Lgica en la Informtica / Logic in Computer Science
June 20th, 2017. Time: 2h30min. No books or lecture notes.

Note on evaluation:

eval(propositional logic) = max{ eval(Problems 1,2,3), eval(partial exam) }.

eval(first-order logic) = eval(Problems 4,5,6).

1 Consider the at-most-one (AMO) constraint, expressing that at most one of the propositional variables $x_1 \dots x_n$ is true, also written $x_1 + \dots + x_n \leq 1$. Consider:

1) the encoding for AMO you know that needs the smallest (in terms of n) number of clauses, and

2) the encoding that needs the smallest number of auxiliary variables.

For each case, write **giving no further explanations**: a) the name of the encoding, b) which, and how many, auxiliary variables it uses, c) which, and how many, clauses (always expressing how many in terms of n).

Answer: 1) a) the ladder encoding. b) n auxiliary variables $a_1 \dots a_n$. c) $3n$ clauses: $\neg a_i \vee a_{i+1}$, $\neg x_i \vee a_i$, $\neg a_i \vee \neg x_{i+1}$, for $1 \leq i \leq n$.

OR a) the Heule-3 encoding. b) $n/2$ auxiliary variables. c) $3n$ clauses, by expressing $x_1 + \dots + x_n \leq 1$ as $x_1 + x_2 + x_3 + a \leq 1 \ \wedge \ \neg a + x_4 + \dots + x_n \leq 1$, and using $\binom{4}{2} = 6$ clauses for $x_1 + x_2 + x_3 + a \leq 1$.

2) a) the quadratic encoding. b) no auxiliary vars c) the $\binom{n}{2}$ clauses $\neg x_i \vee \neg x_j$, for $1 \leq i < j \leq n$.

2 My friend John says that he has found a new way to speed up SAT solving. Before starting his SAT solver, he removes from the set of clauses S some clauses he calls “unnecessary”:

A: if there is some variable x that appears only in positive literals of clauses of S, then he removes from S all clauses containing x

B: similarly, if some variable y appears in S only in negative literals then he removes from S all clauses containing y.

Note that after eliminating some “unnecessary” clauses, step A or B may be (or become) applicable for other variables, so John continues doing this until no more variables of type A or B exist and then launches his solver on a (hopefully) much smaller set of clauses. Is John’s idea correct? Explain why, **in very few words**.

Answer: Yes. Each time such a subset of clauses is removed, the satisfiability is not changed. For case A, let S be the set $S' \cup \{x \vee C_1, \dots, x \vee C_k\}$ where x does not appear in S'. Then S and S' are equisatisfiable: if I is an interpretation with $I \models S$ then $I \models S'$ because $S \supseteq S'$; reversely, if $I' \models S'$ then we can extend I' to a an interpretation I with $I(x) = 1$ and we get $I \models S$. Case B is of course analogous (extending I' with $I(x) = 0$).

3A: What is the complexity of 2-SAT? (just answer, no explanations needed).

3B: Any set of propositional *positive clauses*, that is, clauses with only positive literals (no negations), is of course satisfiable, because the interpretation making all variables true is a model. What is the complexity of deciding the satisfiablity of a given “2-or-positive” set of clauses S, that is, such that every clause in S is either positive or two-literal (or both)? Explain why, **in very few words**.

Hint: with two-literal clauses we can express that one variable is the negation of another variable.

Answer:

3A: 2-SAT is polynomial (linear, in fact).

3B: But if clauses of both kinds are allowed, then it becomes NP-Complete.

If it were polynomial, then SAT for **any** clause set S would be polynomial!

This is because because we can easily transform any clause set S into an **equisatisfiable** 2-or-positive clause set S', by introducing, for each variable x_i, a new variable x'_i. Then S' will consist of the following clauses:

a) For every x_i, two two-literal clauses expressing that x'_i and x_i are the negation of each other: $\neg x_i \vee \neg x'_i$ and $x_i \vee x'_i$.

b) For every clause C of S with more than two literals, a positive clause C' where all negative literals $\neg x_j$ of C have been replaced by positive ones x'_j.

