its solution so that a student can practice more and more problems of a kind, which is not possible in a tutor with fixed database of problems. Also, we discuss about the application of solution generation as an automatic grader.

Contents

1 Introduction

Intelligent tutoring systems share a common goal of providing a proper learning interface to the learner and an interface for the instructor to reduce tedious jobs like grading, creating new problems relating to what was taught in the class etc. The important components of these systems are- the problems presented to the user and the solution generated by the system. In this report we present the methods adopted in our attempt to develop a system for generating problems in Logic and the solution steps required to solve a given problem. A system that could generate proof problems in "Algebra" from a given problem belonging to particular domains [1], was presented in A.A.A.I. Conference, 2012. The proposed method tried to first present a given problem in form of a query, which was general for such problems, and could then be used to find alternatives to that problem.
The problems referred to in this project are the proof problems, that require the use of some deduction rules to deduce the conclusion from premises. A problem comprises of a set of Premises $\{P_1, P_2, P_3, ...P_n\}$ and a conclusion C. We are required to show a proof that contains the steps of deducing conclusion from premises using the Problem Generation is done in three ways,first from a given example problem,second generating a random problem and third ,producing problems that require the use of some particular deduction rules.

2 A Problem

A deduction problem in Propositional Logic is defined as a set of premises and a conclusion, that follows from them. Thus $\{P_1, P_2, ..., P_n, C\}$ is a problem.
These conditions must be satisfied by the set of premises and conclusion in the problem:

- $P_1 \wedge P_2 \wedge P_3 \wedge .... \wedge P_n \Rightarrow C$
- $P_{i_1}, P_{i_2}, P_{i_3}, ...P_{i_k} \not\Rightarrow C$, where $0 < k < n$ where $n > 1$ i.e. Any subset of P_is doesn't imply the conclusion. This can be seen as every premise contributing to deduce the conclusion.

The following is the approach we followed for representing these formulas.

2.1 Representing Truth Tables

Each Premise P_i is made out of variables $\{V_1, V_2, ..., V_m\}$, we store the truth table of P_i, as mentioned below.We then represent them as binary strings and store them as integers.

Consider the premise $A \Rightarrow (B \wedge C)$ constructed out of the variables $\{A, B, C\}$
Each of the variables can acquire a logical ''TRUE'' (represented as 1) or a logical ''FALSE'' (represented as 0). The corresponding Truth-Table computed is

A	**B**	**C**	$\mathbf{B \wedge C}$	$\mathbf{A \Rightarrow (B \wedge C)}$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 1. The truth table of the expression $\mathbf{A \Rightarrow (B \wedge C)}$. Here 0s represent False and 1s represent True.

Here,
Truth-Table Representation of A is seen as 00001111 which is binary notation of 31
Truth-Table Representation of B is seen as 00110011 which is binary notation of 51
Truth-Table Representation of C is seen as 01010101 which is binary notation of 85
Truth-Table Representation of $B \wedge C$ is seen as 00010001 which is binary notation of 17
Truth-Table Representation of $A \Rightarrow (B \wedge C)$ is seen as 11110001 which is binary notation of 241

The initial Truth-Tables of variables can be computed using the below. Also, the size of the truth table will be 2^m rows if there are m variables $\{V_1, V_2, V_3, V_4 ....V_m\}$. Following matrix Var(i,j) of size,m x 2^m helps us in getting the Truth-Tables of variables.

$$\text{Var[i,j]} = \frac{1-(-1)^{\lfloor j/2^i \rfloor}}{2}$$

where j^{th} column of i^{th} row represents the j^{th} bit in the Truth-Table representation of the i^{th} variable. Each row of the truth table will represent the binary representation of the variables and then are stored as 2^m-bit integers. i.e. a problem with n-premise out of m-variables will be stored as n integers of m-bits each. The limitation on only 32-bit integers can be tackled by storing multiple integers, which will represent the binary form of the variables in problems with more than 5-variables.

