Algorithms Anonymous

Sample Problem 1 on Propositional Logic

phoenixcoder

July 2019

1 Motivation

In every program, there is a set of requirements a developer must meet. This could be anywhere from validating a form to assuring a medical surgeon that you have thought through all the possible cases for a robotic surgical tool they are about to use. The range of things you can do with a program to model the real-life problems is vast. It owes its versatility to an overlap with the real world, namely, logic. If you can explain a thought, phenomenon, or pattern in a series of logical steps, you can create a program for it.

Let's say you had a simple form with the fields, name and e-mail, and a checkbox that confirms the user agrees to the terms and conditions of how you'll handle this information. Assuming you didn't have some fancy library to do the validation for you, how would you prove that all states, where the form is valid or invalid, were addressed?

2 Problem

Given the following table of boolean values, find a logical statement whose evaluation matches the value under the formValid column for every line.

nameFieldValid	**emailFieldValid**	**termsChecked**	**formValid**
T	T	T	T
T	T	F	T
T	F	T	T
F	T	T	T
T	F	F	T
F	F	F	F
F	T	T	F
F	F	F	F

3 Solution

The solution for the above is:

$$\begin{aligned}&\textbf{NOT}\ [(\textbf{NOT}\ nameFieldValid)\ \textbf{AND}\ (\textbf{NOT}\ emailFieldValid)\ \textbf{AND}\ termsChecked]\\ \textbf{AND NOT}\ &[(\textbf{NOT}\ nameFieldValid)\ \textbf{AND}\ emailFieldValid\ \textbf{AND}\ termsChecked\ ]\\ \textbf{AND NOT}\ &[(\textbf{NOT}\ nameFieldValid)\ \textbf{AND}\ (\textbf{NOT}\ emailFieldValid)\ \textbf{AND}\ (\textbf{NOT}\ termsChecked)]\end{aligned}$$

Its simplest logical equivalent is:

$$\textbf{IF}\ nameFieldValid\ \textbf{THEN}\ (emailFieldValid\ \textbf{AND}\ termsChecked)$$

In more human terms, the above says, don't bother to check the validity of the other two fields when nameFieldValue is False.

The more complicated of the two statements above is what we call, **Conjunctive-Normal Form (CNF)**. It's a form where you're restricted to using **AND**, **OR**, and **NOT** logical operators. There is a proof out there that says EVERY logical statement can be expressed as a CNF. This is why you can model any real life logical process with a program. However, the proof is beyond the scope of this document.

3.1 Solution Theory

3.1.1 Definitions

The table below is called a, **truth table**. It forms the foundation for logic. It itemizes every permutation of **truth values** each one of the statements could be in, for statements p and q. Truth values are just the boolean values, True and False. The **truth schedule** is a list of desired truth values for each one of the itemized states. The truth schedule can be self-defined based on the problem you're solving.

The truth table can be used to find a **truth function**, which is a **compound statement** composed of statements p and q along with logical connectives such as **AND**, **OR**, **NOT**, etc. The truth function must match the truth schedule exactly for every truth value input of p and q.

Ind	p	q	Truth Schedule
1	T	T	T
2	T	F	T
3	F	T	F
4	F	F	F

3.1.2 Procedure for Finding a Truth Function

The procedure to extract a **truth function** is:

1. Group the lines in the truth table together by False values of the truth schedule.
2. For a given line, we'll make a truth function that evaluates to True. For example, line 3 above will be:

$$\textbf{NOT}\ p\ \textbf{AND}\ q$$

3. For the truth function from the given line above, **NOT** the entire statement to make it evaluate to False. So, the statement above becomes:

$$\textbf{NOT}\ (\textbf{NOT}\ p\ \textbf{AND}\ q)$$

4. Repeat steps 2 and 3 for every False-valued line for the truth schedule.
5. At this point, you should have as many truth functions as there are False-valued line for the truth schedule. **AND** all of these truth functions together. For the above table, you should have the following truth function:

$$\textbf{NOT}\ (\textbf{NOT}\ p\ \textbf{AND}\ q)\ \textbf{AND NOT}\ (\textbf{NOT}\ p\ \textbf{AND NOT}\ q)$$

You may have seen the shortcut using your intuition, but the above simplifies to just:

$$p$$

So, a word of caution, intuition can lead you astray especially as the number of variables grow. Every time you add a variable the number of cases doubles. It doesn't take that long to get to a large number of cases. For instance, 6 variables will result in 64 total cases you'd have to check. The rules above will ensure you come to the intended logical conclusion no matter how many variables are involved.