Lógica en la Informática / Logic in Computer Science

Monday June 16th, 2014

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

1) We want to encode the at-most-one (AMO) constraint $x_1 + \ldots + x_{10} \leq 1$ using 24 clauses and 3 auxiliary variables a, b, and c. Write the smaller constraints into which the constraint is split and write the clauses for each part.

Solution: Using the Heule-3 encoding, we have that $AMO(x_1, \ldots, x_{10})$ is equivalent to
$AMO(x_1, x_2, x_3, a)$ and $AMO(\neg a, x_4, \ldots, x_{10})$. The second constraint is equivalent to:
$AMO(\neg a, x_4, x_5, b)$ and $AMO(\neg b, x_6, \ldots, x_{10})$, where the latter is equivalent to:
$AMO(\neg b, x_6, x_7, c)$ and $AMO(\neg c, x_8, x_9, x_{10})$.
So in total we get four 4-literal AMO constraints:
$AMO(x_1, x_2, x_3, a)$ and $AMO(\neg a, x_4, x_5, b)$ and $AMO(\neg b, x_6, x_7, c)$ and $AMO(\neg c, x_8, x_9, x_{10})$.
For each one of them the quadratic encoding requires 6 clauses, e.g., $AMO(x_1, x_2, x_3, a)$ generates:
$\neg x_1 \vee \neg x_2$, $\neg x_1 \vee \neg x_3$, $\neg x_1 \vee \neg a$, $\neg x_2 \vee \neg x_3$, $\neg x_2 \vee \neg a$ and $\neg x_3 \vee \neg a$,

2) Consider an (undirected) graph: a set of vertices V and a set of edges E. We want to *color* it with K colors: each vertex of V gets exactly one of the K colors such that, for each edge (i, i'), vertices i and i' get different colors. Explain very briefly what is the best way to do this using SAT for different values of K, and why.

Solution: We use a SAT encoding with $|V| \cdot K$ variables x_{ij} meaning "vertex i gets color j".
We need $|V|$ clauses with K literals expressing that each vertex gets at least one color:
for each vertex i, a clause $x_{i1} \vee \ldots \vee x_{iK}$.
We also need K 2-literal clauses per edge:
for each edge (i, i') and each color j, a clause $\neg x_{ij} \vee \neg x_{i'j}$.
If $K \leq 2$ this gives us a 2-SAT problem that can be solved in polynomial time using a specialized 2-SAT solver (e.g., based on resolution, or on detecting cycles in a graph).
If $K > 2$ the problem is NP-hard (so use any good SAT solver).

3A) Consider a number of N items, each item $i \in 1..N$ having a weight of w_i kg and a price of p_i euros. We also have a number of M trucks, each one of capacity 1000kg. Explain how to use a SAT solver to select a *subset* (not necessarily all) of the items that will be carried, such that the total price of all items carried is at least P euros. and to assign each selected item to some truck (respecting truck capacities). Clearly indicate which variables you use and their precise meaning, and which properties you impose using which clauses or which constraints (for cardinality or pseudo-Boolean constraints, it is not necessary to write their encodings into clauses).

Solution: Consider $N \cdot M$ propositional variables x_{ij} meaning "item i will be carried by truck j".
For each truck j, one pseudo-Boolean constraint expressing that its capacity is not exceeded:
$w_1\, x_{1j} + \ldots + w_N\, x_{Nj} \leq 1000$
One pseudo-Boolean constraint to express that the total price is at least P:
$p_1\, x_{11} + \ldots + p_1\, x_{1M} \;+\; p_2\, x_{21} + \ldots + p_2\, x_{2M} \;+\; \ldots \;+\; p_N\, x_{N1} + \ldots + p_N\, x_{NM} \;\geq\; P$
[Optional] For each item i, one AMO constraint expressing that it is in at most one truck:
$AMO(x_{i1}, \ldots, x_{iM})$

3B) How would you use a SAT solver to maximize the total price?

Solution: First run a SAT solver without the pseudo-Boolean constraint for the price. This is always satisfiable, giving a first solution with price k. Then run again, with the additional constraint $p_1x_{11} + \ldots + p_1x_{1M} \;+\ldots+\; p_Nx_{N1} + \ldots + p_Nx_{NM} \geq k+1$ Repeat this: each time we get a solution with price k, search for another one with price at least $k+1$. Repeat this until the problem becomes unsatisfiable, indicating that the last solution found was the optimal one.

4) Prove using first-order logic that sentence 5 is a logical consequence of the first four:
1. All dogs make noise.
2. Anyone who has a cat has no mice.
3. Neurotic people do not have any noisy animals.
4. John has either a cat or a dog.
5. If John is neurotic he has no mice.

