Lógica en la Informática / Logic in Computer Science

June 8th, 2012. Results published: Tuesday June 19. Review: Wednesday June 20, 10h, Omega 139.

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

Note: The answers will also be considered for evaluation of the transversal competences of English and Reasoning (but both with 0 % impact on the global evaluation of LI).

1) When the Canadian wildfire fighters work to extinguish a forest fire, each one of them is equipped with a special mobile radio (there is no mobile phone coverage in the deepest forests). But sometimes communication with the central is difficult due to radio interferences that happen when two or more firefighters are at less than 500m distance from each other. Therefore, it is decided to use two different radio frequencies. Every 10 seconds, all N radios communicate their GPS position to the central, which re-assigns all frequencies based on the list of all pairs of firefighters $(F_1, F_1'), (F_2, F_2') \dots F_m, F_m')$ such that F_i and F_i' are closer than 500m to each other. For each i, different frequencies are given to F_i and F_i'. Note that each firefighter may appear many times in the list of pairs. To be able to solve this problem quickly every 10 seconds, the Canadians decide to use a SAT encoding and a SAT solver.

1a) How would you encode this problem into SAT and which kind of SAT solver would you use?

Answer: Define a SAT encoding with variables x_{ij}, for i in $1..N$ and j in $\{1, 2\}$, meaning that "firefighter i uses frequency j". Then, we need to express that:
-each firefighter gets at least one frequency: one clause $x_{i1} \vee x_{i2}$ for each i in $1..N$ (N clauses)
-each firefighter gets at most one frequency: one clause $\neg x_{i1} \vee \neg x_{i2}$ for each i in $1..N$ (N clauses)
-for each pair of firefighters (i, i') that are closer than 500m to each other, two clauses, saying that they get different frequencies:
$\neg x_{i1} \vee \neg x_{i'1}$ (forbid that both of them get frequency 1)
$\neg x_{i2} \vee \neg x_{i'2}$ (forbid that both of them get frequency 2)
Since the resulting problem is a 2-SAT problem, it is solvable in polynomial time. Any SAT solver that can do this is therefore recommendable. Either a model is found that gives us an adequate frequency assignment, or else the SAT problem is unsatisfible and no such assignment exists. (Note that the problem is equivalent to the polynomial problem of 2-coloring of the graph with firefighters as vertices and with edges between too close firefighters).

1b) Same question, if a given number of frequencies K with $K > 2$ can be used.

Answer: Define a SAT encoding with variables x_{ij}, for i in $1..N$ and j in $1..K$, meaning that "firefighter i uses frequency j".
-each firefighter gets at least one frequency: one clause $x_{i1} \vee \ldots \vee x_{iK}$ for each i in $1..N$ (N clauses)
-each firefighter gets at most one frequency: for each i in $1..N$, an encoding of *at-most-one*$\{x_{i1} \dots x_{iK}\}$; for example, the quadratic encoding requires, for each i, all clauses of the form $\neg x_{ij} \vee \neg x_{ij'}$ for $1 \leq j < j' \leq K$, in total $\binom{N}{2}$ clause for each i.
-for each pair of firefighters (i, i') that are too close, K clauses saying that they get different frequencies: $\neg x_{ij} \vee \neg x_{i'j}$ (forbid that both of them get frequency j, for $1 \leq j \leq K$)
The resulting problem is no longer a 2-SAT problem and it is not polynomial. It is the NP-complete problem of K-coloring a graph, the same graph as before. Any state-of-the-art SAT solver can be used.

2) In SAT, the *at-most-K* constraint states that, of n variables $\{x_1, \dots, x_n\}$, at most K are true.

2a) How would you express this constraint in CNF without using any auxiliary variables? How many clauses are needed?

Answer: We would need one clause of the form $\neg y_1 \vee \ldots \vee \neg y_{K+1}$ for each subset $\{y_1 \dots y_{K+1}\}$ of $\{x_1, \dots, x_n\}$, meaning that of each subset of $K + 1$ variables at least one must be false. So in total

$\binom{n}{K+1} = (\ n \cdot (n-1) \cdots (n-K+1)\ )\ /\ K!$ clauses are needed. This amount grows exponentially in n if, for example, $K = n/2$: $\binom{40}{20} = 137{,}846{,}528{,}820$.

2b) Which other encoding (using auxiliary variables) requires less clauses?
How many clauses are needed and how many auxiliary variables?

Answer: For example, sorting networks, which require $O(n \log^2 n)$ clauses and also $O(n \log^2 n)$ auxiliary variables. Other encodings exist (which work especially well if $k << n$) based on modules with 2^k clauses for counting how many variables of each group of k are true, and merging these groups, which requires $O(k^2)$ clauses for each merger, and at most $2n$ auxiliary variables. (see the lecture notes for the details to be added to this answer).

3) Consider a first-order interpretation I with a finite domain $D_I = \{0, 1\}$ and the full descriptions of f_I and of P_I for a 1-ary function symbol f and a 3-ary predicate symbol P.

3a) Is it decidable whether I satisfies a given formula F? If so, what is the complexity of this?

