The University of Melbourne
Department of Computing and Information Systems
Semester 2 Assessment 2015

COMP90046 Constraint Programming

Mid Semester Test: Sample Answers

Time Allowed: 1 Hour

Student Number:	

Question	Mark
1	
2	
3	
4	

Authorized materials: Books and calculators are not permitted.

Instructions to students:

This mid semester test counts for 0% of your final grade. It is used for self assessment, and for supporting a case for passing the subject if your assignment and final exam marks are borderline. There are 6 pages and 4 questions for a total of 12 marks. Attempt to answer all of the questions. Values are indicated for each question and subquestion — be careful to allocate your time according to the value of each question.

Question 1 [3 marks]

Here is an odd little puzzle which occurred the other day at an archery meeting. The young lady who carried off the first prize scored exactly one hundred points. The scores on the target are: 16, 17, 23, 24, 39 and 40. Can you figure out how many arrows she must have used to accomplish the feat?

Here is a MiniZinc model I wrote to determine the answer?

```
array[int] of int: scores;                                              %%% (1)
int: target;                                                            %%% (2)
int: maxarrow = target div max(scores);                                 %%% (3)
array[index_set(scores)] of var 0..maxarrow: hits;                      %%% (4)
constraint sum(i in index_set(scores))(scores[i]*hits[i]) <= target;    %%% (5)
solve minimize sum(i in index_set(scores))(hits[i]);                    %%% (6)
scores = [16,17,23,24,39,40];                                           %%% (7)
target = 100;                                                           %%% (8)
```

(a) In my rush I made a few errors. Rewrite line (5) to correct the error there.

```
constraint sum(i in index_set(scores))(scores[i]*hits[i]) = target;  %%% (5)
```

[1 mark]

(b) After rewriting line (5) to be correct it gives me

```
=====UNSATISFIABLE=====
```

So there is another problem in one more line: write the correction including the line number

```
int: maxarrow = target div min(scores);                                 %%% (3)
```

[2 marks]

Question 2 [2 marks]

There are n members of the Combinatorics club and n positions in the executive committee: President, Secretary, Treasurer, and $n-3$ ordinary committee members.

```
int: n;
set of int: MEMBER = 1..n;
set of int: POSN = 1..n;
int: President = 1;
int: Secretary = 2;
int: Treasurer = 3;
```

Unfortunately there are certain disagreements amongst the club members

- Some club members wont serve as Secretary or Treasurer with a certain President

  `array[MEMBER] of set of MEMBER: notwithPresident;`

  The set notwithPresident[i] are members who refuse to serve if i is President.
- Some club members wont serve as President or Treasurer with a certain Secretary

  `array[MEMBER] of set of MEMBER: notwithSecretary;`

  The set notwithSecretary[i] are members who refuse to serve if i is Secretary.

(a) What are the decisions variables you would use to define this model, explain why?

```
int: Other = 4;
set of int: POSTYPE = 1..4;
array[MEMBER] of var POSTYPE: position
```

or

```
var MEMBER: Pres;
var MEMBER: Sec;
var MEMBER: Treas;
```

the explanation why is that it is easy to encode the disagreement constraints. [1 mark].

(b) What is a global constraint you might use in the model, explain why?
Best answer is `global_cardinality` if using the first decisions.
You could use `alldifferent` if you used the second set of decisions. [1 mark].

Question 3 [3 marks]

Write down the expected output from the following models, running with all solutions flag checked.

(a)
```
var  0..4: x;
var -1..2: y;
constraint x div y >= 2;
solve satisfy;
```

```
x = 2;
y = 1;
----------
x = 3;
y = 1;
----------
x = 4;
y = 1;
----------
x = 4;
y = 2;
----------
==========
```

[1 mark].

(b)
```
var  0..4: x;
array[1..3] of int: a = [2,0,1];
constraint forall(i in 0..4)(x >= a[i]);
solve satisfy;
```

```
=====UNSATISFIABLE=====
```

[1 mark].

(c)
```
var  0..4: x;
var -1..2: y;
array[1..3] of int: a = [2,0,1];
constraint y div a[x] >= 2;
solve satisfy;
```

```
x = 3;
y = 2;
----------
==========
```

[1 mark].

Question 4 [4 marks]

A Magic Killer Square is an $n \times n$ square filled with the numbers $1..n \times n$ so that

- each square has a different number in it,
- the sum of each row and column is the same
- the sum of elements in a group add up to a given sum, except for group 0

Data files for the Magic Killer Square problem are of the form

```
n = size of square;
g = number of groups;
group = array[1..n,1..n] of 0..g defining the group for each square;
s = array[1..g] of int defining the sum for each group (except 0);
```

For example a Magic Killer Square of size 3 with 2 groups is given by

```
n = 3;
g = 2;
group = [| 0, 1, 1 | 0, 2, 0 | 1, 2, 0 |];
s = [ 17, 8 ];
```

a solution to this problem is given by

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

Note how each row and column adds up to 15, and the groups sum correctly.

Build a MiniZinc model for the Magic Killer Square Problem

```
int: n;
set of int: ROW = 1..n;
set of int: COL = 1..n;
set of int: NUM = 1..n*n;
int: g;
set of int: GROUP = 1..g;
array[ROW,COL] of GROUP: group;
array[GROUP of int: s;

array[ROW,COL] of var NUM: x;
var 0..sum([i | i in NUM]): ss;
%% BETTER int: ss = n*(n*n+1) div 2;
```

```
constraint alldifferent([ x[r,c] | r in ROW, c in COL ]);

constraint forall(r in ROW)
                 (sum(c in COL)(x[r,c]) = ss);
constraint forall(c in COL)
                 (sum(r in ROW)(x[r,c]) = ss);

constraint forall(g in GROUP)
                 (sum(r in ROW, c in COL where group[r,c] = g)
                     (x[r,c]) = s[g]);

solve satisfy;
```