# Constraint Technology (course 1DL023)
# Autumn 2008 – Assignment 1

Pierre Flener

Department of Information Technology, Uppsala University, Sweden

— Hard deadline: 18:00 on Friday 19 September 2008 —

## Submission Instructions

1. This assignment is to be solved in teams of (at most) two students. Take our warning on plagiarism very seriously. *We assume that by submitting a solution you are certifying that it is solely the work of your team, and that everyone on the team contributed to elaborating this solution.*
2. Write clear answers. Justify all answers, except where explicitly not required. State any assumptions you make. Spell-check and grammar-check your answers.
3. Document each program you wrote by:
   - A mathematical description of its model (in pseudocode, as in the lectures), namely its decision variables, redundant decision variables, domains of the decision variables, instance variables, problem constraints, channelling constraints, implied constraints, symmetry-breaking constraints, branching heuristics, search order, etc.
   - Instructions on how to compile and run it.
   - Sample test-run commands with inputs and outputs. (Check whether these test runs are reproducible by the program you submit.)

   Verify whether you are using version 2.2.0 of Gecode/J.
4. Put your answer files into a folder named *Lastname.Firstname* or $Lastname_1.Firstname_1$–$Lastname_2.Firstname_2$. Do not use any special characters in the folder name. Include only source code files, that is remove any executable files. Textual answers must be in *.pdf* or *.txt* format. Include a *ReadMe.txt* file to explain the purpose of each other file. Comply strictly with any answer filenames imposed by the questions. Make a *.tgz* or *.zip* compressed archive of maximum 1MB from this folder. *Verify whether it decompresses properly, reproducing your folder exactly, and actually corresponds to Assignment 1.*
5. Submit your compressed folder via the Course Manager server (whose clock may differ from yours) by the deadline given above. This deadline is hard and no other method of submission will be accepted, except in cases of force majeure. Late solutions will be penalised by 1 point for each 12h of delay, but solutions that are late by more than 60h will get 0 (zero) points.
6. When working in a team, *each* team member must submit a copy of the same solution. Only one of the submitted solution copies of a team will be graded, so make sure they are identical. The lateness penalty, if any, for a team will be determined by the moment of its *last* submitted solution copy.

**Failure to follow the instructions above may result in 0 (zero) points**, as we reserve the right to process your solutions automatically.

## Question 1: $n$-Queens Revisited (3 points)

We have seen the probably most popular model of the $n$-queens problem (see `http://mathworld.wolfram.com/QueensProblem.html`), based on three *distinct* constraints, without any implied constraints or symmetry-breaking constraints. This model is implemented in the `Queens.java` program at `http://www.it.uu.se/research/group/astra/gecode/examples/` (also available in the Gecode/J distribution). Perform the following tasks:

a. Specify the operating system, CPU, CPU speed (in GHz), and cache size (in KB) of the computer you chose for this question. (Hint: Under Linux/Unix, do `uname -a` and `more /proc/cpuinfo` to find this information.)

b. Compile `Queens.java` via `javac examples/Options.java examples/Queens.java`, and run it via `java examples/Queens` $n$. Give the CPU runtimes on the chosen computer and the numbers of failures until the *first* solution for $n = 1, 2, \ldots, 100$, in tabular or plotted form. (Hints: The command `java examples/Queens` $n$ `-nogui -mode time` gives the CPU runtime in milliseconds until the first solution without displaying it, while `java examples/Queens` $n$ `-nogui` reveals the number of failures in addition to the first solution and other statistics. The command `java examples/Queens -help` displays other command-line options. Note that `-solutions 0` gives all solutions.)

c. Now consider the model where each position on the chess board is represented by a Boolean decision variable $q_{ij}$ such that $q_{ij} = \mathit{true}$ if and only if there is a queen on row $i$ and column $j$. Implement this model in Gecode/J as a new program, to be called `queens01.java`. You may find the functions in the file `http://www.it.uu.se/research/group/astra/gecode/help-A1.txt` useful.

d. Experiment for `queens01.java` with at least eight combinations of predefined variable and value ordering heuristics. Identify the seemingly fastest combination until the *first* solution and explain why it beats the other ones. For each combination, give the CPU runtimes on the chosen computer and the numbers of failures until the *first* solution for $n = 1, 2, \ldots, 100$, in tabular or plotted form.

e. Expand the table or plot of task b with the CPU runtimes on the chosen computer and the numbers of failures until the *first* solution of `queens01.java` for $n = 1, 2, \ldots, 100$ for the fastest combination of branching heuristics you identified in task d, in tabular or plotted form. Identify the seemingly most efficient of the two programs and explain why it is more efficient.

Keep in mind the general instructions about programs on the first page.

## Question 2: Magic Hexagon (2 points)

We have seen the magic square problem. This question considers a related problem, where the layout is hexagonal (see `http://mathworld.wolfram.com/MagicHexagon.html`):

```
    A B C
   D E F G
  H I J K L
   M N O P
    Q R S
```

There are 19 letters. The goal is to find an assignment to each letter of an integer within $\{1, \ldots, 19\}$ such that any two letters take different values and the sums of all rows and diagonals are the same: $\mathtt{A} + \mathtt{B} + \mathtt{C} = \mathtt{D} + \mathtt{E} + \mathtt{F} + \mathtt{G} = \cdots = \mathtt{A} + \mathtt{D} + \mathtt{H} = \mathtt{B} + \mathtt{E} + \mathtt{I} + \mathtt{M} = \cdots = \mathtt{H} + \mathtt{M} + \mathtt{Q}$. Perform the following tasks:

a. Specify the operating system, CPU, CPU speed (in GHz), and cache size (in KB) of the computer you chose for this question.

