LINGI2365 : Constraint Programming

Assignment 3 :

Propagation

F.Massen & Y. Deville

March 2012

1 Objective and practical details

The goal of this assignment is for you to understand and implement constraint propagators.

You should respect the following constraints

- Deadline : **Friday 23 March 2012 2pm**.
- This assignment is **mandatory**.
- This assignment must be completed **by groups of two** (the same groups as in the previous assignments).

Modalities

- A **hardcopy of your report** must be turned in (box in front of the INGI secretary) before 2pm on March 23rd.
- Deposit your files (**code and report pdf**) on your repository in the directory `milestone3` (the SVN server will not accept commit operations after 2pm)
- In case of problems, write to `florence.massen@uclouvain.be`.

2 Questions

1. Explain in detail the time and space complexities of the DC3 algorithm. What is the time complexity if all the constraints are domain consistent when algorithm DC3 is called?
2. In the algorithm `valRemoveAC4`, in the loop

   $$\texttt{forall}(a \in S[y,b,c] : a \in D(x)) \ \{\ldots\}$$

   why do we test if $a \in D(x)$? Would the AC4 algorithm be still correct if this check was removed? Would the time complexity of AC4 change if this check was removed?

3. Let the constraint $X + Y = a$. What is the definition of domain consistency and of bound consistency for this algorithm? Is this constraint automatically bound consistent if it is domain consistent?

3 Problems

Before starting with these problems take a look at the slides "Propagation in Comet" covered in class.

Even if we do not specifically ask for it, we expect you to give a detailed explanation of your implementation for all the problems. The propagation of a constraint is triggered often and should therefore be as efficient as possible. The efficiency of your propagators will be taken into account for grading this assignment.

The `.co` files you submit for each problem must be named as specified in the problem title. Your constraints must respect the expected constructor signature (detailed for each problem).

3.1 AC3 propagator : `AC3Geq.co`

Implement an AC3 propagator for the constraint

$$X \geq Y + a$$

1. What is the definition of domain consistency for this constraint.
2. Are domain consistency and bound consistency equivalent for this constraint? Prove why they are, or why they are not.
3. Implement a constraint-based propagator in Comet as seen in class. The signature of the constructor for your constraint must be `Geq(int a, var<CP>{int} X, var<CP>{int} Y)`.

3.2 The channeling constraint : `Channeling.co`

The channeling constraint is used to link two models for a same problem:

$$channel(X[], Y) \equiv X[i] = (Y == i)$$

where X is an array of boolean variables and Y is an integer variable.

1. Give the definition of domain consistency for this constraint
2. Implement an AC5 propagator achieving domain consistency for this constraint. The constructor signature must be `Channel(var<CP>{int}[] X, var<CP>{int} Y)`.

3.3 AC2001 propagator

3.3.1 Generic constraint : `AC2001Constraint.co`

Here you are asked to implement a generic (constraint-independent) AC2001 propagator for binary constraints on integer variables. We provide you with a skeleton of the abstract class `AC2001Constraint` in the file `AC2001Constraint.co`. You have to add:

1. The firstSupport data structures. You **MUST** use `trail{int}` variables for these to ensure they are correctly backtracked.
2. the body of the constructor
3. the body of the post method
4. the body of the propagate method

The class `AC2001Constraint` also contains an abstract method `check(int a, int b)`. This method will be implemented by subclasses of `AC2001Constraint`. It returns `true` if couple $(X = a, Y = b)$ respects the implemented constraint. You will need to use this (abstract) method in your propagator.

3.3.2 Specific constraints : `AllConstraints.co`

We provide you with a file `AllConstraints.co`. Some examples of constraints are implemented at the end of the file. Implement the method `check` for the first three constraints in the file `AllConstraints.co`. Test your implementation using the file `testAC2001.co`. The solution returned should be $W = 1, X = 4, Y = 1, Z = 3$.

3.4 The AllDifferent constraint : `AllDiffFC.co`

We want you to implement a propagator for the AllDifferent constraint achieving weaker consistency than domain consistency. The constraint $AllDiff(X_1, \ldots, X_k)$ states that all variables $X_1, \ldots, X_k$ must take different values. The decomposition in binary constraints is $\bigwedge_{1 \leq i < j \leq k} X_i \neq X_k$.

1. Define forward-checking consistency for the **the decomposition of** $AllDiff$ **in binary constraints**.
2. Implement a propagator ensuring forward-checking consistency for the decomposition of AllDiff (`AllDiffFC(var{int}[] X)`).
3. Modify the `Nqueens.co` file provided into file `NqueensFC.co` such that it uses your implementation of alldifferent (you will need to include your `AllDiffFC.co` file).

4. Test both yours (`NqueensFC.co`) and Comet's (`Nqueens.co`) implementation of AllDifferent on the NQueens problem. Compare the number of failures and choices between the two implementations and for each $n = 12 \ldots 32$ in a table.

5. What did you find in your comparative study? Explain the results!