Lógica en la Informática / Logic in Computer Science

Friday June 7th, 2013

Time: 2h45min. No books, lecture notes or formula sheets allowed.
The part about propositional logic comprises the first three questions.
SOLUTIONS

1A) Given two propositional formulas F and G, is it true that $F \to G$ is a tautology iff $F \wedge \neg G$ is unsatisfibale? Prove it using only the formal definitions of propositional logic.

Solution: It is true:

$F \to G$ tautology	iff	[def. $\to$]
$\neg F \vee G$ tautology	iff	[def. tautology]
for every I, $eval_I(\neg F \vee G) = 1$	iff	[def. $eval_I \vee$]
for every I, $max(eval_I(\neg F), eval_I(G)) = 1$	iff	[def. $eval_i \neg$]
for every I, $max(1 - eval_I(F), eval_I(G)) = 1$	iff	[arithmetic: $eval_I(F) = 0$ or $eval_I(G) = 1$]
for every I, $min(eval_I(F), 1 - eval_I(G)) = 0$	iff	[def. $eval_I \neg$]
for every I, $min(eval_I(F), eval_I(\neg G)) = 0$	iff	[def. $eval_I \wedge$]
for every I, $eval_I(F \wedge \neg G) = 0$	iff	[def. unsat.]
$F \wedge \neg G$ unsatisfiable		

1B) Given two propositional formulas F and G, is it true that $F \models G$ iff $F \to G$ is satisfiable? Prove it using only the formal definitions of propositional logic.

Solution: It is false. A counterexample is as follows: let F be the formula p and G be the formula $\neg p$. The resulting formula $p \to \neg p$ is satisfiable ($I(p) = 0$ is a model), but $p \not\models \neg p$, as the interpretation $I(p) = 1$ is a model of p but not a model of $\neg p$.

2) Consider a set of propositional clauses F, a clause $C \vee l_1 \in F$ and a binary clause $l_1 \vee l_2 \in F$. Prove, by reasoning about interpretations, that $F \setminus \{C \vee l_1\} \cup \{C \vee l_1 \vee \neg l_2\} \equiv F$.

Solution: Let F' be the left-hand side $F \setminus \{C \vee l_1\} \cup \{C \vee l_1 \vee \neg l_2\}$.

In general, to prove $F' \equiv F$, it suffices to show A) that any model of F' is also a model of F, and B) that any model of F is also a model of F'. Here, part B) is trivial because the only difference is that a literal has been added to a clause.

Let us now do A): pick a model I of F'. We only need to prove that $I \models C \vee l_1$ (the only clause in F that is not in F'). If $I(l_1) = 1$ we are done. If $I(l_1) = 0$ then $I(l_2) = 1$ (because $I \models F'$ and $l_1 \vee l_2 \in F'$), and then (since $I \models C \vee l_1 \vee \neg l_2$) also $I \models C \vee l_1$.

Imagine we want to check the satisfiability of a set of clauses $\{p \vee q,\ q \vee r,\ q \vee \neg r\} \cup G$. Explain why, by applying the previous result, one can remove the clause $p \vee q$ and only check the satisfiability of $\{q \vee r,\ q \vee \neg r\} \cup G$.

Solution: By picking $C = p$, $l_1 = q$ and $l_2 = r$ we can apply the previous result and use $p \vee q \vee \neg r$ instead of $p \vee q$ to check the satisfiability. But now we can apply the result again with $C = p \vee \neg r$, $l_1 = q$ and $l_2 = \neg r$ to use instead the clause $p \vee q \vee \neg r \vee r$. Since it is a tautology we can now remove it to check satisfiability.

3) We have a list of n professional programmers $\{p_1 \dots p_n\}$ and a set of m programming tasks $\{t_1 \dots t_m\}$. We know, for each programmer p_i, the amount of euros e_i (s)he charges; that is, p_i receives e_i euros in total iff (s)he does at least one task (*not* e_i per task, but in total). For each task t_i, we also know which subset $S_i \subseteq \{p_1 \dots p_n\}$ of programmers have the right skills to do the task t_i.

We want to know whether we can handle all tasks with less than K euros. Which (and how many) variables and clauses do we need to do this using SAT? Note: if needed, you can leave arithmetical constraints (at-most-one, cardinality, pseudo-Boolean...) without encoding them into SAT.

