KTH Information and
Communication Technology

Constraint Programming (ID2204), Spring 2015
Christian Schulte

Assignment 1

The different tasks on this assignment are marked with either **EXP** for an *exploration* task or **SUB** for a *submission* task. Only submission tasks are to be submitted. Submission for all these tasks is by Email to `cschulte@kth.se`:

- Use `ID2204: A1` as Email subject.
- Clearly state in the Email who you are (that is, name and personnummer). Do that for all group members.
- Make sure that all group members are also receivers of the Email (via cc). By that we can simply reply to the Email and everybody gets our comments.
- Submission deadline is **2015-04-13, 13:00**. Remember: you will get bonus points only if you submit in time.

More information you can find on the course's homepage.

1 Install Gecode (EXP)

Please install Gecode on your computer, instructions are on the Gecode webpage and in Chapter 2 of "Modeling and Programming with Gecode" (MPG).

2 Send More Money (EXP)

Try the Send More Money script from the second lecture. Use the programs in Chapters 2 and 3 of MPG.

3 Send Most Money (EXP)

Try the Send Most Money script from the second lecture. You can use the programs in Chapters 2 and 3 of MPG.

			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			

Figure 1: A Sudoku instance

4 Donald (EXP)

Give a script which solves the following crypto-arithmetic puzzle: find distinct digits for the letters such that

$$DONALD + GERALD = ROBERT$$

Can you find a better heuristic than first-fail?

5 Sudoku (SUB, 2 points)

Write a program to solve Sudoku puzzles.
A Sudoku puzzle consists of a 9 by 9 array of squares. Each square should contain one of the digits $1, 2, \ldots, 9$. The rules of the puzzle are that in a solution, each row should contain all the different digits, each column should contain all the different digits, and each major 3 by 3 block should contain all the different digits. An instance of a Sudoku puzzle is a square with some of the digits pre-filled. A valid Sudoku instance will have exactly one solution. In Figure 1 a Sudoku instance is shown. The major blocks are marked by having double-lines between them.
Write a program that solves Sudoku instances using Gecode.
Experiment with different heuristics and propagator strengths. You can change the propagator strength of distinct, by giving one `ICL_DEF`, `ICL_VAL`, `ICL_BND`, and `ICL_DOM` as the last argument. See if there is any difference in the sizes of the search-trees.
On the course's webpage, you will find a file containing several Sudoku instances, represented as 9 by 9 arrays of integers. Each non-zero integer represents a pre-filled square. Your program should be able to solve theese instances.

6 n-Queens (EXP)

In the examples folder of Gecode, there is a model for the n-Queens problem, in the file `queens.cpp`. This model uses the standard first-fail heuristic that has been presented in the lectures.
Try to find a good heuristic for this problem: it is possible to be better for many n than first-fail! Potential candidates are:

- Select a middle value.
- Select the variable with the smallest minimal value: this is also known as knight-move heuristic. Can you find out why?

To find out what strategies are available, read Section 9.1 in "Modeling and Programming with Gecode".

7 n-Queens With 0/1 Variables (SUB, 3 points)

In Lecture 2 you have seen that a different constraint model for n-Queens uses a matrix of $n \times n$ variables taking either 0 or 1 as value.
Modify the example model `queens.cpp` that comes with Gecode so that it uses this constraint model instead.

- Specify the constraints in this model.
- Represent the $n \times n$ matrix by a `Matrix` (read Section 7.2 in "Modeling and Programming with Gecode") of width n and height n.
- What can you do about branching?

List advantages and disadvantages of this model compared to the "standard" model as used in the lecture.

8 Is Propagation Compositional? (EXP)

Assume that `A`, `B`, `C`, and `X` are finite domain variables and that you are given the following statement:

```
A+B+C+B = X
```

It looks intuitive that you can transform this into

```
IntVar U
A+B = U
U+B+C = X
```

without changing anything. Now the question is: what does anything exactly mean?
Proceed as follows:

- Write two scripts `S1` and `S2` containing the statements together with the necessary declarations such that the variables take values between 1 and 5.
- Give a program which tests whether both scripts have the same solutions.
- Is there any other aspect in which they compute differently? In which aspect? Try to explain what happens (hint: think about using Gist).