Constraint Programming: Assignment no. 10/11

Marco Kuhlmann, M.Sc., Dipl.-Inform. Guido Tack

http://www.ps.uni-sb.de/courses/cp-ss07/

This week's assignment is about symmetries and arc consistency.

Exercise 10/11.1 (Symmetric golfers)

Modify the social golfers example (exercise 9.1) such that the symmetry among groups and the symmetry among weeks is broken (using static lex-leader style constraints).

Exercise 10/11.2 (Symmetric queens)

In this exercise, you will implement a model for the n-queens problem that contains symmetry breaking constraints for all the symmetries of a chess board.

Start with the classic n-queens model, using three distinct-constraints. Remember that there are eight symmetries: the identity, the flip around the x or y axis, the flip around one of the two diagonals, and rotation of 90, 180, and 270 degrees. We will use lex-leader constraints for breaking these symmetries statically.

a) Add the *dual model* to your script: Use a `VarMatrix<BoolVar>`, where an entry $(i, j) = 1$ iff a queen is placed at (i, j). Use reified constraints to connect the dual and the original model.
b) Implement a method `VarMatrix<BoolVar> r90(VarMatrix<BoolVar> q)` that returns a new matrix that contains the variables from `q`, but rotated by 90 degrees.
c) Implement a method `VarMatrix<BoolVar> d1(VarMatrix<BoolVar> q)` that returns a new matrix that contains the variables from `q`, but flipped around one diagonal.
d) Use lex-leader constraints (`rel`) to break all symmetries. Remember that the symmetry group for a square is generated by r90 and d1.
e) Check that you really break all the symmetries (and don't exclude too many solutions) by entering the number of solutions you get for $n = 1, \ldots, 12$ into the online encyclopedia of integer sequences.

Exercise 10/11.3 (Arc consistency)

Recall algorithm AC-3 from Lecture 11.

```
ac3()
1 Q := {(i, j) | c_{i,j}}
2 while Q not empty do
3 |   remove (k, m) from Q
4 |   if revise(k, m) then
5 |   |   N := {(i, k) | c_{i,k}, i ≠ m}
6 |   |   Q := Q ∪ N
7 |   end
8 end
```

```
revise(i,j)
1 modified := false
2 for x ∈ D_i do
3 |   if ∄ y ∈ D_j with (x, y) ∈ c_{i,j} then
4 |   |   D_i := D_i \ {x}
5 |   |   modified := true
6 |   end
7 end
8 return modified
```

a) Assume that the constraint network is normalized. This means that between two variables i and j, there is at most one constraint $c_{i,j}$, and the constraint $c_{j,i}$ exists if and only if $c_{i,j}$ exists and is the converse of $c_{i,j}$. Explain why under this condition the optimization in line 5 of ac3 is correct, i.e., why the back-edge (m, k) does not have to be considered for revision.

b) The AC algorithms you have seen so far can only deal with binary constraints. The generalization to n-ary constraints is called *generalized arc consistency* (GAC) or *hyperarc consistency*. Remember that the *scope* of a constraint is the set of variables it deals with. Modify algorithm AC-3 such that it achieves GAC. Instead of arcs, the set Q should now contain pairs of a variable and a constraint that has this variable in its scope. The revise procedure gets such a pair as its argument.

c) All AC algorithms presented in the lecture are *variable-centered*, i.e., they keep track of which variables were modified, and then propagate all constraints that belong to these variables. In contrast, the propagation loop as presented in Lecture 4 is *propagator-centered*, it keeps track of which propagators are not at

fix-point. Now assume that the network is not normalized, i.e., there can be several constraints between each pair of variables. Explain how the two approaches differ with respect to fixpoint reasoning and entailment.