AUTOMATICALLY GENERATING PROBLEMS IN PROPOSITIONAL LOGIC

Ashudeep Singh, Pulkit Jain

Mentors: Dr. Amey Karkare, Dr. Subhajit Roy

Computer Science and Engineering, IIT Kanpur

Motivation

In an Intelligent Tutoring System, the feature of automatic problem generation is of utmost importance in subjects where generating fresh problems involving similar concepts and having similar difficulty levels, is a tedious job for the instructor. Also, there are students motivated to solve newer problems or practice problems of a particular difficulty. There are tutoring systems that provide problems from a finite set of problems. However, this approach doesn't provide sufficient personalization for the instructor and students. We desire to create three different interfaces on which problem generation can be done:

- Generating a problem similar to a given problem.
- Generating a problem from scratch.
- Generating a problem whose solution uses only specific axioms.

Introduction

In this project, we have made an attempt to automatically generate proof problems related to the course- "Introduction to Logic" in the topic of Propositional Logic. In particular, we generate proof problems, of the form $P_1, P_2, P_3, P_4 \ldots\ldots P_n \Rightarrow C$. Firstly, describing generating a problem similar to a given problem.

We describe our problem as $P_1, P_2, P_3, \ldots\ldots\ldots P_n \Rightarrow C$ where each P_i is a logical formula from variables $V_1, V_2, V_3, \ldots\ldots V_m$. We firstly convert our premises into truth tables and represent each variable, premise and the conclusion as a sequence of 0's and 1's which we store as a m-bit integer.

e.g. The problem: $P \Rightarrow Q$

$\neg Q$ // $\therefore \neg P$

is represented as follows:

P	Q	P⇒Q	¬Q	¬P
0	0	1	1	1
0	1	1	0	1
1	0	0	1	0
1	1	1	0	0

So, we store P, Q, P->Q, ¬Q, ¬P as integers corresponding to the above truth tables i.e. 3,5,13,10,12 respectively.

Two ways to change a given Problem:

1. Replace a Premise
2. Replace the Conclusion

Replacing a Premise

Constraints for a generated Problem:

1. $P_1 \wedge P_2 \wedge P_3 \wedge \ldots\ldots\ldots\ldots \wedge P_n \Longrightarrow C$.
2. $\wedge$ (any subset of P_js) $\nRightarrow C$.
3. $P_i \nRightarrow P_j$ and $P_j \nRightarrow P_i$ $\forall i, j, i \leq n, j \leq n$.

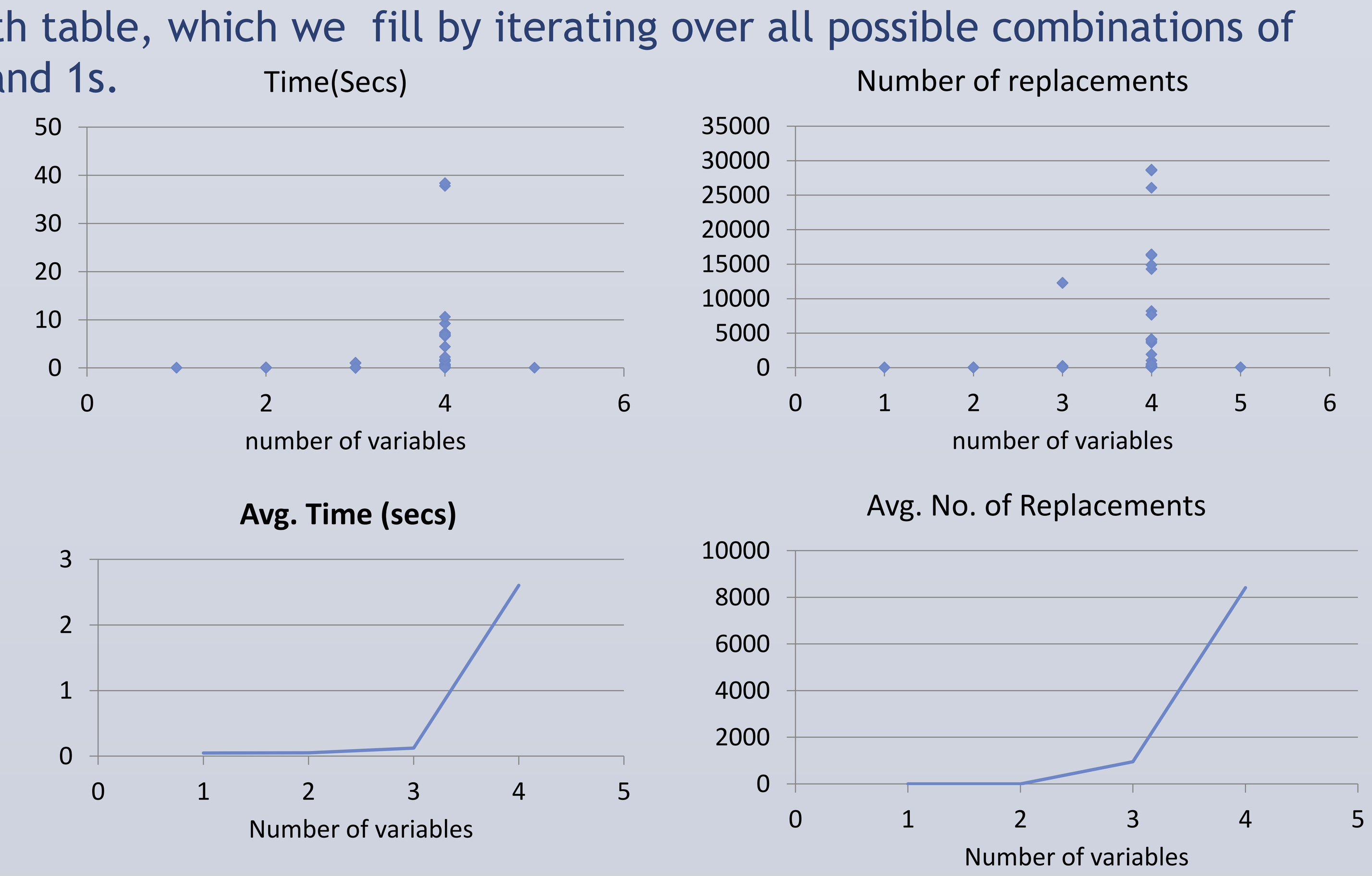
Now we apply the following constraints to the problem truth table above to find out the possible truth tables for the premise to be replaced.

