The University of Melbourne
Department of Computing and Information Systems
Semester 2 Assessment 2012

COMP90046 Constraint Programming

Time Allowed: 3 Hours
Reading Time: 15 minutes

Authorized materials: Books and calculators are not permitted.

Instructions to Invigilators: One 21 page script. Exam paper may leave the room.

Instructions to students:

This exam counts for 70% of your final grade. There are 8 pages and 8 questions for a total of 70 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 [7 marks]

Consider the following MiniZinc predicate:

```
predicate strange(array[int] of var int:s,
                  array[int] of int:d, var int:e) =
          forall(i in index_set(s))(
                if i != max(index_set(s)) then
                   s[i] + d[i] <= s[i+1]
                else true endif
                /\ s[i] + d[i] <= e );
```

(a) Show the conjunction of primitive constraints of the form $a + b \leq c$ from unrolling the MiniZinc constraint

```
constraint strange([x,y,z],[1,3,2],t);
```

[4 marks].

$x + 1 \leq y$, $y + 3 \leq z$, $x + 1 \leq t$, $y + 3 \leq t$, $z + 2 \leq t$

(b) The definition of `strange` is not as simple as it could be and generates redundant constraints. Give a new MiniZinc definition of `strange` that is as efficient as possible but still has the same logical meaning. [3 marks].

```
predicate strange(array[int] of var int:s,
                  array[int] of int:d, var int:e) =
          forall(i in index_set(s))(
                if i < max(index_set(s)) then
                   s[i] + d[i] <= s[i+1]
                else
                   s[i] + d[i] <= e endif );
```

Question 2 [7 marks]

Alex, Brook, Cody, Dusty, and Erin recently found out that all of their birthdays were on the same day, though they are different ages.

On their mutual birthday, they were jabbering away, flapping their gums about their recent discovery. And, lucky me, I was there. Some of the things that I overheard were...

- Dusty said to Brook: "I'm nine years older than Erin."

- Erin said to Brook: "I'm seven years older than Alex."
- Alex said to Brook: "Your age is exactly 70% greater than mine."
- Brook said to Cody: "Erin is younger than you."
- Cody said to Dusty: "The difference between our ages is six years."
- Cody said to Alex: "I'm ten years older than you."
- Cody said to Alex: "Brook is younger than Dusty."
- Brook said to Cody: "The difference between your age and Dusty's is the same as the difference between Dusty's and Erin's."

Since I knew these people – and how old they were, I knew that they were not telling the whole truth.

After thinking about it, I realized that when one of them spoke to someone older, everything they said was true, but when speaking to someone younger, everything they said was false.

Write a MiniZinc model to determine the ages of each each person assuming they are integers in the range 5..100.

```
var 5..100:  A;
var 5..100:  B;
var 5..100:  C;
var 5..100:  D;
var 5..100:  E;
constraint alldifferent([A,B,C,D,E]);
constraint D < B <-> D = E + 9;
constraint E < B <-> E = A + 7;
constraint A < B <-> 10 * B = 17 * A;
constraint B < C <-> E < C;
constraint C < D <-> abs(C - D) = 6;
constraint C < A <-> A = 10 - C;
constraint C < A <-> B < D;
constraint B < C <-> abs(C - D) = abs(D - E);

solve satisfy;
```

Question 3 [8 marks]

(a) Explain what it means for a propagator to be idempotent. Given an example of a non-idempotent propagator. [3 marks].

An idempotent propagator f is such that $f(D) = f(f(D)$ for all domains D, that is the propagator will always return a domain which is a fixpoint for itself. An example of a non-idempotent propagator is the bounds propagator for $2x = 3y$. On domains $D(x) = [0..11]$, $D(y) = [0..11]$ it returns the $f(D)(x) = [0..11]$, $f(D)(y) = [0..7]$, but $f(f(D)(x) = [0..10]$.

(b) Give pseudo-code defining a bounds propagator for the constraint:

$$x = abs(y - z)$$

where x, y and z are integers. The meaning of the constraint should be clear, it has solutions $\{x \mapsto 1, y \mapsto 4, z \mapsto 3\}$ and $\{x \mapsto 4, y \mapsto 4, z \mapsto 0\}$ among many others. Make the propagator as strong as possible. Use functions $\mathsf{lb}(v)$ and $\mathsf{ub}(v)$ to access the current bounds of variable v, and functions $\mathsf{setlb}(v, d)$ and $\mathsf{setub}(v, d)$ to set the bounds of a variable. You can assume that the setting functions won't change to a weaker bound: e.g. $\mathsf{setlb}(x,\mathsf{lb}(x) - 1)$ will have no effect. [5 marks].

