Topics in Constraint Programming: Exercises

K. Kuchcinski, J. Małuszyński, U. Nilsson

October 22, 1999

README.FIRST

The objective of the attached exercises is to give some practical experience with:

- the constraint solvers of Oz/Mozart
- the constraint handling rules (in SICStus Prolog),
- the interval CLP in Prolog IV.

Hardcopy solutions to the exercises must be handed in to Kris Kuchcinski NO LATER THAN DECEMBER 6. The solutions must include justifications and explanations in form of comments, and documentation of test runs. The exercises should be solved individually, but you may discuss alternative solutions with one another. The final seminar will be devoted to presentation and dissemination of alternative solutions.

1. The Newspapers Problem (revisited)

There are four students, Algy, Bertie, Charlie, and Digby, who share a flat. Four newspapers are delivered to the house: the Financial Times, the Guardian, the Daily Express, and the Sun. Each of the students reads all of the newspapers, in particular order and for a specified amount of time (see below). Given that Algy gets up at 8:30, Bertie and Charlie at 8:45, and Digby at 9:30, what is the earliest that they can all set off for college?

	Algy		Bertie		Charlie		Digby	
1st	FT	60 min	Guardian	75 min	Express	`5 min`	Sun	90 min
2nd	Guardian	30 min	Express	3 min	Guardian	15 min	FT	1 min
3rd	Express	2 min	TF	25 min	FT	10 min	Guardian	1 min
4st	Sun	5 min	Sun	10 min	Sun	30 min	Express	1 min

NOTE: Formulate the problem using the Oz language and its finite domain constraints and find the solution. Please, note that both cumulative and serialization constraints are available in OZ (see, System Modules, Part II: Constraint Programming).

2. Partitioning of logic network graphs

Multi-output Boolean functions can be specified using logic network graphs. For example, a simple network with 5 inputs, 6 terms and 2 outputs is depicted in the Figure 1. Partitioning is an

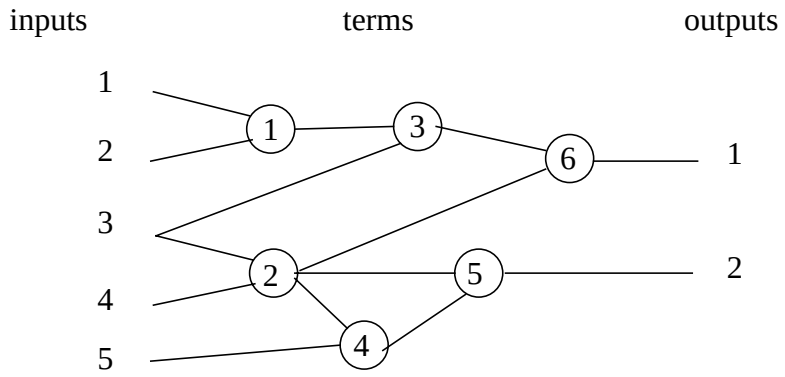

Figure 1. An example of a logic network graph.

important task which needs to be performed before implementation. The partitioning means that the network is decomposed into number of subnetworks which can be mapped into an available hardware fulfilling constraints on the number of inputs, outputs and terms allowed in each partition. All inputs and terms which contribute to the output assigned to a partition need to be assigned to the same partition. For example, the Figure 1 depicts a non-disjoint partitioning of the logic network graph from the Figure 1 which has at most 1 output in each partition. Note, that the input 3 and the term 2 belong to both partitions.

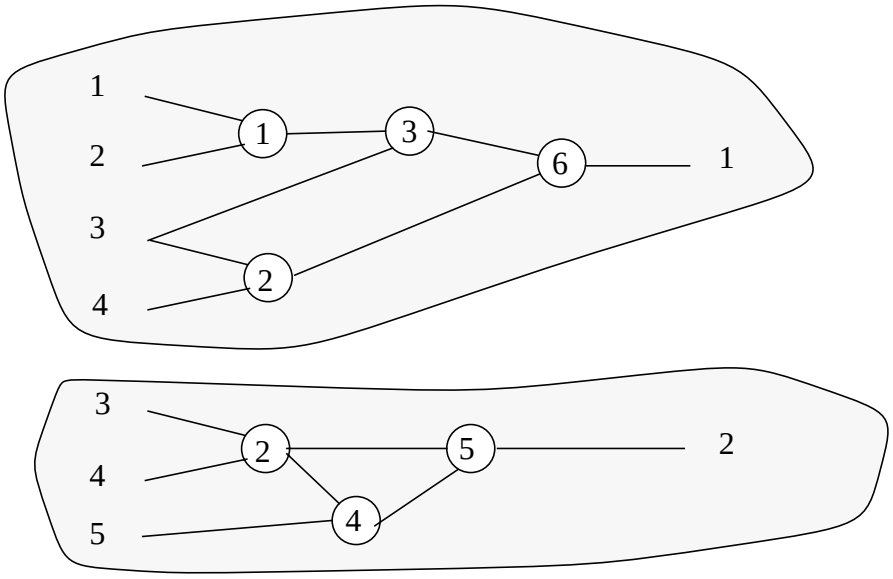

Figure 2. An example of a partitioning of the logic network graph from the Figure 1.

