Lógica en la Informática / Logic in Computer Science

Monday Nov 5th, 2018

Time: 1h30min. No books, lecture notes or formula sheets allowed.

1) (2 points)

1a) Is it true that for any two propositional formulas F and G, we have that $\neg F \vee G$ is a tautology if and only if $F \models G$? Prove it using only the definition of propositional logic.

Answer: Yes.

$\neg F \vee G$ is a tautology iff	by definition of tautology
forall I, $\quad I \models \neg F \vee G$ iff	by definition of $\models$
forall I, $\quad eval_I(\neg F \vee G) = 1$ iff	by definition of evaluation of $\vee$
forall I, $\quad max(eval_I(\neg F), eval_I(G)) = 1$ iff	by definition of evaluation of $\neg$
forall I, $\quad max(1 - eval_I(F), eval_I(G)) = 1$ iff	by definition of max and eval
forall I, $\quad eval_I(F) = 0$ or $eval_I(G) = 1$ iff	by definition of satisfaction
forall I, $\quad I \not\models F$ or $I \models G$ iff	by definition of logical consequence
$F \models G$	

1b) Is it true that for any three propositional formulas F, G, H, we have that $F \wedge (G \vee H)$ is a tautology iff $(\neg G \wedge H) \vee \neg F$ is unsatisfiable? Prove it using only the definition of propositional logic.

Answer: No. Counter example:

Let F be $p \vee \neg p$, let G be p and let H be $\neg p$. Then $F \wedge (G \vee H)$ is the formula $(p \vee \neg p) \wedge (p \vee \neg p)$ which is a tautology. But $(\neg G \wedge H) \vee \neg F$ is the formula $(\neg p \wedge \neg p) \vee \neg(p \vee \neg p)$, which is satisfiable, because any I with $I(p) = 0$ is a model.

2) (2 points)

Our friend John has invented something he calls a "propositional database", which consists of a set of positive unit clauses (propositional symbols) and of a set of "rules" of the form:
$Condition \rightarrow Consequence$, where $Condition$ and $Consequence$ are sets of propositional symbols.

I asked John what he considers to be true facts, or simply "facts", in his database, and he said: "the minimal set of facts such that all positive unit clauses are facts, and, for every rule $Condition \rightarrow Consequence$, if all elements of $Condition$ are facts, then also all elements of $Consequence$ are facts".

2a) Given such a database D, we want to know if a certain symbol p is a fact in D. Explain very briefly. What is the cost of deciding this? How?

2b) I want to know how many facts are true in D. Explain very briefly. What is the cost of counting this? How?

Answer for 2a and 2b: The definition of the set of true facts is precisely the way Horn SAT is decided by positive unit propagation: all positive unit clauses are facts, and each rule $\{p_1, \ldots, p_n\} \rightarrow \{q_1, \ldots, q_m\}$ has exactly the same effect as the propagation of the Horn clauses

$\neg p_1 \vee \ldots \vee \neg p_n \vee q_1 \quad \ldots \quad \neg p_1 \vee \ldots \vee \neg p_n \vee q_m$.

So after the positive unit propagation finishes, we can check whether a given p is a fact, and we can count the number of facts, all of which can be done in linear time.

3) (4 points) MaxSAT is a problem related to SAT. It takes as input a natural number k and a set S of n propositional clauses over propositional symbols $\mathcal{P}$, and it asks whether there is any interpretation $I : \mathcal{P} \to \{0, 1\}$ that satisfies at least k clauses of S.

3a) Do you think that MaxSAT is polynomial? NP-complete? Exponential? Why?

Answer: It is NP-complete. For this, we need to show two things:
A) MaxSAT is NP-hard (not easier than general SAT for cnfs), since we can polynomially reduce SAT (for cnfs) to MaxSat: SAT is the particular case of MaxSAT where $k = n$.
B) MaxSat is in NP (not harder than SAT) since one can polynomially reduce MaxSat to SAT as shown in the answer to 3b. Another independent reason (without using 3b): MaxSat is in NP because one can verify a given solution, an interpretation I, in polynomial (in fact, linear) time, checking whether indeed I satisfies at least k clauses.