Its equivalent to the propagators for $x = abs(v), v = y - z$.
The first propagator is
if $lb(v) > 0 \vee ub(v) < 0$ setlb(x,max(lb(v),-ub(v)));
setub(x, max(-lb(v),ub(v));
if $lb(v) > 0$ { setlb(v, lb(x)); setub(v,ub(x)); };
if $ub(v) < 0$ { setlb(v, -ub(x)); setub(v,-lb(x)); };
setub(v,max(ub(x),-lb(x));
setlb(v,min(lb(x),-ub(x));

The second propagator is
setlb(v, lb(y) - ub(z));
setub(v, ub(y) - lb(z));
setlb(y, lb(v) + lb(z));
setub(y, ub(v) + ub(z));
setlb(z, lb(y) - ub(v));
setub(z, ub(y) - lb(v));

Question 4 [6 marks]

The global cardinality constraint, `disjoint`, has signature

```
disjoint(array[int] of var int:  x, array[int] of int:  y)
```

It constrains that the values taken by variables in x and the values taken by variables in y are disjoint. For example the constraints

`disjoint`$([1, 2, 2, 1, 2, 3, 5, 1], [4, 6, 4, 6])$
`disjoint`$([1, 2, 2, 1, 2, 2, 4, 1], [3, 3, 3, 3])$

both hold while

$\texttt{disjoint}([1,2,2,1,2,3,5,1],[4,6,2,4])$

does not.

(a) Give a predicate defining the global cardinality constraint `disjoint` by decomposition. [3 marks].

```
predicate disjoint(array[int] of var int:  x, array[int] of int:  y)
                    forall(i in index_set(x), j in index_set(y))
                          (x[i] != y[j]);
```

(b) The constraint problem

$\texttt{disjoint}([x_1,x_2,x_3,x_4],[y_1,y_2,y_3,y_4]) \wedge x_1+x_2+x_3+x_4+y_1+y_2+y_3+y_4 \leq 8$
$\wedge\ abs(y_1-x_1)=1 \wedge abs(y_2-x_2)=1 \wedge abs(y_4-x_3)=1 \wedge abs(y_4-x_4)=1$

has a number of symmetrys. Write down the symmetrys of the problem, and write new constraints to be added to the model to break as many symmetries as possible. [4 marks].

> The only symmetry is (x_1,x_2,y_1,y_2) can be transposed with (x_2,x_1,y_2,y_1). A symmetry breaking constraint to improve this would be $x_1 \leq x_2$. A better one would be $\texttt{lex\_lesseq}([x_1,x_2,y_1,y_2],[x_2,x_1,y_2,y_1])$.

Question 5 [10 marks]

(a) Given the following basic feasible solved form, show the result of a single pivot moving towards the optimal solution:

$$\begin{array}{lrcrcrcrl} \text{maximize} & z & = & 0 & - & 0.5C & + & 0.5E & \text{subject to} \\ & B & = & 3 & - & 0.5C & - & 0.5E & \wedge \\ & D & = & 2 & & & - & E & \wedge \\ & A & = & 4 & & & - & E & \wedge \\ & A,B,C,D,E & \geq & 0 & & & & & \end{array}$$

[3 marks].

(b) Given a system of linear constraints $x+y \geq 3 \wedge 3x+2y \leq 4 \wedge x-2y \geq 5$, where x and y are non-negative, write down an initial system in basic feasible solved form that could be the starting point for the 2-phase Simplex method. [3 marks].

(c) Briefly explain how the network simplex algorithm finds an initial basic feasible tree. [2 marks].

(d) Briefly explain the difference between the primal and dual Simplex methods. Explain why dual Simplex is preferable for use in a mixed integer programming solver. [2 marks].

Question 6 [13 marks]

A stock maintenance problem for an abattoirs is defined as follows. Each month there is demand for meat (in whole cows). The abattoir can meet the demand by buying cows or using those kept in its stockyard. The price for cows varies with each month, and there is a limit on availability each month, and additionally the regulations require that the abottoir cannot buy cattle on any two consecutive months. There is cost to keep cattle in the stockyard, and a maximum capacity.