The goal of this assignment is to partition a logic network graph into two parts. Each part can not have more than 6 inputs, 6 outputs and 16 terms. The partitioning does not need to create disjoint sets of inputs or terms, i.e. the same term or input can belong to more than one partition. We are looking for one solution to this problem which satisfies input, output and term constraints for each partition, i.e., no optimization is required.
Solve this problem using finite set constraints offered by Oz. The Oz definition of the logic network graph to partition is given below. Each row represents one output specifying inputs and terms used to create it.

```
Def = def(1: output(inputs: [1 2 3 4 5 6]  terms: [1 4 5 6 7 8 9 10 15 16])
          2: output(inputs: [1 2 3 4 7 8]  terms: [15 16 17 19 21 22 24])
          3: output(inputs: [1 2 3 4 5 6]  terms: [1 6 15 16 ])
          4: output(inputs: [1 2 3 4 7 8]  terms: [15 16 18 20 21 22])
          5: output(inputs: [1 2 3 4 5 6]  terms: [1 3 4 5 11])
          6: output(inputs: [1 2 3 4 5 6]  terms: [3 4 5 11])
          7: output(inputs: [3 4 5 6]      terms: [2 12 13 14])
          8: output(inputs: [1 2 3 4 5 6]  terms: [2 4 5 7 9 10 12 13 15 16])
          9: output(inputs: [1 2 3 4 7 8]  terms: [15 16 17 20 21 23])
          10: output(inputs: [1 2 3 4 7 8] terms: [15 16 18 19 21 23 24])
          )
```

3. Constraint Handling Rules

This exercise is intended to familiarize you with Constraint Handling Rules (CHR), which is a general purpose language for writing constraint solvers. In this exercise you will implement a small solver for a minimal subset of finite domain constraints. The finite domain language that we consider here consists of three syntactic categories (apart from integers and variables); arithmetic *expressions*, domain *declarations* and *constraints* (which are either declarations or equalities):

$$\begin{aligned} E &\rightarrow Var \mid Int \mid E + E \\ D &\rightarrow Var \texttt{ in } Int \texttt{ .. } Int \\ C &\rightarrow D \mid E \texttt{ \#= } E \end{aligned}$$

Implement a solver using the CHR-library in SICStus Prolog. The solver doesn't have to be complete, however, it should be able to do the following (it is likely that the solver lists also other constraints in the answer):

```
| ?- X in 1..2, Y in 3..5, Z #= X+Y.

Z in 4..7,
X in 1..2,
Y in 3..5 ?
```

Moreover, propagation should work both ways:

```
| ?- X in 1..2, Z in 3..5, Z #= X+Y.

Z in 3..5,
X in 1..2,
Y in 1..4 ?
```

Then extend the solver with a labeling/0 procedure (which enumerates solutions to the constraints). For instance:

```
| ?- X in 1..2, Z in 3..5, Z #= X+Y, labeling.

X = 2,
Y = 1,
Z = 3,
labeling ?
```

Note The description in the article on labeling is outdated. Please see the examples in the CHR web site instead (e.g. the solver for booleans).

Hint 1 You probably have to use some of the built-in predicates of Prolog. The ones that are most likely to come to use are the arithmetic predicates, `ground`/1 (for checking is a term contains no variables) and `var`/1 (for checking if a term is an unbound variable). It is also likely that you have to implement a couple of "predefined" constraints in Prolog.

Hint 2 To facilitate the use of infix notation, include the following in the first part of your file:

```
:- use_module(library(chr)).

handler fd.

operator(700,xfx,(in)).
operator(600,xfx,(..)).
operator(700,xfx,(#=)).

constraints in/2, (#=)/2, labeling/0.
```

4. Modeling a nonlinear problem in Prolog IV

A ball is pushed at some start point with the initial speed Vb and rolls on the ground with deceleration $0.5\ m/sec^2$ until it stops. After $1\ sec$ another ball is thrown in the air from the same start point with the intention to hit the first ball while it is still rolling. Write a Prolog IV program that describes this problem. Use it to compute the initial speed vector of the second ball provided that $Vb = 5\ m/sec$ and the second ball hits the first one at $10m$ from the start point. Discuss what are other possible uses of your program.

5. Finding minima and maxima of the non-linear functions

Find global minimum and maximum of the following “six-hump camel-back function”:

$$f(X) = 4 \cdot {X_1}^2 - 2.1 \cdot {X_1}^4 + \frac{1}{3} \cdot {X_1}^6 + X_1 \cdot X_2 - 4 \cdot {X_2}^2 + 4 \cdot {X_2}^4$$

in the box X1=[-2.5, 2.5] and X2=[-2.5, 2.5].

NOTE: To solve this problem you need to implement a simple branch and bound algorithm which divides intervals and checks if there is a minimal or maximal solution in it. Checking for a solution can be done using interval splitting and the following Prolog construct which verifies the existence of the solution without binding variables:

verify(X) :- \+ (\+ (X)).

where X is a verified predicate.