2.2 Advantages of this Representation

The major advantage of using integer arrays over boolean arrays to store Truth-Tables are:

- It facilitates the conversion of integers back to formulas without going back to the truth tables because from truth table to formulas we can only get SOP or POS forms through Karnaugh maps, which will lead to unnecessary calculations.
- Bitwise operations over integers is easier than to implement the logical AND,OR and NEGATION over Truth-Tables

3 Generating a Problem from an Example problem

One way to find similar problems to a given, we can substitute one or more of the given set of premises and conclusion.The following is the method to find the substitutes for premises and conclusion.

3.1 Replacing Premise

Substituting a premise requires finding a new Truth-Table and hence an integer or an array of integers that substitute the original premise. The substitute P_i' of P_i has to satisfy the following constraints:

$$\begin{gathered}
P_1 \wedge P_2 \wedge P_3 .. \wedge P_i' ... \wedge P_n \Rightarrow C \\
\text{and} \\
P_2 \wedge P_3 \wedge P_4 ... \wedge P_i' ... \wedge P_n \nRightarrow C \\
P_1 \wedge P_3 \wedge P_4 ... \wedge P_i' ... \wedge P_n \nRightarrow C \\
P_1 \wedge P_2 \wedge P_4 ... \wedge P_i' ... \wedge P_n \nRightarrow C \\
. \\
. \\
P_1 \wedge P_2 ... \wedge P_{i-1} \wedge P_{i+1} ... \wedge P_n \nRightarrow C \\
. \\
. \\
P_1 \wedge P_2 \wedge P_3 ... \wedge P_i' ... \wedge P_{n-1} \nRightarrow C
\end{gathered}$$

These constraints are to be kept in mind to satisfy the conditions mentioned above on Page 4.
The first constraint listed above states that the substitute premise P_i' should be such that the new set of premises still implies the conclusion. We call this the ZERO CONSTRAINT.

The rest of the constraints account for the fact that only all premises taken together should imply the conclusion and any subset of these premises must not imply the conclusion.

3.1.1 ZERO CONSTRAINT for Premises

$$P_1 \wedge P_2 \wedge P_3 .. \wedge P_i' ... \wedge P_n \Rightarrow C$$

The constraint can be easily solved at the level of Truth-Tables.
For the set $\{P_1, P_2, ..., P_i', ..P_n\}$ to lead to the conclusion, all indexes which are set to 1 in $P_1 \wedge P_2 \wedge P_3 ... \wedge P_{i-1} \wedge P_{i+1} ... \wedge P_n$ and 0 in C should be 0 in P_i'.
This follows from the logical implication rules

FALSE $\Rightarrow$ FALSE is TRUE
FALSE $\Rightarrow$ TRUE is TRUE
TRUE $\Rightarrow$ FALSE is FALSE
TRUE $\Rightarrow$ TRUE is TRUE

3.1.2 NOT-P_k CONSTRAINTS for Premises

The NOT-P_k CONSTRAINTS can also be easily dealt with using the above stated logical implication rules. To solve these constraints we are going to use the following lemma

Lemma : If $(A \wedge B) \not\Rightarrow C, then (A \not\Rightarrow C) \wedge (B \not\Rightarrow C)$

Proof : Consider the premise $(A \wedge B) \Rightarrow C$

$$(A \wedge B) \Rightarrow C \Leftrightarrow \neg(A \wedge B) \vee C \qquad \text{(By Material Implication)}$$

$$\neg(A \wedge B) \vee C \Leftrightarrow \neg A \vee \neg B \vee C \vee C \qquad \text{(By DeMorgans Law)}$$

$$\neg A \vee C \vee \neg B \vee C \Leftrightarrow (A \Rightarrow C) \vee (B \Rightarrow C) \qquad \text{(By Material Implication)}$$

$$\therefore (A \wedge B) \Rightarrow C \Leftrightarrow (A \Rightarrow C) \vee (B \Rightarrow C)$$

For $(A \wedge B) \Rightarrow C$ to be FALSE $(A \Rightarrow C)$ and $(B \Rightarrow C)$ must be FALSE.