Data for the problem is as follows:

```
int:  months;                      % number of months planning
array[1..months] of int:  demand; % meat demand on each month
int:  start;                       % starting stock of cattle
array[1..months] of int:  price; % price for one cow each month
array[1..months] of int:  avail; % availability of cows each month
int:  holdcost;                    % cost to hold a cow in stockyard
int:  capacity;                    % capacity of stockyard
```

For example the data file:

```
months = 8;
demand = [6,4,2,1,3,5,7,5];
start = 7;
price = [8,3,3,3,4,8,6,9];
avail = [10,10,10,10,10,10,10,10];
holdcost = 1;
capacity = 20;
```

describes a situation where there is an 8 month plan, with varying demands, an initial stock of 7, prices per cow variying, with 10 available at each time period, the cost to hold stock is 1, and the stockyard capacity is 20.

Expected output is a plan of the form:

```
cost = 137
buy = [0, 6, 0, 0, 10, 0, 10, 0]
stock = [1, 3, 1, 0, 7, 2, 5, 0]
```

showing the cost of the plam, the number of cows bought in each month, and the stock at the end of each month.

(a) Give a MiniZinc model for the abattoir problem. You do not need to repeat the data declarations. Ensure that each integer variable is given a tight range of possibilities. Ensure the output item returns output in the correct form. [8 marks].

```
array[0..months] of var 0..capacity: stock;
array[1..months] of var 0..capacity: buy;

constraint stock[0] = start;
constraint forall(i in 1..months)(
          buy[i] <= avail[i] /\
  stock[i] = stock[i-1] + buy[i] - demand[i]
   );
constraint forall(i in 1..months-1)(buy[i] = 0 \/ buy[i+1] = 0);

solve :: int_search(buy,input_order,indomain_min,complete)
      minimize cost;

var int: cost = sum(i in 1..months)(price[i] * buy[i] + holdcost * stock[i]);
```

(b) Give a good search annotation for the model. Explain why you think this is likely to give good search behaviour. [2 marks].

> By searching on *buy* with minimum values we are trying to create low cost solutions. By searching in order, the solver will be able to fix all the stock variables using the equations

(c) Discuss which solver would be best for solving this problem: finite domain, mixed integer programming, Boolean satisfiability or lazy clause generation. Argue why your chosen solver is likely to be preferable. [3 marks].

> The constraints are all linear except the disjunctive constraint, and this linearizes well, so the MIPsolver will be best.

Question 7 [8 marks]

(a) Explain in a paragraph how clausal resolution works, give an example resolution step. [2 marks].

(b) Give a BDD that enocdes the cardinality constraint $x_1 + x_2 + x_3 + x_4 \leq 2$. [3 marks].

(c) There are at least 3 different ways to model a integer constraint in a SAT solver using clauses: BDD, Binary and Unary. In a paragraph define the three different approaches and discuss the advantages and disadvantages of each. [3 marks].

Question 8 [11 marks]

Given a problem with bounds consistent propagators for the constraints: $x \neq y$, $x \neq z$, $y \neq z$, $x \leq u$, $y \leq u$, $z \leq u$, where the initial domain of each of the variables x, y, z and u is [1..4]. Assume the search strategy makes decisions in the order: $u \leq 2$, $x \leq 1$, and $y \leq 2$.

(a) Show the result of bounds propagation using the propagators: initially, and after every addition of a new constraint by the search. Assume that at each stage in the search the propagation loop continually examines all propagators in the order shown until a fixpoint is reached. Show the domains of all variables after any change and the propagator which caused it. [4 marks].

c	x	y	z	u
	1..4	1..4	1..4	1..4
$u \leq 2$				1..2
$x \leq u$	1..2			
$y \leq u$		1..2		
$z \leq u$			1..2	
$x \leq 1$	1			
$x \neq y$		2		
$x \neq z$			2	
$y \neq z$		$\emptyset$	$\emptyset$	

(b) In a lazy clause generation solver propagation is recorded in an implication graph. Give the implication graph for the problem above, and determine the 1UIP nogood that results. Explain where execution backtracks to, and show the result of propagation using the new nogood. [5 marks].

(c) Briefly discuss how a lazy clause generation solver will differ in its computation if the constraints $x \neq y$, $x \neq z$, $y \neq z$ are replaced by `alldifferent`$([x_1, x_2, x_3])$ assuming a domain consistent propagator. [2 marks].

The alldifferent constraint will detect failure after the first decision, examining the domains 1..2 for x, y and z. The lazy clause generation solver will learn a nogood $u \leq 2 \rightarrow false$ or $u \geq 3$.