3b) Is it true that, using auxiliary variables, one can decide MaxSAT in a single call to a SAT solver? Explain why.

Answer: Yes, a single call to a SAT solver is sufficient.
Let S be $\{C_1, \ldots, C_n\}$. Use new auxiliary variables $a_1, \ldots, a_n$. Let S' be $\{C_1 \vee a_1, \ldots, C_n \vee a_n\}$. Then we need to find a model I for S' such that at least k of the auxiliary variables are false. Let $Card$ be the set of clauses obtained by encoding the cardinality constraint $\neg a_1 + \ldots + \neg a_n \geq k$. Note that using, e.g., sorting networks, $|Card|$ is polynomial ($O(n\ log^2 n)$). Then, running the solver with input clauses $S' \cup Card$ will find the desired I if it exists, and return "unsat" otherwise.

3c) How would you use a SAT solver to solve the optimization version of MaxSAT, that is, how to find the I that satisfies as many of the clauses of S as possible? Give one single (and short) explanation.

Answer: For this we need more than one call to the SAT solver.
A) Run the solver on the input S' defined as in 3b). Note that S' is satisfiable (just set all a_i's to 1).
B) If it finds a model where m a_i's are false, run again with input $S' \cup Card$, where $Card$ is the set of clauses obtained by encoding the cardinality constraint $\neg a_1 + \ldots + \neg a_n > m$. Repeat step B (each time finding models with more false $a_i's$), until the solver returns unsat. The last solution found is the optimal one.

[Another algorithm is to make calls to the solver with S' and a constraint $\neg a_1 + \ldots + \neg a_n \geq k$, where at the first call k is n; if it is unsatisfiable, then try $n - 1$, etc., until a model is found, which is then optimal. Yet another algorithm is to do a binary search. But the first algorithm given here works very well, because it is usually easier to find a model than to prove unsatisfiability and because the improvement in solution quality at each iteration frequently is by more than 1.]

3d) We want to 3-color a given graph with n nodes and m edges: assign one of the 3 colors to each node such that for no edge (u, v) nodes u and v get the same color. Of course this may be impossible, so we will allow some nodes to remain uncolored: they get *no* color. How can we use the ideas of 3b,c) to compute the 3-coloring with the *minimal* number of such uncolored nodes?

Answer: As usual in graph coloring, define variables $x_{i,c}$ meaning "node i gets color c" and add clauses $\neg x_{u,c} \vee \neg x_{v,c}$ for each edge (u, v) and color c. For each node i, to express that i gets at least one color (here, of the 3 colors), we would have a clause $x_{i,1} \vee x_{i,2} \vee x_{i,3}$, but to these clauses we add the auxiliary variable a_i, getting $x_{i,1} \vee x_{i,2} \vee x_{i,3} \vee a_i$. To express that at least k nodes get a color, add the cardinality constraint $\neg a_1 + \ldots + \neg a_n \geq k$, and then iteratively find the maximal k as in 3c).

4) (2 points) Is 3-SAT NP-complete? Explain your answer very briefly, using the fact that SAT (deciding the satisfiability of an arbitrary propositional formula F) is NP-complete.

Answer: 3-SAT is the problem of deciding the satisfiability of a set of clauses S with at most 3 literals per clause. Yes, it is NP-complete. For this, we need to show two things:
A) 3-SAT is NP-hard (not easier than general SAT) since we can polynomially reduce SAT to 3-SAT: the Tseitin transformation in polynomial time reduces SAT for an arbitrary F to a polynomial-sized 3-SAT set of clauses S that is equisatisfiable to F.
B) 3-SAT is in NP (not harder than SAT) since one can polynomially reduce 3-SAT to SAT: this is trivial because 3-SAT *is* already a particular case of SAT (since S is already a propositional formula).