Elementary MiniZinc for Mathematics and Puzzles

Charles Joscelyne

December 19. 2020

Preface

MiniZinc: *https://www.MiniZinc.org/* is a free and open source constraint solving programming language. Concisely put, MiniZinc allows a user 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 MiniZinc 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
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 MiniZinc 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 so.

Before proceeding to the study of individual problems and their solution in MiniZinc, I should note that this tract is not entirely intended to be a tutorial on MiniZinc. Although, I expect the reader would be able to learn many basic features of the MiniZinc language here, the coverage of the language 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* somewhat concise, but nonetheless useful tutorial can be found here: https://www.minizinc.org/tutorial/minizinc-tute.pdf. My aim here rather is to solve what I view to be interesting problems, employing where possible, only the more elementary constructs of MiniZinc. A challenge one faces when learning and subsequently using any programming language is to have an ample supply of accessible code samples upon which to study and apply. I believe code samples are even more vital when programming in a functional styled language like MiniZinc. 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 MiniZinc. Learning to program functionally is best supported through accessible and at the same time interesting code samples. Unfortunately, it has been my experience that many sample MiniZinc models, due to the complexity of the problem being addressed, tend to obscure the language features 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 to in order to adapt or extend.

In the following pages, problems are categorized by the language features employed in the problem's solution. This is opposed to a grouping by problem content. In this way, a MiniZinc modeler when needing a particular language feature will be able to more easily index one or more relevant solved examples. This approach, of course, is the one familiarly taken by many programming language expositions along with dictionaries and lexicons where sample sentences illustrate a term's usage. The contents section given below can then 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 being solved with the language will surely grow. In fact, one could possibly be lulled into adopting the unrealistic mindset that every combinatorial or optimization problem encountered can be viewed as a constraint problem solvable by MiniZinc. The possibility of acquiring such a narrow view should not raise alarm for the liberally educated who will surely consign MiniZinc to be just another problem solving tool in their kit.

When time and energy permits, I encourage the reader to attempt to solve a problem before referencing the MiniZinc solution. One should realize that for almost all the problems presented here, a purely mathematical solution exists, which for the so inclined can be of equal interest to discover. To help the reader to gain more experience using MiniZinc, a collection of related exercises and their solutions are presented at this tract's end.

Before proceeding to the problems and their solution in MiniZinc, I should point out that all the MiniZinc code presented here is linked to authors GitHub account, which can be found here: https://github.com/pjoscely.

Contents