Lógica en la Informática / Logic in Computer Science

Thursday April 18th, 2013

Time: 1h45min. No books, lecture notes or formula sheets allowed.

1A) Is it true that if F is unsatisfiable then $\neg F$ is a tautology? Prove it using only the formal definitions of propositional logic.

Answer: F unsatisfiable iff, by definition, $\forall I,\ eval_I(F) = 0$, which implies $\forall I,\ 1 - eval_I(F) = 1$, which, by definition of evaluation of $\neg F$, implies $\forall I,\ eval_I(\neg F) = 1$, which is the same as $\forall I,\ I \models \neg F$ and $\neg F$ tautology.

1B) Is it true that if F, G, H are formulas such that $F \wedge G \not\models H$ then $F \wedge G \wedge H$ is unsatisfiable? Prove it using only the formal definitions of propositional logic.

Answer: This is false. Counterexample: $F = p$, $G = p$, $H = q$. Then $p \wedge p \not\models q$, but $p \wedge p \wedge q$ is satisfiable.

2) Using the Tseitin transformation, we can transform an arbitrary propositional formula F into a set of clauses $T(F)$ (a CNF with auxiliary variables) that is *equisatisfiable*: F is SAT iff $T(F)$ is SAT. Moreover, the size of $T(F)$ is linear in the size of F.
2A) Is there any transformation T' into an equisatisfiable linear-size DNF? If yes, which one? If not, why?

Answer: No (unless $P = NP$). If such a similar transformation existed, then we could solve an NP-complete problem (is F SAT?) by transforming F in linear time into the DNF $T'(F)$, and then deciding whether the DNF $T'(F)$ is satisfiable (which, as we know, can be done in linear time for DNFs).

2B) Is there any similar transformation T' into a linear-size DNF, such that F is a tautology iff $T'(F)$ is a tautology? If yes, which one? If not, why?

Answer: Yes. F is a tautology iff $\neg F$ is unsatisfiable iff the normal Tseitin transformation $T(\neg F)$ is unsatisfiable iff $\neg T(\neg F)$ is a tautology. And indeed $\neg T(\neg F)$ can be easily transformed into a DNF: $T(\neg F)$ is a conjunction of claues $C_1 \wedge \ldots \wedge C_n$. Its negation $\neg(C_1 \wedge \ldots \wedge C_n)$ is equivalent to $\neg C_1 \vee \ldots \vee \neg C_n$, and each $\neg C_i$ is of the form $\neg(l_1 \vee \ldots \vee l_m)$ which is equivalent to $\neg l_1 \wedge \ldots \wedge \neg l_m$. Note that, unlike what happened in the previous case, here we transform an NP-complete problem into another NP-complete problem.

3) A pseudo-Boolean constraint has the form $a_1x_1 + \ldots + a_nx_n \leq k$ (or the same with $\geq$), where the coefficients a_i and the k are natural numbers and the x_i are propositional variables. Which clauses are needed to encode the pseudo-Boolean constraint $2x + 3y + 5z + 6u + 8v \leq 11$ into SAT, if no auxiliary variables are used? Which clauses are needed in general, with no auxiliary variables, for a constraint $a_1x_1 + \ldots + a_nx_n \leq k$?

Answer: To encode $2x + 3y + 5z + 6u + 8v \leq 11$, for every (minimal) subset of variables such that the sum of its coefficients is more than 11, we forbid that all of them are true. In this case, it suffices to have five clauses: $\neg v \vee \neg u$, $\neg v \vee \neg z$, $\neg v \vee \neg x \vee \neg y$, $\neg u \vee \neg z \vee \neg y$ and $\neg u \vee \neg z \vee \neg x$.

Note that "minimal" here means that, for example, the clause $\neg v \vee \neg z \vee \neg y$ is not needed because it is subsumed by the stronger clause $\neg v \vee \neg z$.

In general, given a constraint $a_1x_1 + \ldots + a_nx_n \leq k$, we need one clause $\neg x_{i_1} \vee \ldots \vee \neg x_{i_k}$ for each subset $S = \{i_1 \ldots i_k\}$ of $\{1 \ldots n\}$ such that $a_{i_1} + \cdots + a_{i_k} > k$, and such that moreover S is minimal ($a_{i_1} + \cdots + a_{i_k} - a_{i_j} \leq k$ for every j with $1 \leq j \leq k$).

4) For organizing the general elections in Ecuador, we need to decide where to locate the polling places (the places where people can vote). To do this, we have a long list $L = \{1 \ldots N\}$ of possible places (schools, town halls, etc), and for each inhabitant i of the m inhabitants of Ecuador ($i \in \{1 \dots m\}$), a sublist L_i of L with those places that are close enough to i's home. To save costs, we would like to only open K polling places, but of course guaranteeing that every inhabitant i can vote at one of the places on its list L_i. Which (and how many) variables and clauses do we need to solve this problem using SAT?

Answer: We introduce variables x_i meaning "polling place i is opened", for all i with $1 \leq i \leq N$. We need to express that for each inhabitant i, at least one of the polling places of its list $L_i = \{i_1 \dots i_p\}$ is opened. This we can express with one clause of length $|L_i| = p$ of the form $x_{i_1} \vee \dots \vee x_{i_p}$ (one clause per inhabitant: m clauses). We also need to express that at most K polling places are opened: this we do by encoding into SAT an at-most-K constraint $x_1 + \cdots + x_N \leq K$, which we can express using any of the standard methods for these constraints, for example, using sorting networks ($O(Nlog^2N)$ clauses) or cardinality networks ($O(Nlog^2K)$ clauses) or with no auxiliary variables ($\binom{N}{K+1}$ clauses).

5) We have a computing facility with N identical computers, and we need to handle T computing tasks, with $T > N$. Each task $i \in \{1 \dots T\}$ has a duration of d_i seconds. Each computer can handle no more than one task at the same time. We want to determine whether it is possible to distribute the tasks over these computers in such a way that after less than K seconds all tasks are finished. Which (and how many) variables and clauses do we need to solve this problem using SAT? Note: if needed, you can leave arithmetical constraints (at-most-one, cardinality, pseudo-Boolean...) without encoding them into SAT.

Answer: We introduce variables $x_{i,j}$ meaning "task i will be run at computer j", for all i with $1 \leq i \leq T$ and for all j with $1 \leq j \leq N$ (total: $N \cdot T$ variables).

We need to express that each task i, will be run at at least one computer. This we do with one N-literal clause per task i, of the form $x_{i,1} \vee \dots \vee x_{i,N}$ (that is, T clauses). We also need to express that the tasks scheduled at each computer have a total duration of no more than k. This we can do with one pseudo-Boolean constraint per computer j of the form $d_1 \cdot x_{1,j} + \cdots + d_T \cdot x_{T,j} \leq k$ (N pseudo-Boolean constraints).

Note that a solution does not tell us in which order the tasks are handled (any order is ok), and also note that it is not really necessary to express that each task is at at most one computer: in any solution where some task is scheduled at more than one computer we can simply choose one of these computers.