CS 30: Discrete Math in CS (Winter 2020): Lecture 3supp

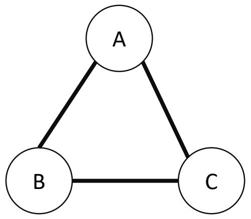
Date: 9th January, 2020 (X-hour)

Topic: An application

Disclaimer: These notes have not gone through scrutiny and in all probability contain errors.
Please discuss in Piazza/email errors to deeparnab@dartmouth.edu

---

In this supplement, we show a simple, toy example how logic can be used to *formally* prove theorems. Imagine a situation as shown in the following figure: In our toy puzzle, we need to put



in *pegs* into these circles. The rule is that on any two adjacent circles joined by lines, we are not allowed to put pegs on both. What is the maximum number of pegs we can put in?

If you have understood what the question is asking, then you immediately know the answer: it is **one**. We can put one peg. And we cannot put in two pegs. But pause for a minute and ask yourself: *why?* What is the *proof* that we cannot put two pegs in? Indeed one can make an argument in English – all pairs are adjacent, and therefore of you put two pegs, the pair of pegs you put on are adjacent and thus violates the rule. But, and this is important, English is just a medium for us humans to talk and understand; we share a common context which allows us to comprehend and be convinced of this reasoning. But how do we convince a computer? How does a computer come up with a proof? This is where logic will help us.

Remark: *Before I embark upon this, I want to impress another fact on you. Note what we are trying to show – we are trying to show no matter how you put two pegs, we will violate some rule. This ruling out of* all *possible answers is actually a highly non-trivial problem in general. Not here, because there are only 3 circles and 2 pegs. But in general it is. More on this later.*

The main idea is to *convert* the peg-placing problem into a question of satisfiability/unsatisfiability of some Boolean formula. To this end, let is introduce some *Boolean variables*. Namely,

$$\begin{aligned} a &: \text{This is true if a peg is placed in circle } A. \\ b &: \text{This is true if a peg is placed in circle } B. \\ c &: \text{This is true if a peg is placed in circle } C. \end{aligned}$$

Note that any placement of pegs assigns a truth value to the three variables, and contrapositively, every truth value assignment can be read off as a peg placement in the circle. Convince yourself of this fact before moving on.

Now, we introduce formulae (the jargon often used is *clauses*) for each rule we have. For instance, we have the rule that A gets a peg, then neither B nor C gets a peg. As a formula this is encoded as

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

If there is a peg placement that satisfies the rule, then the above formula (clause) evaluates to true. There are two more rules (for circles B and circles C). Namely,

$$b \Rightarrow (\neg c \wedge \neg a) \quad \text{and} \quad c \Rightarrow (\neg a \wedge \neg b)$$

A valid peg placement is one that satisfies all these rules. Therefore, a valid peg placement must assign truth values which satisfy the following formula

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

Indeed, a, b, c all set to false is a satisfying assignment to the above formula (the formula evaluates to true); indeed, putting no pegs anywhere satisfies all conditions.

With me so far? Good. Now let us come to the number. How do we encode the fact that we are putting in (at least) 2 pegs? Well two of a, b or c must be set to true. This is equivalent to saying the following formula must evaluate to true.

$$((a \wedge b) \vee (a \wedge c) \vee (b \wedge c))$$

Therefore, if there is a peg placement of (at least) 2 pegs which satisfies all the rules, then we must have a satisfying formula to the following composite formula (the AND of the above two formulae):

$$\phi := \Big(a \Rightarrow (\neg b \wedge \neg c)\Big) \wedge \Big(b \Rightarrow (\neg c \wedge \neg a)\Big) \wedge \Big(c \Rightarrow (\neg a \wedge \neg b)\Big) \quad \wedge \quad ((a \wedge b) \vee (a \wedge c) \vee (b \wedge c))$$

Furthermore, any satisfying formula of ϕ can lead to a valid peg placement of (at least) 2 pegs.

Therefore, to prove that one cannot put 2 pegs in the above toy game, one only needs to show

$$\phi \quad \text{is unsatisfiable.}$$

✍

Exercise: *Using the logical equivalences done in class, show indeed that ϕ is unsatisfiable.*

Great. But what's the big deal of solving this trivial toy problem in such a roundabout way. The point is that this roundabout way of creating ϕ is *absolutely* mechanical. Instead of a triangle, suppose I had a square and the rules were the same. How many pegs can you put in then? How about a pentagon? Or some other graph? While as humans we would have to come up with possibly different methods of arguing, the above method can be mechanized to produce a formula for *any* figure. Furthermore, one can then use the logical equivalences (which also is pretty mechanical), to prove the formula is unsatisfiable. Or one can use some existing code out there (yes, believe it ot not, it's a pretty huge business – checking if formula are satisfiable or not.) The point is, that logic can allow *mechanical* ways of *proving* theorems. And indeed, there are some theorems out there for which this[1] approach is the *only* way we know how to prove them. ✍

Exercise: *To see if you have understood the above method, try to mimic the same thing for the square.*

[1]By this, I don't mean only using just the operations we did, but similar approach