CONSTRAINTS

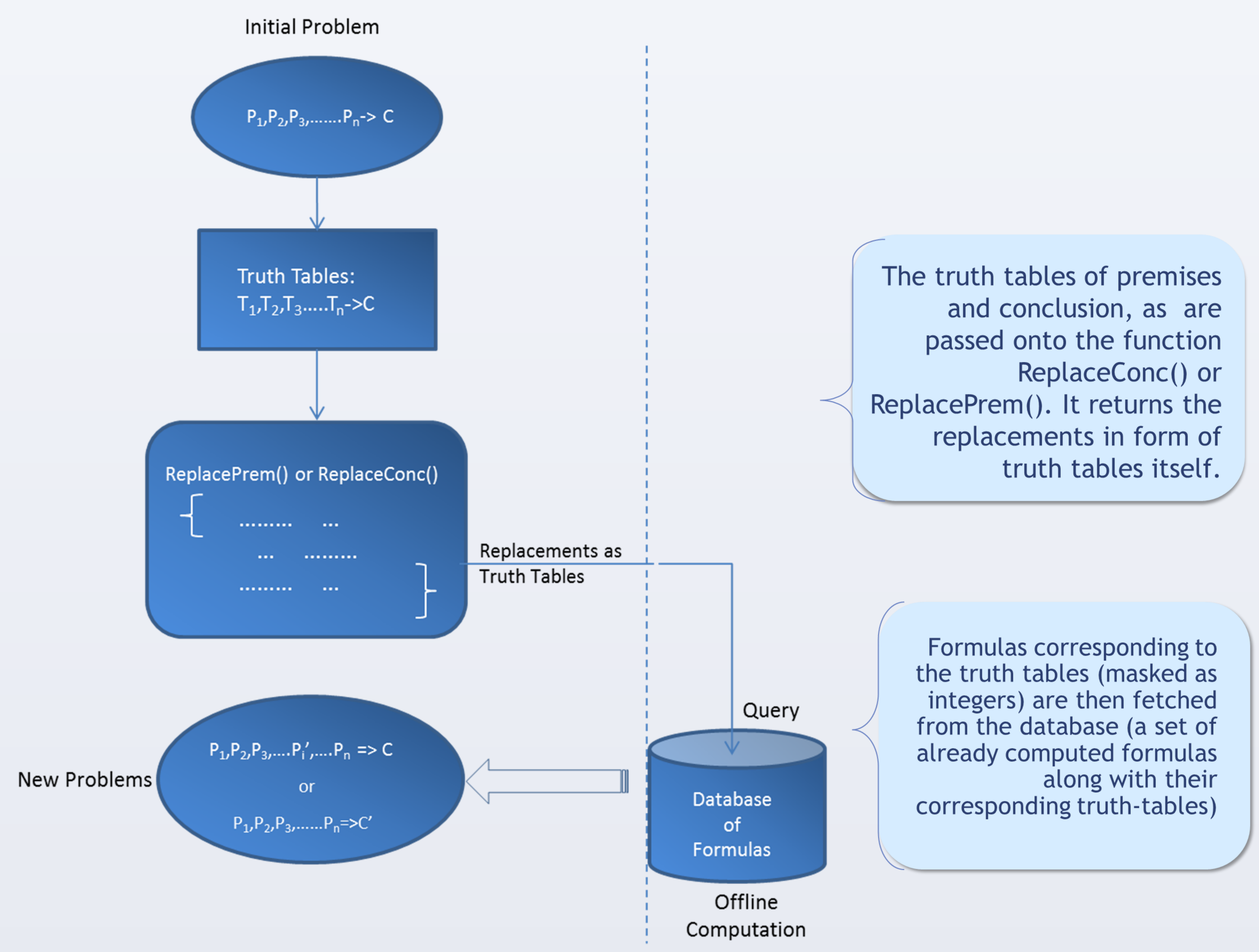
- **ZERO Constraint:** The rows in the truth table where $P_1 \wedge P_2 \wedge P_3 \wedge \ldots\ldots P_{i-1} \wedge P_{i+1} \ldots\ldots \wedge P_n$ is 1 and C is 0, put $P_i = 0$.
- **$\neg P_1$ Constraint:** The rows where $P_2 \wedge P_3 \wedge P_4 \wedge \ldots\ldots P_{i-1} \wedge P_{i+1} \ldots\ldots \wedge P_n$ is 1 and C=0, assign Pi' =1 for a subset of those positions. So, that $P_2 \wedge \ldots\ldots P_{i-1} \wedge P_{i'} \wedge P_{i+1} \ldots\ldots P_n \nRightarrow C$ as in Rule 2 above.
- **$\neg P_2$ Constraint, $\neg P_3$ Constraint and so on :** ....similar to ¬P1 Constraint.