b. Write a Gecode/J program called `hexagon1.java` posting just the problem constraints. For the variable ordering, apply the first-fail principle. Give the CPU runtimes on the chosen computer and the numbers of failures until the *first* solution, in tabular or plotted form, under all predefined value orders, and state those first solutions. Identify the seemingly fastest value order until the *first* solution and explain why it beats the other ones. How many solutions are there?

c. Derive implied constraints in a way similar to the one for the magic square problem. Add these constraints to `hexagon1.java`, giving a new program, to be called `hexagon2.java`. Give the CPU runtimes on the chosen computer and the numbers of failures until the *first* solution, in tabular or plotted form, under all predefined value orders, and state those first solutions. Is the seemingly fastest value order until the *first* solution still the same? Is the new program more efficient? Is the first solution still the same? Explain your answers.

d. Identify the symmetries of the problem and state suitable symmetry-breaking constraints in mathematical notation. Add these constraints to `hexagon1.java` and `hexagon2.java`, giving two new programs, to be called `hexagon13.java` and `hexagon23.java`, respectively. Give the CPU runtimes on the chosen computer and the numbers of failures until the *first* solution, in tabular or plotted form, under all predefined value orders, and state those first solutions. Is the seemingly fastest value order until the *first* solution still the same? Are the new programs more efficient? Are the first solutions still the same? Explain your answers. How many solutions are there now?

Keep in mind the general instructions about programs on the first page.

## Question 3: Consistency (3 points)

Using the *propagate* algorithm of the course, namely the version with events (also known as propagation conditions) and status messages (and thus without the set $MV$ of modified decision variables), perform the pre-search propagation of the following two constraint satisfaction problems (CSPs):

$$Q_1 = \langle c_1 : x < y,\ c_2 : x + y < 8,\ c_3 : y > z\ ;\ x \in \{2, \ldots, 10\},\ y \in \{0, \ldots, 10\},\ z \in \{3, \ldots, 10\}\rangle$$

$$Q_2 = \langle c_4 : x + y < z,\ c_5 : x < y,\ c_6 : \mathit{distinct}(\{x, y, z\})\ ;\ x, y, z \in \{1, \ldots, 5\}\rangle$$

The following choices are imposed: Achieve *bounds consistency* for *all* the constraints. Post the constraints in the *left-to-right order* in which they appear in the CSP. Handle the decision variables in the *left-to-right order* in which they are declared in the CSP. Use a *first-in first-out queue* for implementing the collection $N$ of propagators that are not known to be at fixpoint.

In other words, for both $Q_1$ and $Q_2$, perform the following tasks, letting $p_i$ be the propagator of constraint $c_i$:

a. Give (without proof) the smallest set of events that trigger the enqueuing of $p_i$, for each $i$.

b. Give the initial store: $s = \{x \mapsto \{\ldots\},\ y \mapsto \{\ldots\},\ \ldots\}$

c. Give the initial value of the set $P$ of all non-subsumed propagators: $P = \{\ldots\}$

d. Give the initial value of the queue $N$ of propagators not known to be at fixpoint: $N = [\ldots]$

e. Give the following information after every iteration of *propagate*:

> Picked propagator: ...
> Store: $s = \{x \mapsto \{\ldots\},\ y \mapsto \{\ldots\},\ \ldots\}$
> Status message: 'subsumed', 'at fixpoint', or 'not known to be at fixpoint'
> Raised events: $\{\ldots\} \subseteq \{\mathit{any}(x), \mathit{fix}(x), \mathit{min}(x), \mathit{max}(x), \ldots\}$
> Set of dependent propagators: $DP = \{\ldots\}$
> $P = \{\ldots\}$
> $N = [\ldots]$

f. Is the CSP solved after the last iteration? Why?

g. What changes in your answers when instead achieving *domain consistency* for all the constraints? Why?

## Question 4: Stores and Propagators (2 points)

Perform the following tasks:

a. Consider the following constraint stores:

$$s_1 = \{x \mapsto \{1,2\}, y \mapsto \{2,3\}\}$$

$$s_2 = \{x \mapsto \{1\}, y \mapsto \{2,3\}\}$$

$$s_3 = \{x \mapsto \{4\}, y \mapsto \{2,3\}\}$$

$$s_4 = \{x \mapsto \{1,2,4\}, y \mapsto \{2,3,5\}\}$$

For all $1 \leq i \neq j \leq 4$, decide whether $s_i < s_j$. List all assignments $a$ such that $a \in s_1$.

b. Given $V = \{x, y\}$ and $U = \{0, \ldots, 5\}$, consider the function $p_=$ defined on stores $s : V \to 2^U$ as follows:

$$p_=(s) = \left\{ \begin{array}{l} x \mapsto s(x) \cap s(y) \\ y \mapsto s(x) \cap s(y) \end{array} \right\}$$

Argue that $p_=$ is a propagator (that is, prove that it is contracting and monotonic). Argue that $p_=$ implements the $x = y$ constraint.

c. Given $V = \{x, y\}$ and $U = \{0, \ldots, 10\}$, consider the function $p$ defined on stores $s : V \to 2^U$ as follows:

$$p(s) = \left\{ \begin{array}{l} x \mapsto \{a \mid a \in s(x) \wedge \exists b \in s(y) : b = 3 - a\} \\ y \mapsto \{b \mid b \in s(y) \wedge \exists a \in s(x) : a = 3 - b\} \end{array} \right\}$$

Argue that $p$ is a propagator. What constraint does $p$ implement? Why?