

CS 30: Discrete Math in CS (Winter 2019): Lecture 12

Date: 24th January, 2019 (X hour)

Topic: Satisfiability of Formulas

Disclaimer: These notes have not gone through scrutiny and in all probability contain errors.

Please discuss in Piazza/email errors to deeparnab@dartmouth.edu

1. Tautologies, Contradictions, and Satisfiability.

A formula ϕ is a *tautology* if it takes the truth value true *no matter* what values the underlying variables take. That is, ϕ is logically equivalent to true. We have already see one tautology: $p \vee \neg p$ in the operation with negation.

Here is another example


$$\phi := p \wedge (p \Rightarrow q) \Rightarrow q$$

One way to check this the truth table.

p	q	$p \Rightarrow q$	$p \wedge (p \Rightarrow q)$	$p \wedge (p \Rightarrow q) \Rightarrow q$
true	true	true	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	true

Another way is to reduce the formula as follows to prove that the formula is equivalent to true.

$$\begin{aligned} p \wedge (p \Rightarrow q) \Rightarrow q &\equiv p \wedge (\neg p \vee q) \Rightarrow q && \text{Implication as OR} \\ &\equiv ((p \wedge \neg p) \vee (p \wedge q)) \Rightarrow q && \text{Distributivity} \\ &\equiv (\text{false} \vee (p \wedge q)) \Rightarrow q && \text{Operation with Negation} \\ &\equiv (p \wedge q) \Rightarrow q && \text{Operation with false} \\ &\equiv \text{true} && \text{Drill} \end{aligned}$$

A formula ϕ is a *contradiction* or *unsatisfiable* if it takes the truth value false *no matter* what values the underlying variables take. That is, ϕ is logically equivalent to false. 

Exercise: Prove that the following formula is a contradiction

$$\left((\neg p \wedge q) \vee (p \wedge \neg q) \right) \wedge (p \Rightarrow q) \wedge (q \Rightarrow p)$$

A formula ϕ is *satisfiable* if there is some setting of the underlying variables which makes it true. That is, it is *not* unsatisfiable.

2. Toy Application: 2-Queens Problem.

The above may seem quite abstract, so let us give a toy but concrete application : the 2-queens problem. We are given a 2×2 chess-board (i.e. a grid), and the question is whether we can place 2 queens on this board so that they aren't attacking each other.

Of course, the answer is no. What is the proof? Furthermore, how can you get a machine convinced of the proof (or alternately, how can you get the machine to prove it?). This is where logic is key. We are now going to cast this problem as asking whether a certain formula is satisfiable.

To do so, we need to define variables. We have four variables (atomic propositions) a, b, c, d where a takes the truth value true if a queen is placed in the north-west corner and is false otherwise. Similarly, b is true if and only if a queen is placed in the north-east corner, c is true if and only if a queen is placed in the south-west corner, and d is true if and only if a queen is placed in the south-east corner.

How do we encode that the queens should not attack another? If we place a queen at the north-west corner, then it rules out a queen in any other corner. We can write this as

$$a \Rightarrow (\neg b \wedge \neg c \wedge \neg d)$$

If a is true, then b, c, d must be set to false. Similarly we have three other compound propositions, all of which must be satisfied. So the formula till now is

$$\begin{aligned}\phi_1 &= a \Rightarrow (\neg b \wedge \neg c \wedge \neg d) \\ &\wedge b \Rightarrow (\neg a \wedge \neg c \wedge \neg d) \\ &\wedge c \Rightarrow (\neg a \wedge \neg b \wedge \neg d) \\ &\wedge d \Rightarrow (\neg a \wedge \neg b \wedge \neg c)\end{aligned}$$

Next we need to encode there must be at least two of the squares with queens. We can encode this by going over all possible pair of locations and putting a queen in both of those; at least one of these pairs must be true. That is, let

$$\phi_2 := (a \wedge b) \vee (a \wedge c) \vee (a \wedge d) \vee (b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$$

The final formula is $\phi := \phi_1 \wedge \phi_2$.

Theorem 1. The 2-queens problem has a solution if and only if ϕ is a satisfiable formula.

Proof. Suppose the 2-queens problem is solvable. Depending upon the locations of where the 2 queens are placed, we set the corresponding variable to be true and the rest to be false. For example, if the queens are placed in the north-west and south-east corners, then we set a and d to true and b and c to false. If the queens are not attacking, then this setting sets ϕ_1 to be true. Since at least (actually exactly) two of the variables are set to true, we get ϕ_2 is set to true.

Similarly, given a setting of variables for a, b, c, d which leads to ϕ being true, we put a queen in every position for which the variables are set to true. The formula ϕ_1 being true implies the positions are non-attacking. The formula ϕ_2 being true implies we put at least two queens. □

Exercise: Can you prove, without writing the truth table, that $\phi = \phi_1 \wedge \phi_2$ written above is indeed unsatisfiable?

This may seem like using a cannon to kill a mosquito, but note that the above easily generalizes for n -queens for any n . Sure, the number of variables become n^2 instead of 4, and the “size” of ϕ also grows (so it won’t be pretty to write on paper), but we can write code to make a computer understand the formula. Then, if the computer had a fast way to figure out if ϕ was satisfiable, it could solve the n -queens problem.

3. A Million Dollar Question.

Given a formula, can we decide whether it is satisfiable? Sure. We can write the truth table and check if any of the values evaluate to true. However, if there are n variables this may take a LOT of time (filling up the truth table only for the variables will take 2^n entries to fill). The *efficient version* of the decidability question is the following: is there a method to decide whether a formula on at most n variables and n clauses which runs in time no more than a fixed *polynomial* of n ? This is a version of the famous P vs NP question. A detailed discussion is beyond the scope of this course, but CS 39 goes in deep detail.