Constraint Programming: Assignment no. 7/8

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

This week's assignment is about scheduling, rostering (using the regular constraint), and SAT solving.

Exercise 7/8.1 (Nurse rostering)

In Lecture 6, you have seen how to use the *regular* constraint for rostering applications. The Gecode/J library provides a class for specifying regular expressions, and a propagator for the *regular* constraint.

a) Implement a script that constrains an array of 7 variables with the regular expression $mm^*eee^*(ffm + mff)$ (similar to the expression from the lecture).
b) The solutions you get are not cyclic. For instance, $eeeffmmm$ is not a valid solution. Use the propagator for *regular* to implement a cyclic roster. Hint: post the constraint on an array of size 14. The regular expression needs slight modification, too.
c) Explain why the search does not produce any failures in exercise a), and why b) cannot be solved without failures.
d) Implement a roster for four nurses. All individual rosters should satisfy the constraint from b) (i.e., he cyclic version). Each day, at least one nurse is required for the morning shift and exactly one for the evening shift, and at least one nurse should have a day off. Hint: have a look at the `count` constraints.

Exercise 7/8.2 (Propagating *regular*)

Consider a model with an array of 5 variables and a regular constraint with the following regular expression: $((12)^* + (31)^*)44^*(1 + (24)^*)^*$.

a) Give a DFA for the regular expression.
b) Draw the layered graph for the following variable domains: $x[i] = \{0, 1, 2, 3, 4, 5\}$ for $0 \leq i \leq 4$.
c) Simulate propagation using pencil and paper. In particular, draw the layered graph after propagation is complete, and give the resulting variable domains.
d) How can you use propagation to optimize the DFA?

(2, 1)	(0, 3)	(1, 6)	(3, 7)	(5, 3)	(4, 6)
(1, 8)	(2, 5)	(4,10)	(5,10)	(0,10)	(3, 4)
(2, 5)	(3, 4)	(5, 8)	(0, 9)	(1, 1)	(4, 7)
(1, 5)	(0, 5)	(2, 5)	(3, 3)	(4, 8)	(5, 9)
(2, 9)	(1, 3)	(4, 5)	(5, 4)	(0, 3)	(3, 1)
(1, 3)	(3, 3)	(5, 9)	(0,10)	(4, 4)	(2, 1)

Table 1: A 6×6 job-shop problem

Exercise 7/8.3 (Job-shop)

A job-shop problem of size $m \times n$ is a set of m jobs, with each job consisting of n tasks with fixed duration. All tasks in one job must be performed in the given order. All tasks in one job are performed on different machines. No two tasks from different jobs, which require the same machine, can run in parallel. Find a minimal completion time for the whole job-shop, i.e. minimize the maximum of the end times of the individual jobs.

Table 1 shows a specification for a 6×6 job-shop problem. Each row represents one job. Each task in a job is encoded as a pair $(\text{machine}, \text{duration})$. Write a script using Gecode/J that computes the minimal completion time.

Exercise 7/8.4 (A clause propagator for Gecode)

Write a propagator in Gecode/J that implements a Boolean clause. The propagator should be posted with an array of `BoolVars` v and an array of booleans p (of the same size as v). For $v[i]$, the boolean $p[i]$ determines the *polarity* of the literal.

For example, given `BoolVars` x, y, z, you could post the clause $x \vee \neg y \vee \neg z$ as

```
VarArray<BoolVar> bs = new VarArray<BoolVar>(x,y,z);
boolean ps[] = {true, false, false};
Clause.post(this, bs, ps);
```

a) Implement a simple version of the propagator which subscribes to all variables.
b) Implement the propagator using *watched literals*: at every time, only subscribe to two variables.

Note that the watched literals we implement here are different from what you saw in the lecture, as they do not survive a backtrack. Still, it is beneficial to implement Boolean propagators like this.

`BoolVar` is a subclass of `IntVar`, so you can just use `IntVarViews` in your propagator. The variables follow the usual convention that 0=false and 1=true.

Exercise 7/8.5 (Conflict clause learning)

We want to decide satisfiability of the following set of clauses:

$$\begin{aligned}
\omega_1 &= x_4 \vee x_5 \vee x_7 \\
\omega_2 &= x_5 \vee x_6 \\
\omega_3 &= \neg x_7 \vee x_8 \\
\omega_4 &= \neg x_6 \vee x_8 \\
\omega_5 &= \neg x_8 \vee x_1 \vee x_{10} \\
\omega_6 &= \neg x_8 \vee x_2 \vee x_9 \\
\omega_7 &= x_3 \vee \neg x_9 \vee \neg x_{10} \\
\omega_8 &= x_6 \vee x_8
\end{aligned}$$

a) Perform search up to the first conflict with pencil and paper, using the following variable order: $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}$ (always pick false first, then true).
b) Draw the implication graph for the first conflict. (The first conflict happens after assigning x_5 to false.)
c) Determine the first unique implication point (1UIP) and the corresponding learned clause.
d) Backtrack and perform propagation again. Your conflict clause from exercise c) should cause failure immediately. Determine the new conflict clause. How far can you backjump according to the new conflict clause?