Answer: Yes, this is decidable. If the domain is finite, one can apply the standard evaluation algorithm: if the outermost quantifier is $\forall x$, then one has to check that the remaining formula evaluates to true *for every* value of the domain for x; if it is $\exists x$, then it s for *some* value, etc. This clearly terminates, but the complexity is high. The best known algorithms are exponential (in fact it is P-space complete in general, i.e., it is believed to be even harder than NP-complete problems). To see that it is at least as hard as 3-SAT even for this simple set of symbols, note that a 3-SAT problem like $(\overline{x_7} \vee x_8 \vee \overline{x_2})\ \wedge\ \ldots$ is satisfiable iff the formula $\exists x_1 \exists x_2 \ldots \exists x_n\ P(f(x_7), x_8, f(x_2)) \wedge \ldots$ evaluates to true if f is interpreted as logical negation ($f_I(0) = 1$ and $f_I(1) = 0$) and $P_I(x, y, z) = true$ iff at least one of its arguments is 1.

3b) Now consider an interpretation I that has as (infinite) domain the integer numbers, and two 2-ary function symbols f and g interpreted as the integer sum and product respectively, and a 1-ary predicate symbol P interpreted as $P_I(0) = true$ and $P_I(x) = false$ if $x \neq 0$. Is it decidable whether I satisfies a given formula F? If so, what is the complexity of deciding this?

Answer: This is undecidable. One can express well-known undecidable problems like checking whether an arbitrary integer polynomial over several variables has roots: for example $x^3y+3z^2+\ldots = 0$ has solutions iff $\exists x \exists y \exists z\ \ P(\ f(f(g(\ g(g(x,x),x),y),g(z,z)),\ldots)\ )$ evaluates to true in this interpretation.

4) Formalize and prove by resolution that sentence D is a logical consequence of the other three:
A: Everybody loves his father and his mother.
B: John is stupìd.
C: When someone is stupid, at least one of his parents is stupid too.
D: There are stupid people that are loved by someone.

Answer: For the vocabulary: j is a constant symbol for "John", $f(x)$ and $m(x)$ are function symbols "for father of x" and and "mother of x"; $S(x)$ is a predicate symbol: "x is Stupid"; $L(x, y)$ is a predicate symbol: "x loves y", the sentences become:

A: $\quad \forall x\ (L(x, f(x)) \wedge L(x, m(x)))$
B: $\quad S(j)$
C: $\quad \forall x\ (S(x) \rightarrow (S(f(x)) \vee S(m(x))))$
$\neg D$: $\quad \neg \exists x\ \exists y\ (S(x) \wedge L(y, x))$

In clausal form, these become:

$A1$. $\quad L(x, f(x))$
$A2$. $\quad L(x, m(x))$
B. $\quad S(j)$
C. $\quad \neg S(x) \vee S(f(x)) \vee S(m(x))$
$\neg D$. $\quad \neg S(x) \vee \neg L(y, x)$

By resolution we obtain:

6. $S(f(j)) \vee S(m(j))$ $B + C$
7. $\neg L(y, f(j)) \vee S(m(j))$ $D + 6$
8. $\neg L(y, f(j)) \vee \neg L(z, m(j))$ $D + 7$
9. $\neg L(z, m(j))$ $A1 + 8$
10. $\square$ $A2 + 9$

5) Complete the following DPLL-like procedure for SAT in Prolog. Program everything, except built-in predicates of GNU-Prolog and the predicate `readclauses(F)`, which reads a list of clauses, where each clause is a list of integers. For example, $p_3 \vee \neg p_6 \vee p_2$ is represented by `[3,-6,2]`. It is mandatory to follow the indications, and do things as simple as possible.

```
p:- readclauses(F), dpll([],F).
p:- write('UNSAT'),nl.

dpll(I,[]):- write('IT IS SATISFIABLE. Model: '), write(I),nl,!.
dpll(I,F):-
    decision_lit(F,Lit),  % Must select a unit clause if there is any. Otherwise, an arbitrary one.
    simplif(Lit,F,F1),    % Simplifies F. Warning: may fail and cause backtracking
    dpll( ...   ,   ... ).
```

Answer:

```
p:- readclauses(F), dpll([],F).
p:- write('UNSAT'),nl.

dpll(I,[]):- write('IT IS SATISFIABLE. Model: '), write(I),nl,!.
dpll(I,F):-
    decision_lit(F,Lit),
    simplif(Lit,F,F1),
    dpll( [Lit|I], F1 ).

decision_lit(F,Lit):- member([Lit],F),!.  % unit propagation if possible
decision_lit([[Lit|_]|_],Lit).            % otherwise, select 1st lit of 1st clause of F
decision_lit([[Lit|_]|_],Lit1):- Lit1 is -Lit.   % or its negation!

simplif(_,[],[]).
simplif(Lit, [C|S], S1 ):-       % remove true clause (clause containing Lit)
    member( Lit, C ),  !,
    simplif(Lit,S,S1), !.
simplif(Lit, [C|F], [C1|F1] ):-  % remove false lit -Lit from clause
    Lit1 is -Lit,
    memberAndRest( Lit1, C, C1 ),!,
    C1 \= [],                    % causes failure if empty clause detected
    simplif(Lit,F,F1),!.
simplif(Lit, [C|F], [C|F1] ):-   % nothing to simplify in 1st clause
    simplif(Lit,F,F1),!.

memberAndRest(X,[X|L],L).
memberAndRest(X,[Y|L],[Y|L1]):- memberAndRest(X,L,L1).
```