Hence $(A \wedge B) \not\Rightarrow C$, then $(A \not\Rightarrow C) \wedge (B \not\Rightarrow C)$

The new set of premises $\{P_1, P_2, ..., P_i', ..P_n\}$ should be such that no subset of premises implies the conclusion, then using the above lemma, it is sufficient to for us to verify that

$$P_2 \wedge P_3 \wedge P_4 ... \wedge P_i' ... \wedge P_n \not\Rightarrow C \qquad ..(1)$$

$$
\begin{array}{lr}
P_1 \wedge P_3 \wedge P_4 ... \wedge P_i' ... \wedge P_n \nRightarrow C & ..(2) \\
P_1 \wedge P_2 \wedge P_4 ... \wedge P_i' ... \wedge P_n \nRightarrow C & ..(3) \\
. & \\
. & \\
P_1 \wedge P_2 ... \wedge P_{i-1} \wedge P_{i+1} ... \wedge P_n \nRightarrow C & ..(\mathrm{i}) \\
. & \\
. & \\
P_1 \wedge P_2 \wedge P_3 ... \wedge P_i' ... \wedge P_{n-1} \nRightarrow C & ..(\mathrm{n})
\end{array}
$$

LHS of the $NOT-P_1$ constraint (equation 1) is the conjunction of all premises except P_1. Using the lemma stated, no subset of the the remaining premises leads to the conclusion.
For this to be satisfied, at least one of the indexes where the conjunction of all n-2 premises except P_1 and P_i is 1 and C is zero, P_i' should be 1.
All the remaining equations are similar except for the equation (i).
If we are substituting the premise P_i,then we have no control over this equation and in this case we assume the equation to be satisfied.And since,substitutes of any of the premises may have to be computed,so the equation(i) should be valid $\forall i$. Thus the example problem should itself also satisfy the constraints that are to be satisfied by the produced problem.This is a limitation to the example problem that the user enters.

3.2 Replacing Conclusion

Similar to finding substitutes of a premise,here too we have to find substitute Truth-Tables for the conclusion and we have similar equations and thus similar ZERO CONSTRAINT and NOT-P_k CONSTRAINTS that require to be satisfied by a Truth-Table in order to be a substitute. The equations to be satisfied are

$$
\begin{array}{l}
P_1 \wedge P_2 \wedge P_3 .. \wedge P_i ... \wedge P_n \Rightarrow C' \\
\text{and} \\
P_2 \wedge P_3 \wedge P_4 ... \wedge P_i ... \wedge P_n \nRightarrow C' \\
P_1 \wedge P_3 \wedge P_4 ... \wedge P_i ... \wedge P_n \nRightarrow C' \\
P_1 \wedge P_2 \wedge P_4 ... \wedge P_i ... \wedge P_n \nRightarrow C' \\
. \\
. \\
P_1 \wedge P_2 \wedge P_3 ... \wedge P_i ... \wedge P_{n-1} \nRightarrow C'
\end{array}
$$

3.2.1 ZERO CONSTRAINT for Conclusion

As in case of replacing premises, the basic condition of the ZERO CONSTRAINT remains the same. For the new conclusion to be concluded by the premises, all indexes for which conjunction of all premises is 1 should be set to 1 in the substitute conclusion C'.

3.2.2 NOT-P_k CONSTRAINTS for Conclusion

The NOT-P_k CONSTRAINTS here too mean to achieve the same condition. The conclusion should be concluded only from the complete set of n premises and not from any subset of it. The equation corresponding to NOT-P_1 Constraint requires that C' should not be implied without the premise P_1.For this the indexes at which the conjunction of all premises except P_i is 1 , C' should be 0. And the rest of the NOT-CONSTRAINTS are similar.

Above stated constraints give us an integer representation of the required Truth-Table. Now we need to reconstruct the logical formula that represents the Truth-Table ,given the variables out of which the formula has to be constructed. One way of doing this could be finding the Sum of Products or Product of Sums of the given Truth-Table and then use Karnaugh Maps to reduce it. But this is unnecessary and tedious task to be done. Hence we precomputed the integers of all small formulas and the formula corresponding to a integer needs not to be calculated every time it is required.By "small" formulas,we refer to those formulas which are not too long in their string representation. Ex: $P \Rightarrow (Q \wedge (P \vee \neg Q)))$ may not be a desirable formula in two variables P,Q. Thus now we can find substitutes for premises and the conclusion, so we can get a new problem by replacing a number of premises and/or conclusion from the given example problem. Next we discuss generating a problem from scratch.

