Elementary Minizinc for Mathematics and Puzzles

Math problems and puzzles solved with minizinc

Charles Joscelyne

October 15, 2020

Preface

Minizinc *https://www.minizinc.org/* is a free and open source constraint solving programming language. Concisely put, minizinc allows users to model a quantitative problem as a collection of constraints expressed in terms of modeler defined decision variables. Once the constraints have been defined, minizinc calls on its library of pre-programmed solvers to find one or more solutions of the variables satisfying the constraints. Being more akin to a functional programming language than an imperative one, minizinc transfers the burden of writing a specific combinatorial or optimization algorithm imperatively to that of expressing a given problem functionally in terms of minzinc language constructs.

By way of example let us consider problem 13 from the 2019 American Mathematics Competition 8 Examination. *https://artofproblemsolving.com/wiki/index.php/2019_AMC_8_Problems*

A palindrome is a number that has the same value when read from left to right or from right to left. (For example, 12321 is a palindrome.) Let N be the least three-digit integer which is not a palindrome, but which is the sum of three distinct two-digit palindromes. What is the sum of the digits of N?

```
%Decision variables
var 1..9:N1;
var 0..9:N2;
var 0..9:N3;

var 1..9:A;
var 1..9:B;
var 1..9:C;


%Constraints variables
constraint N1 != N3;
constraint A != B;
constraint A != C;
constraint B != C;
constraint 100*N1 + 10*N2 + N3 = 10*A+A+10*B+B+10*C+C;
```

```
%solve
solve minimize 100*N1 + 10*N2 + N3;


%Display solution
output [ "\(N1)"++" "++"\(N2)"++" "++"\(N3)"++" sum of digits = "+.
+"\(N1+N2+N3)"];

% 1 1 0 sum of digits = 2
```

The solution: 110 = 77 + 22 + 11 to the palindrome problem is referenced as a minzinc comment in the program's last line.

A cursory glance of the program above shows the three components of any elementary minizinc model:

1. Definition of the model's decision variable(s).
2. Definition of model's decision constraints.
3. A solve statement.

Although, an output statement is included in above program, minizinc will usually display useful results in the absence of one. Nonetheless, a minizinc modeler most likely will want to control the contents and format of any output. Minizinc's output statement enables a modeler to do such.

Before proceeding to the study of individual problems and their solution in minizinc, I should note this tract is not entirely intended to be tutorial on minizinc. Although, I expect the reader would be able to learn various features of the minizinc language here, the language coverage is far from exhaustive. For those new to minizinc, Coursera offers an introductory course that can be found here: https://www.coursera.org/learn/basic-modeling. *A* rather concise, but nonetheless useful tutorial can be found here: https://www.minizinc.org/tutorial/minizinc-tute.pdf. My main aim here is rather to solve what I view to be interesting problems, employing where possible, only the more elementary features of minizinc. A challenge one faces when learning or applying any programming language is to have an ample supply of accessible code samples upon which to study and build upon. Code samples, I believe, are even more vital when programming in a functional styled language like minzinc. Whereas no more than few examples of loop constructs or conditional branch statements might suffice when learning to program in Java or Python, the same is not true when first approaching a functional or functional styled language like Haskell, R, or minzinc. Learning to program functionally is best supported through accessible and at the same time intriguing code samples. Unfortunately it has been my experience that many example minizinc models, by the complexity of the problem being addressed, tend to obscure the language features being employed. This makes it less likely for one new to the language to be successful later in applying what has been presented in a novel setting. Hopefully, the nature of the problems solved here in minizinc will serve as a compendium of readily accessible examples, which one can refer back to in order to adapt or extend.

In the following pages, when it is possible to do so, problems are categorized by the language features employed in its solution. This is opposed to the usual approach of grouping by problem content. In this way, a minizinc modeler when needing a particular language feature will be able to index relevant solved examples. This approach of course is the one familiarly taken by dictionaries and lexicons where sample sentences illustrate a word's usage. This tract's Contents section given below can be seen to be organized in this manner.

A constraint solver like minizinc is a powerful tool. As one becomes more proficient in its use, one's appreciation of scope and range of problems amenable to the language will only increase. Over time, It has been my experience that one adopts as a matter of course (perhaps unrealistically) the mindset that every problem encountered can be viewed as some type of constraint problem!

I closing I should add that when time and resources permits, I encourage the reader to attempt a solution to a problem before referencing the given solution. One should keep in mind that for almost all problems presented here, a purely mathematical solution exists outside of minizinc, which for the so inclined could be an interesting challenge to find.

Before proceeding to the problems and their solution in minizinc, I would note that the all minizinc code is linked to authors GitHub account, which can be found here https://github.com/pjoscely.

Contents

Modeling with Integer Variables

For our initial example, let us consider the following high algebra coin problem: https://github.com/pjoscely/Math-Comp-minizinc/blob/master/AMC%208/Alg%202%20H%20Coin%20Problem.mzn

```
A bag contains twice as many pennies as nickels and
four more dimes than quarters. Find all possibilities
for the number of each coin if their total value is $2.01.


%Coin variables
var 0..201:P;
var 0..41:N;
var 0..21:D;
var 0..8:Q;

%twice as many pennies as nickels
constraint P = 2*N;

%four more dimes than quarters
constraint D = Q + 4;

%total value is $2.01
constraint 201 = P + 5*N + 10*D + 25*Q;

%required solve statement
solve satisfy;

%display results
output["P = "++"\(P)"++" N = "++"\(N)"++" D = "++"\(D)"++" Q = "++"\(Q)"];

/*
P = 46 N = 23 D = 4 Q = 0
----------
P = 36 N = 18 D = 5 Q = 1
----------
P = 26 N = 13 D = 6 Q = 2
----------
P = 16 N = 8 D = 7 Q = 3
----------
P = 6 N = 3 D = 8 Q = 4
----------
==========
*/
```

The program's output is included as a comment after the code. Single line comments in minizinc begin with a "%", while multiline comments begin with "/*"

and end with “*/“. The double dashed line concluding the output’s indicates there are no more solutions. While the model is self-explanatory, it should be pointed out that the defined decision variables are declared with integer bounds in accordance with the problem. Doing so narrows the search and lessens the final number of constraint statements. More on this later.

problem 7 from the 2019 American Mathematics Competition 8 Examination.

https://github.com/pjoscely/Math-Comp-minizinc/blob/master/AMC%208/amc8_2019_7.mzn

**Shauna takes five tests, each worth a maximum of 100 points.
Her scores on the first three tests are 76, 94, and 87.
In order to average 81 for all five tests, what is the lowest score
she could earn on one of the other two tests?**

(A) 48 (B) 52 (C) 66 (D) 70 (E) 74

Minizinc Model

```
%test scores
par int: first = 76;
par int: second = 94;
par int: third = 87;
var int: fourth;
var int: fifth;

%bounds on possible test scores
constraint fourth >= 0;
constraint fifth >= 0;
constraint fourth < 101;
constraint fifth < 101;

%final average constraint
constraint (first + second + third + fourth + fifth)/5 = 81;

%minimize fourth test score
solve minimize fourth;

%display result
output [" lowest fourth score = "++"\(fourth)"]

 %lowest score = 48
```