Solution:

1.$\forall x\ Dog(x) \rightarrow Noisy(x)$
2.$\forall x\ (\exists y(Has(x,y) \wedge Cat(y)) \rightarrow \neg\ \exists z\ Has(x,z) \wedge Mouse(z)$
3.$\forall x\ Neurotic(x) \rightarrow \neg \exists y\ Has(x,y) \wedge Noisy(y)$
4.$\exists x\ Has(John,x) \wedge (Cat(x) \vee Dog(x))$
not5.$\neg(Neurotic(John) \rightarrow \neg\ \exists x\ Has(John,x) \wedge Mouse(x))$

Transforming into clausal form, intermediate steps are:

2. $\forall x\ (\exists y\ (Has(x,y) \wedge Cat(y)) \rightarrow \neg\ \exists z\ Has(x,z) \wedge Mouse(z)$ becomes :
   $\forall x\ \neg(\exists y\ (Has(x,y) \wedge Cat(y)) \vee \neg\ \exists z\ Has(x,z) \wedge Mouse(z)$ becomes :
   $\forall x\ \forall y\ \neg Has(x,y) \vee \neg Cat(y)) \vee \neg\ \exists z\ Has(x,z) \wedge Mouse(z)$ becomes :
   $\forall x\ \forall y\ \neg Has(x,y) \vee \neg Cat(y)) \vee \forall z\ \neg Has(x,z) \vee \neg Mouse(z)$
3. $\forall x\ Neurotic(x) \rightarrow \neg \exists y\ Has(x,y) \wedge Noisy(y)$ becomes :
   $\forall x\ \neg Neurotic(x) \vee \neg \exists y\ Has(x,y) \wedge Noisy(y)$ becomes :
   $\forall x\ \neg Neurotic(x) \vee \forall y\ \neg Has(x,y) \vee \neg Noisy(y)$ becomes :
   $\forall x\ \forall y\ \neg Neurotic(x) \vee \neg Has(x,y) \vee \neg Noisy(y)$
4. $\exists x\ Has(John,x) \wedge (Cat(x) \vee Dog(x))$ becomes :
   $Has(John,c) \wedge (Cat(c) \vee Dog(c))$

not5. $\neg(Neurotic(John) \rightarrow \neg\ \exists x\ Has(John,x) \wedge Mouse(x))$ becomes :
   $\neg(\neg Neurotic(John) \vee \neg\ \exists x\ Has(John,x) \wedge Mouse(x))$ becomes :
   $Neurotic(John) \wedge\ \exists x\ Has(John,x) \wedge Mouse(x)$ becomes :
   $Neurotic(John) \wedge Has(John,d) \wedge Mouse(d)$

and the clausal form is:

1. $\neg Dog(x) \vee Noisy(x)$
2. $\neg Has(x,y) \vee \neg Cat(y) \vee \neg Has(x,z) \vee \neg Mouse(z)$
3. $\neg Neurotic(x) \vee \neg Has(x,y) \vee \neg Noisy(y)$

$4a$: $Has(John,c)$
$4b$: $Cat(c) \vee Dog(c)$
$5a$: $Neurotic(John)$
$5b$: $Has(John,d)$
$5c$: $Mouse(d)$

Which by resolution gives us:

6. $3 + 5a$ $\neg Has(John,y) \vee \neg Noisy(y)$
7. $6 + 4a$ $\neg Noisy(c)$
8. $7 + 1$ $\neg Dog(c)$
9. $8 + 4b$ $Cat(c)$
10. $9 + 2$ $\neg Has(x,c) \vee \neg Has(x,z) \vee \neg Mouse(z)$
11. $10 + 4a$ $\neg Has(John,z) \vee \neg Mouse(z)$
12. $11 + 5b$ $\neg Mouse(d)$
13. $12 + 5c$ []

5) Consider an $n \times n$ chessboard (for any natural number $n > 3$, not necessarily 8), where n is defined by a Prolog clause `boardSize(`n`)`. (for example, `boardSize(14)` if if $n = 14$). Define a Prolog predicate `horse(I1,J1,I2,J2)` that writes the shortest possible sequence of positions that a horse of chess traverses to go from initial position `I1,J1` to final position `I2,J2` on the board (positions are (row,column), each one in $1..n$). It must write the sequence in the right order, the first position being `[I1,J1]`, and write "no solution" if no such a sequence exists.

Solution:

```
horse(I1,J1,I2,J2):-
  boardSize(N), NumSquares is N*N,
  between(0,NumSquares,K),
  path( [I1,J1], [I2,J2], [[I1,J1]], Path ),
  length(Path,K),
  reverse(Path,Path1),
  write(Path1).
horse(_,_,_,_):- write('no solution').

path( E,E, C,C ).
path( CurrentState, FinalState,  PathSoFar, TotalPath ):-
  oneStep( CurrentState, NextState ),
  \+member(NextState,PathSoFar),
  path( NextState, FinalState, [NextState|PathSoFar], TotalPath ).

reverse([],[]).
reverse([X|L],L1):- reverse(L,L2), append(L2,[X],L1),!.


exists(I,J):- boardSize(N), between(1,N,I), between(1,N,J).

oneStep( [I1,J1], [I2,J2] ):- I2 is I1+1, J2 is J1+2, exists(I2,J2).
oneStep( [I1,J1], [I2,J2] ):- I2 is I1-1, J2 is J1+2, exists(I2,J2).
oneStep( [I1,J1], [I2,J2] ):- I2 is I1+1, J2 is J1-2, exists(I2,J2).
oneStep( [I1,J1], [I2,J2] ):- I2 is I1-1, J2 is J1-2, exists(I2,J2).

oneStep( [I1,J1], [I2,J2] ):- J2 is J1+1, I2 is I1+2, exists(I2,J2).
oneStep( [I1,J1], [I2,J2] ):- J2 is J1-1, I2 is I1+2, exists(I2,J2).
oneStep( [I1,J1], [I2,J2] ):- J2 is J1+1, I2 is I1-2, exists(I2,J2).
oneStep( [I1,J1], [I2,J2] ):- J2 is J1-1, I2 is I1-2, exists(I2,J2).
```