Lógica en la Informática / Logic in Computer Science

Thursday January 12th, 2017

Time: 2h30min. No books, lecture notes or formula sheets allowed.

Note on evaluation:
eval(propositional logic) = max{ eval(Problems 1,2,3,4), eval(partial exam) }.
eval(first-order logic) = eval(Problems 5,6,7).

1) Let F and G be arbitrary propositional formulas.

1a) Can it happen that $F \models G$ and $F \models \neg G$? Prove it from the definitions of propositional logic.

Answer: Yes, if F is unsatisfiable (e.g., if F is $p \wedge \neg p$); then *any* formula is a logical consequence of F.

1b) Assume $F \models G$. Prove, from the definitions of propositional logic, that then $F \equiv F \wedge G$.
Hint: prove $F \equiv F \wedge G$ distinguishing the two cases: $I \models F$ and $I \not\models F$ (and use $F \models G$ in one case).

Answer: To prove $F \equiv F \wedge G$, (by definition of $\equiv$) we have to prove that F and $F \wedge G$ have the same models, that is, for every interpretation I, $eval_I(F) = eval_I(F \wedge G)$. We distinguish two cases:
Case $eval_I(F) = 1$. This (since $F \models G$) implies
$eval_I(F) = eval_I(G) = 1$ which (by definition of *min*) implies
$min(eval_I(F), eval_I(G)) = 1$ which (by definition of $eval_I(\wedge)$) implies $eval_I(F \wedge G) = 1$.
Case $eval_I(F) = 0$. This (by definition of *min*) implies
$min(eval_I(F), eval_I(G)) = 0$ which (by definition of $eval_I(\wedge)$) implies $eval_I(F \wedge G) = 0$.

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

Answer: No such a transformation T' is known, because then it would also be known that $P = NP$: 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).

3) Answer **very briefly:** What is 2-SAT? Is it polynomial? Why?

Answer: 2-SAT is **the problem of** deciding the satisfiability of a given set of clauses S where each clause in S has at most two literals. It is polynomial. One method is by checking if the empty clause is in the closure under resolution of S, which is quadratic since only clauses with at most two literals are generated, and only a quadratic number of such clauses exist.

4) Answer **very briefly:** 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 $1 \leq j \leq k$).

5) Is the following first-order formula satisfiable? If not, explain why. If yes, give a model (and no explanations).

$\forall x \ \neg p(x,x) \ \wedge$
$\forall x \, \forall y \, \forall z \ (\ p(x,y) \wedge p(y,z) \rightarrow p(x,z) \ ) \ \wedge$
$\forall x \, \exists y \ p(x,y) \ \wedge$
$\forall x \, \forall y \ (\ p(x,y) \rightarrow \exists z \, (p(x,z) \wedge p(z,y)) \ )$

Answer: Yes. Example of model I: $D_I = \mathbb{Q}$ and $p_I(x,y) = 1$ iff $x < y$ (x is a smaller rational number than y). Note: there is no finite model; the first three conjuncts (p is an irreflexive, transitive and serial relation), are the typical example of this.

6)

6a) Explain in a few words how you would formally prove, given two first-order formulas F and G, that $F \not\models G$.
6b) Same question for $F \models G$.
6c) F is $\forall x \, p(a,x) \wedge \exists y \, \neg q(y)$ and G is $\exists v \exists w \, (\neg q(w) \wedge p(v,a))$. Do we have $F \models G$? Prove it.
6d) F is $\forall x \exists y \, p(x,y)$ and G is $\exists y \forall x \, p(x,y)$. Do we have $F \models G$? Prove it.

Answer:

6a: Giving a counter example, an interpretation I such that $I \models F$ but $I \not\models G$.

6b: By proving that $F \wedge \neg G$ is unsatisfiable, turning it into clausal form S, and obtaining the empty clause from S by resolution and factoring.

6c: Yes. $F \models G$. We prove it as in 6b. Here F gives two clauses: $1: \ p(a,x)$ and $2: \ \neg q(c_y)$ (here c_y is the Skolem constant introduced for y).
$\neg G$ is $\neg(\exists v \exists w \, (\neg q(w) \wedge p(v,a)))$ which becomes $\forall v \forall w \, \neg(\neg q(w) \wedge p(v,a))$ which becomes
$\forall v \forall w \, (q(w) \vee \neg p(v,a))$ which becomes the clause $3: \ q(w) \vee \neg p(v,a)$.
By one step of resolution between 1 and 3 with mgu $\{x = a, v = a\}$ we get clause $4: q(w)$, and with one more step of resolution between 2 and 4 with mgu $\{w = c_y\}$ we get the empty clause.

6d: No. $F \not\models G$. We prove it as in 6a. Consider the interpretation I where $D_I = \{a, b\}$ and p_I is interpreted as equality on this domain. Then $I \models F$ but $I \not\models G$.

7) Write a program `return(L,A)` in swi prolog (using `library(clpfd)` or not, feel free) that outputs (writes) the *minimal* number of coins needed for returning the amount `A` if (infinitely many) coins of each value of the list `L` are available. Two examples:

`?-return([1,5,6],10).` writes `2` (since 2*5 = 10).
`?-return([1,2,5,13,17,35,157],361).` writes `5` (since 1*13 + 2*17 + 2*157 = 361).

Answer: Note: greedy algorithms (starting with the largest coin, etc.) do not work (try the first example!).

```
return(L,A):- between(0,A,N), coins(N,L,A), write(N), nl.
coins(0,[],0).
coins(N,[C|Cs],A):- N>0,A>0,  between(0,N,K),  N1 is N-K, A1 is A-K*C,  coins(N1,Cs,A1).
```

Another answer:

```
:-use_module(library(clpfd)).
return(L,A):- length(L,N), length(V,N), V ins 0..A,  sumExpression(V,SumExpr),
              totalValueExp(V,L,Expr),  Expr #= A,   labeling( [min(SumExpr)], V ),
              S is SumExpr, write(S), nl,!.

sumExpression( [],     0      ):- !.
sumExpression( [V|Vs], V+Expr ):- sumExpression(Vs,Expr),!.

totalValueExp( [],     [],     0          ):- !.
totalValueExp( [V|Vs], [C|Cs], V*C + Expr ):- totalValueExp(Vs,Cs,Expr),!.
```