Lógica en la Informática / Logic in Computer Science

Monday June 13, 2016

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

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

1a) Let F and G be propositional formulas such that F is a tautology. Is it true that $F \land G \equiv G$? Prove it using only the definitions of propositional logic.

`Answer:` By definition of $\equiv$, we have to prove that $\forall I\ eval_I(F \land G) = eval_I(G)$.
Let I be an interpreation. Then:
$eval_I(F \land G) =$ by definition of $eval_I$ of a $\land$
$min(eval_I(F), eval_I(G)) =$ since F is tautology
$min(1, eval_I(G)) =$ by def. of min and since $eval_I(G)$ is either 0 or 1
$eval_I(G)$.

1b) Let F and G be propositional formulas such that F is satisfiable and $F \to G$ is also satisfiable. Is it true that G is satisfiable? Prove it using only the definitions of propositional logic.

`Answer:` This is false. Counter example: let F be p and let G be $p \land \neg p$. Then F is satisfiable with the model I such that $I(p) = 1$. And $F \to G$ is also satisfiable, with the model I such that $I(p) = 0$. But $p \land \neg p$ is unsatisfiable.

2) Let us remember the well-known graph coloring problem. **Input:** a natural number k, and an (undirected) graph with n vertices and m edges of the form $(u_1, v_1) \ldots (u_m, v_m)$, with all u_i and v_i in $\{1 \ldots n\}$, and **Question:** is there a way to "color" each vertex with a color (a number) in $1 \ldots k$ such that adyacent vertices get different colors?

We know that graph coloring is NP-complete in general. But what is its complexity if $k = 2$? Explain why using sat-based arguments.

`Answer:` One can express a graph coloring problem (for any k) as a SAT problem with variables $x_{i,j}$ meaning "vertex i gets color j". We need one clause $x_{i,1} \lor \ldots \lor x_{i,k}$ for each vertex i (it gets at least one color). We also need a two-literal clause $\neg x_{i,k} \lor \neg x_{j,k}$ for each edge (i, j) and color k (i and j do not both get color k).

If $k = 2$ this is a 2-SAT problem, which can in fact be solved in linear time.

3) Let S be a satisfiable set of propositional Horn clauses.
3a) What is the complexity of finding the *minimal* model of S, that is, the model I with the minimal number of symbols p such that $I(p) = 1$?
3b) What is the complexity of deciding whether S has only one model or more than one?
For both questions, explain very, very, briefly why.

`Answer:`
3a) Horn SAT can be decided by unit propagation of positive unit literals (see problem 3 of the April 2016 exam for details and examples). Once the unit propagation finishes, a model I is obtained, in linear time, by setting the propagated positive units to 1 and all other variables to 0 ($I(p) = 1$ iff p is a propagated positive unit). This model I is minimal, since each positive unit p that gets propagated is a logical consequence of S and hence *must* be true in *all* models of S.
3b) Any other model must *extend* the unique minimal model I with at least one more true symbol. It suffices to do the following after the propagation of case 3a: pick one q such that $I(q) = 0$, and propagate q. Another (non-minimal) model exists iff for some such a picked q this does not generate the empty clause. Therefore this problem is polynomial as well, since at most $|\mathcal{P}|$ more unit propagations have to be tried.

4) We want to write a computer program that takes as input two arbitrary first-order formulas F and G and always terminates writing "yes" if $F \equiv G$, and "no" otherwise. Explain very shortly the steps you would follow to do this, or to get something as similar as possible.

`Answer:` No such program can exist, since this question is undecidable. It is only semi-decidable: the best one can get is a program that terminates with "yes" if $F \equiv G$, and otherwise terminates with "no" or does not terminate. Steps for this:
1. Convert $(F \wedge \neg G) \vee (\neg F \wedge G)$ into its clausal form S_0. We have $F \equiv G$ iff S_0 unsat.
2. Compute the closure under resolution+factoring of S_0 by levels, in successive steps for $i = 0, 1, 2 \ldots$:
   2a: If the empty clause is in S_i, terminate with "yes: $F \equiv G$".
   2b: Otherwise, obtain S_{i+1} by adding to S_i all new clauses one can get by one step of resolution or factoring on clauses in S_i.
   2c: If no new clause was obtained from S_i, terminate with "no"; else, go to 2a with the next i.

5) Formalize and prove by resolution that sentence E is a logical consequence of the other four.
A: If a person likes logic, he does not like football.
B: Brothers of football players like football.
C: Messi is a football player and Ney is his brother.
D: Ney likes logic.
E: Our teacher is a nice guy who knows a lot about football and logic.

`Answer:` We prove that $A \wedge B \wedge C \wedge D$ is unsatisfiable and therefore $A \wedge B \wedge C \wedge D \models E$. Formalizing with unary predicates ll, lf, fp, binary predicate br, the constants $messi$ and ney, and expressing the sentences in clausal form, we get the clauses:

$A)$ $\neg ll(X) \vee \neg lf(X)$
$B)$ $\neg fp(X) \vee \neg br(X,Y) \vee lf(Y)$
$C1)$ $fp(messi)$
$C2)$ $br(messi, ney)$
$D)$ $ll(ney)$

By resolution we obtain the empty clause as follows:

num :	*by :*	*mgu :*	*get :*
1)	$res(A,D)$	$X = ney$	$\neg lf(ney)$
2)	$res(B,C1)$	$X = messi$	$\neg br(messi,Y) \vee lf(Y)$
3)	$res(2,C2)$	$Y = ney$	$lf(ney)$
4)	$res(3,C2)$	$Y = ney$	$\square$

6) Complete the following graph coloring program (see problem 2). Do `makeConstraints` recursively, using `#\=` and the built-in predicate `nth1(I,L,X)` ("the `I`th element of the list `L` is `X`").

```
:- use_module(library(clpfd)).
numVertices(5).
edges([ 1-2, 1-3, 2-3, 2-4, 2-5, 3-5 ]).
numColors(3).

main:- numVertices(N),edges(Edges), listOfNPrologVars(N,Vars), ...
       Vars ins ...
       makeConstraints(Edges,Vars),
       ...
       write(Vars), nl.

makeConstraints(...

listOfNPrologVars(...
```

Answer:

```
main:- numVertices(N),edges(Edges), listOfNPrologVars(N,Vars), numColors(K),
       Vars ins 1..K,
       makeConstraints(Edges,Vars),
       label(Vars), write(Vars), nl.

makeConstraints([],_).
makeConstraints( [ U-V | Edges ], Vars ):-
        nth1( U, Vars, VarU ),
        nth1( V, Vars, VarV ),
        VarU #\= VarV,
        makeConstraints(Edges,Vars).

listOfNPrologVars(0,[]):-!.
listOfNPrologVars(N,[_|Vars]):- N1 is N-1, listOfNPrologVars(N1,Vars).
```