3.3 Results and Observations

Firstly, as in Figure 1, the number of replacements for premises/conclusion vary from problem to problem. But, as in Figure 2, when the average number of replacements per problem were considered against number of variables, the trend show that the number increases to almost 16 times on increase of variables from 3 to 4.
Similarly comparing average time against number of variables(Figure 4), we observe the avg. time for all replacements almost increasing by 25 times on going from problems with 3 variables to 4 variable problems. The brute-force method would have lead to increase in time by 2^8 times from 3 to 4. But on applying the constraints we hugely reduce the time to a mere 25 times.

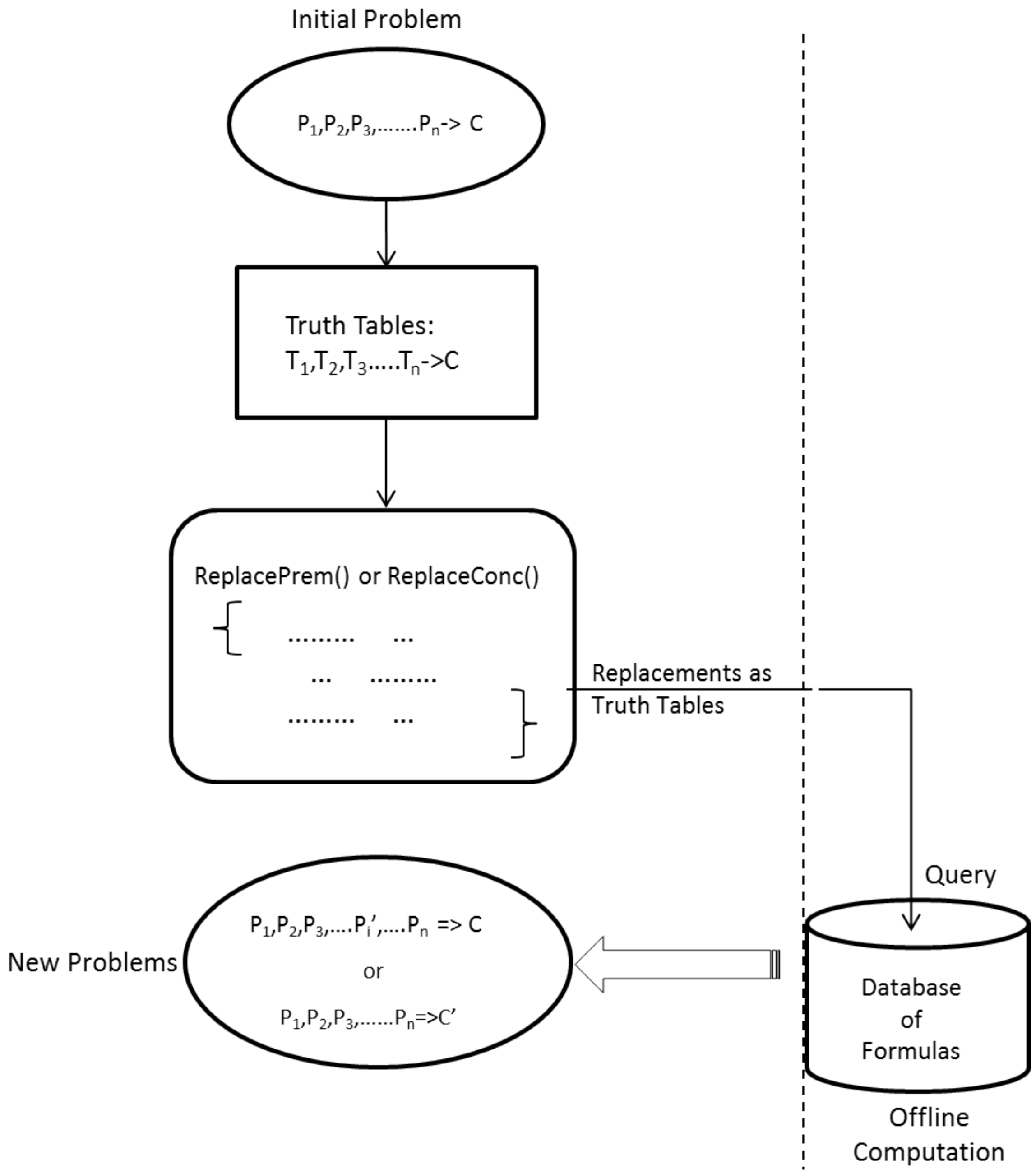

Figure 1: The above flowchart describes the adopted approach for replacing a premise or a conclusion.

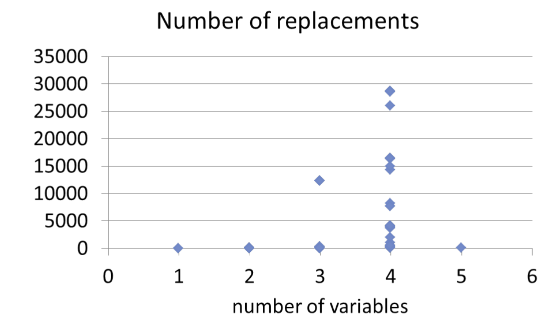

Figure 2: Number of Replacements vs number of variables

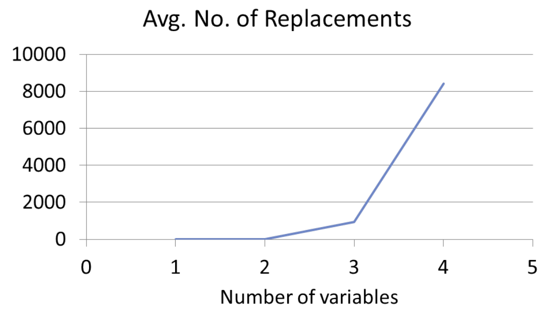

Figure 3: Average Number of Replacements vs number of variables

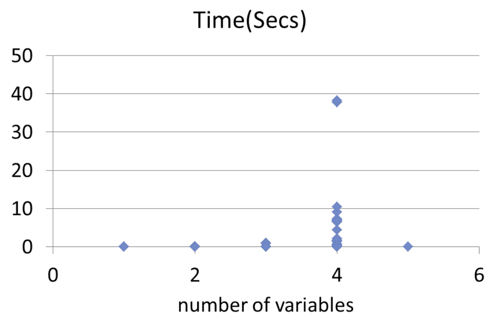

Figure 4: Time vs number of variables

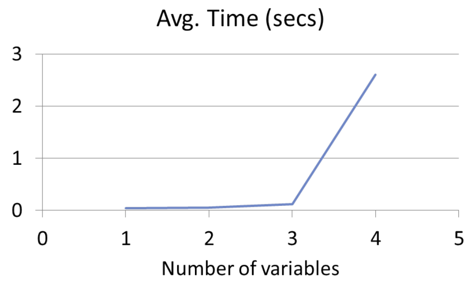

Figure 5: Average Time vs number of variables

4 Generating a Problem from Scratch

Generating problems from scratch requires us to produce a set of premises and a conclusion in which each of the premises and conclusion satisfies the ZERO CONSTRAINT and the NOT-P_k CONSTRAINTS. Since each Truth-Table is represented as an integer array, it becomes easier for us to be able to generate a problem. The starting parameters of problem generation is the number of premises to be generated and the number of variables to be used in constructing the premises and the conclusion.
Suppose we want to generate N premises out of K variables. Now all possible Truth-Tables that can be constructed using K variables will have their integer representation lying in the range [0,2^K-1] So we begin with a choosing a random integer c in the above range which represents the conclusion,another random integer p_1 representing the premise P_1,such that the corresponding Truth-Tables T_{p_1}, T_C satisfy $T_{p_1} \nRightarrow T_C$. This can be easily checked using the bitwise operations on the corresponding integers. We keep on adding integers $p_1, p_2, p_3, ...p_{n-1}$ such that $\forall(T_{p_i}, T_{p_j}), T_{p_i} \nRightarrow T_{p_j}$.This constraint ensures that no premise can be directly derived from some other premise. At last, we find the $n_t h$ premise by finding a substitute for the $n_t h$ premise,as described earlier.
Now we have a set of Truth-Tables and hence logical formulas that satisfy the required constraints.