Solution: We introduce n propositional variables x_i meaning "programmer i works" (i.e., "programmer i does one or more tasks"). For each task t_i, we need one clause of length $|S_i|$, saying that at least one programmer of the set S_i works: $\bigvee_{p_j \in S_i} x_j$. We also need one pseudo-Boolean constraint saying that the programmers that work cost less than K euros in total: $e_1x_1 + \cdots + e_nx_n < K$. The number of clauses needed for this pseudo-Boolean constraint depends on the encoding used.

4) Suppose we describe undirected graphs in Prolog using predicates as in the following example:

```
vertices([1,2,3,4]).
edge(1,2).
edge(1,4),
edge(2,3),
edge(3,4),
nonedge(1,3).
nonedge(2,4).
```

Program in Prolog a new predicate `tree(T)` which means that `T` is a subset of the vertices that forms a tree: all vertices in `T` are connected but there are no cycles within `T`. Also give all auxiliary predicates.

Solution:

```
tree(T):- vertices(V), subset(V,T), connected(T), \+hasCycle(T).

connected([]):-!.
connected([X|V]):- con([X],V).  % here [X] is the already connected part
con(_,[]).  % Below, find vertex Y to add to the already connected part C:
con(C,V):- memberRest(Y,V,Rest), member(X,C), edge1(X,Y), !, con( [Y|C], Rest ).

hasCycle(T):- subset(T,C), permutation(C,P), P=[First|_], isCycle(First,P).
isCycle(First,[Last] ):- edge1(Last,First),!.
isCycle(First,[X,Y|L]):- edge1(X,Y), isCycle( First, [Y|L] ), !.

%%%%% well-known auxiliary predicates:
subset([],[]).
subset([X|L],[X|S]):-subset(L,S).
subset([X|L],   S ):-subset(L,S).
permutation( [X|L], Perm ):- permutation(L,P), append(P1,P2,P), append(P1,[X|P2],Perm).
memberWithRest(X,Set,Rest):- append(A,[X|B],Set),  append(A,B,Rest).
edge1(X,Y):- edge(X,Y). % edge1: edges in both directions
edge1(X,Y):- edge(Y,X).
```

5) Using the CLP module for finite domains (clpfd) of SWI prolog, we can force labelings of the variables to *maximize* a given expression. For example, the following code writes `[5,5]`:

```
[X,Y] ins 1..10,  X+Y #= 10, labeling([max(X*Y)],[X,Y]),  write([X,Y]).
```

Now you are flying back from China, and you should write such a program to compute how many units of each one of six products you shoould take in your suitcase with capacity 80Kg, if you want to maximize the total value, and the products have the following weights (Kg) and values (Euros):

	p_1	p_2	p_3	p_4	p_5	p_6
weight:	1	2	3	5	6	7
value:	1	4	7	11	14	15

Solution:

```
china:-  L = [A,B,C,D,E,F],
         L ins 0 .. 80,   1*A + 2*B + 3*C +  5*D +  6*E +  7*F #=< 80,
         labeling( [ max( 1*A + 4*B + 7*C + 11*D + 14*E + 15*F ) ], L ), write(L), nl,!.
```

6) Formalize in first-order logic and prove by resolution that $A \wedge B \wedge C \wedge D \models E$:
A) All owners of crazy dogs are stupid
B) Everything has some owner
C) There are no stupid people
D) Pluto is a crazy dog
E) Mourinho is very well educated

Solution: We need to prove that $A \wedge B \wedge C \wedge D \wedge \neg E$ is unsatisfiable. In fact, here $A \wedge B \wedge C \wedge D$ is already unsatisfiable (and E has nothing to do with them). Note that, intuitively, Pluto is a crazy dog (by D) with some owner (by B) that is stupid (by A), and this contradicts C, so one needs all four these sentences.

A) All owners of crazy dogs are stupid (here $O(x, y)$ means "x owns y"):
$\forall x(\exists y O(x, y) \wedge CD(y)) \rightarrow S(x)$
$\forall x \neg(\exists y O(x, y) \wedge CD(y)) \vee S(x)$
$\forall x \forall y \neg O(x, y) \vee \neg CD(y) \vee S(x)$

B) Everything has some owner:
$\forall x \exists y O(y, x)$
$\forall x O(f_y(x), x)$

C) There are no stupid people:
$\forall x \neg S(x)$

D) Pluto is a crazy dog:
$CD(pluto)$

So we get four clauses:
A) $\neg O(x, y) \vee \neg CD(y) \vee S(x)$
B) $O(f_y(z), z)$
C) $\neg S(v)$
D) $CD(pluto)$.

By resolution we get:
1) $\neg O(x, pluto) \vee S(x)$ (from A and D)
2) $S(f_y(pluto))$ (from 1 and B)
3) [] (from 2 and C)