4: Consider the following Prolog program and its well-known behaviour:

```
brother(joan,pere).
father(enric,joan).
uncle(N,U):- father(N,F), brother(F,U).

?- uncle(X,Y).
X = enric,
Y = pere.
```

Express the program as a set of first-order clauses P and prove that $\exists x\, \exists y\; uncle(x, y)$ is a logical consequence of P. Which values did the variables x and y get (by unification) in your proof? **Only write the steps and values. No explanations.**

Answer: We have to prove that $P \wedge \neg(\exists x\, \exists y\; uncle(x, y))$ is unsatisfiable.

Note that `uncle(N,U):- father(N,F), brother(F,U)` is
$\forall N\, \forall U\; uncle(N, U) \leftarrow (\exists F\; father(N, F) \wedge brother(F, U))$, which is
$\forall N\, \forall U\; uncle(N, U) \vee \neg(\exists F\; father(N, F) \wedge brother(F, U))$, which is
$\forall N\, \forall U\; uncle(N, U) \vee (\forall F\; \neg father(N, F) \vee \neg brother(F, U))$.

Furthermore, the negation of $\exists x\, \exists y\; uncle(x, y)$ is $\forall x\, \forall y\; \neg uncle(x, y)$.
So, from $P \wedge \forall x\, \forall y\; \neg uncle(x, y)$ we get four clauses:

1. $brother(joan, pere)$
2. $father(enric, joan)$
3. $uncle(N, U) \vee \neg father(N, F) \vee \neg brother(F, U)$
4. $\neg uncle(x, y)$

By resolution we get:

5. $\neg father(x, F) \vee \neg brother(F, y)$ — by resolution between 3 and 4, $\sigma = \{N = x, U = y\}$
6. $\neg brother(joan, y)$ — by resolution between 5 and 2, $\sigma = \{x = enric, F = joan\}$
7. the empty clause — by resolution between 5 and 2, $\sigma = \{y = pere\}$

Here x and y got the values $x = enric$ and $y = pere$.

5: For each statement, say whether it is true or false and show why **in an as simple and short as possible way**:

5A: The formula $\forall x\, \exists y\; (p(x, f(y)) \wedge \neg p(x, y))$ is satisfiable.

Answer: True. The following interpretation I is a model: $D_I = \{a, b\}$, $p_I(a, a) = 1$, $p_I(a, b) = 0$, $p_I(b, a) = 1$, $p_I(b, b) = 0$, $f_I(a) = a$ and $f_I(b) = a$.

5B: $\forall x\, \forall y\, \exists z\; q(x, z, y) \models \forall x\, \exists z\, \forall y\; q(x, z, y)$.

Answer: False. For following interpretation I, we have $I \models \forall x\, \forall y\, \exists z\; q(x, z, y)$, but
$I \not\models \forall x\, \exists z\, \forall y\; q(x, z, y)$: $D_I = \{a, b\}$, and $q_I(x, y, z) = 1$ iff $y = z$.

6: My good old friend John says that he has written a C++ program P that takes as input an arbitrary first-order formula F, and such that, if F is a tautology, P always outputs "yes" after a finite amount of time, and if F is not a tautology, P outputs "no" or it does not terminate.

Is this possible? If this is not possible, explain why. If it is possible, explain how P would work. **A very short answer suffices.**

Answer: Yes. It is possible. We have F tautology iff $\neg F$ unsatisfiable iff $S = clausal\_form(\neg F)$ unsatisfiable iff the empty clause is in the closure under resolution and factoring of S. So John's program P can implement those steps and:

-If F is a tautology, terminate with output "yes" as soon the empty clause appears (note that this happens after finite time).

-If F is not a tautology, the empty clause will not appear. Then P only terminates (with output "no") if the closure under resolution and factoring of S is finite.