After applying the above premise I have some blank spaces inside the truth table, which we fill by iterating over all possible combinations of 0s and 1s.

RESULTS



Approach



For generating problems from scratch, we first generate a random truth table which will represent C. Then we find another truth table P_1 such that $P_1 \nRightarrow C$. Then find n-2 premises such that $\forall (P_i, P_j)$ $P_i \nRightarrow P_j$ and $P_j \nRightarrow P_i$. Conjunction of any subset of P_ks $\nRightarrow C$. Then use the the function ReplacePrem() to find all possible P_ns.

Solution Generation

Formally, the solution Generation starts from two components:

- A problem <$P_1, P_2, P_3 \ldots\ldots\ldots\ldots P_n$,C> which includes the premises and the conclusion, that is to be derived from the premises.
- A set of mappings $\{<I_i, O_i, \Phi i(I_i, Oi)> \mid i=1,2,\ldots\ldots N\}$[1]. I_i is a tuple of logical formulas of size 1, 2 or 3 which is mapped to O_i under the rule Φ_i.

There are 18 rules (Φ_is) that map a tuple of logical formulas to a single formula O_i[3]. We need to find a function **solve**(I,O) that uses $\{O_1, O_2, O_3, \ldots\ldots O_n\}$ as temporary variables and returns the sequence of Φ_is which will lead us to the conclusion.

solve(I, O)	solve(I, O)	solve(I, O)
1. $O_1 := \Phi_2(I_1, I_2)$; 2. $O_2 := \Phi_1(O_1)$;	1. $O_1 := \Phi_1(I_1)$; 2. $O_2 := \Phi_2(O_1, I_2)$;	1. $O_1 := \Phi_1(I_2)$; 2. $O_2 := \Phi_2(I_1, O_2)$;

Fig. The possible sequence of Φ_is which can be obtained from two rules Φ_1(a unary rule) and Φ_2(a binary rule).

Then we apply brute-force method to find all the possible sequences of Φs that lead us to the conclusion.

Further Application of Solution Generation

The approach above provides us with multiple solutions of the same problem. These solutions can be used to provide hints in a problem solving UI.

References

- [1] Sumit Gulwani, Sushmit Jha, Ashish Tiwari, Ramarathnam Venkatesan. Synthesis of Loop-free Programs. PLDI 2011.
- [2] Rohit Singh, Sumit Gulwani, Sriram Rajamani. Automatically Generating Algebra Problems. AAAI 2012.
- [3] Patrick J.Hurley. A Concise Introduction to Logic. Wadsworth Publishing Company (2011-01-01)