We next discuss the procedure of finding the solution steps required to derive the conclusion from the premises before the last interface of problem generation,since it requires solution generation part to be used in it.

5 Finding Solution Steps for Deriving the conclusion

Given a set of premises and conclusion,the goal is to a find an ordered set of rules that applied on the premises and the intermediate results, eventually leads to the conclusion. For this we need to know all the deduction rules or axioms that are used in solving the problems. As stated in [3],there are 18 rules that can be used in the process of deducing a conclusion from premises. Some of these are

Modus Ponens Rule : $P \Rightarrow Q$, P , $\therefore Q$
Modus Tollens Rule : $P \Rightarrow Q$, $\neg P$, $\therefore Q$
Hypothetical Syllogism : $P \Rightarrow Q$, $Q \Rightarrow R$, $\therefore P \Rightarrow R$
Disjunctive Syllogism : $P \vee Q$, $\neg P$, $\therefore Q$

Let us call these rules as $f_1, f_2, f3....., f_{18}$
These rules can be seen as a map from a tuple of formulas to a tuple of formulas. Most of the rules map a pair of logical formula to a single formula or a single formula

to other formula. We proceed by applying a rule on a pair of premises to get another premise and then recursively solve the new set of premises to reach the conclusion. Keeping a track of the rules applied and premises used in respective rules gives us a step-wise solution to deduce the conclusion.
This method of exhaustive search over possible applications of rules on possible combination of premises produces multiple solutions to same problem.
The mappings of formulas under various rules need to done before-hand. We use formulas as strings and not integers since rules that map a formula to its equivalent forms (formulas have been said equivalent if they have the same Truth-Table and hence same integer representation Ex : $\neg P \vee Q$ and $P \Rightarrow Q$) will not be accounted then and rules will be wrongly applied then.

Ex : $\{\neg(P \wedge Q) \Rightarrow R, \neg P \vee \neg Q\} \Rightarrow R$

The deduction steps of this problem should include
1)Application of DeMorgan's Law on $\neg P \vee \neg Q$ to obtain $\neg(P \wedge Q)$
2)Application of Modus Ponens on $\neg(P \wedge Q)$ and $\neg(P \wedge Q) \Rightarrow R$ to obtain the conclusion R.

However storing a mapping from integers to integers would skip the first step since $\neg P \vee \neg Q$ and $\neg(P \wedge Q)$ are equivalent. This would result in incomplete solutions,and hence we store mappings as strings.

6 Generating Problems based on use of specific deduction rules

Generating problems based upon specific deduction rules requires the user to specify the deduction rules he wants to be used in the problem that is generated. This interface of problem generation is important since the users that have prior knowledge of the rules would want to practice the problems of various difficulties using the same rules to master his knowledge over that rule, and users that have no or little knowledge of deduction rules would also want to acquire the concepts involved in the rule by practicing problems specific to the the rules. This part of the problem statement has not been dealt with in this internship and shall be done later .
One possible way of attacking the problem is tagging the problems as users use the systems to solve them and building a huge database and then using these tags to produce problems that the user asks for.
Another possible way of approaching this problem is the converse of finding the solution steps. In solution generation we converge to a conclusion starting from an initial set of premises using various possible steps. For producing a problem specific to deduction rule the starting point would be a formula that has been mapped from

some premises to itself under the considered rule. Now diverging from here we get a set of formulas that can serve as premises.
The depth or level of splitting could be used a measure of the difficulty level which would further enrich the system.

References

[1] Rohit Singh, Sumit Gulwani, Sriram Rajamani, *Automatically Generating Algebra Problems*, AAAI 2012

[2] Sumit Gulwani, *Synthesis from Examples*, WAMBSE 2012 (Keynote Paper)

[3] Patrick J. Hurley, *A Concise Introduction to Logic*, 7th ed: Ch 7, Wadsworth Publishing Co Inc (2005)