UNIVERSIDAD DE GRANADA

Técnicas de los Sistemas Inteligentes
grado en ingeniería informática

Práctica 2

Satisfacción de Restricciones

Autor
José María Sánchez Guerrero

Rama
Computación y Sistemas Inteligentes || Grupo 2 - Pablo Mesejo

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2019-2020

Índice

Introducción

Esta práctica consiste en la resolución de una serie de ejercicios utilizando el lenguaje de modelado de restricciones ***MiniZinc***. En este programa, simplemente tendremos que escribir las variables y restricciones que se nos pidan en el ejercicio, y posteriormente ejecutarlo. Sin embargo, tendremos distintas configuraciones para el solucionador o "*solver*" a utilizar, y tras varias pruebas, el que mejor funcionaba es el "***Chuffed 0.10.4***".

Ejercicio 1

En este problema se nos pide codificar 9 letras con un dígito diferente para cada una (entre el 0 y el 9). Esta asignación de dígitos tiene que satisfacer la siguiente suma:

$$TESTE + FESTE + DEINE = KRAFTE$$

Para resolverlo hemos utilizado dos *sets* de enteros, uno con el número de dígitos/letras que hay, y el otro con los posibles valores que pueden tomar. También hemos utilizado un array con los posibles valores para aplicarle las posteriores restricciones; junto a otro pero de string para poder imprimir las letras en el resultado (este último no intervendrá en las operaciones).

Las primera restricción necesaria ha sido la función "*all_ different()*", que servirá para que todos los valores que puede adoptar el array solución, sean distintos. La segunda restricción consiste en la propia suma, de forma que asigno a cada posición un valor de la suma, segmentando el número en unidades, decenas, centenas....

Finalmente, imprimo la solución gracias al array de string declarado anteriormente: **T=7 | E=0 | S=6 | F=8 | D=9 | I=5 | N=3 | K=2 | R=4 | A=1**

$$30.830 + 60.830 + 50970 = 142.630$$

Ejercicio 2

En este jercicio tenemos que encontrar un número de 10 cifras, en el cual se cumpla que el primer dígito sea el número de 0s que hay en el propio número, el segundo sea el número de 1s, (...), y el último sea el número de 9s.

Para ello hemos utilizado un array de enteros de tamaño 10 y que valores de cada posición tengan un valor entre el 0 y el 9. Para asignar este valor, utilizamos

la función "*forall()*" para recorrer todas las posiciones del array. En cada una de ellas, realizaremos un conteo de las todas las posiciones con la función "*count()*", y comprobamos que el número contado es igual al de la posición en el array.

Si ejecutamos, el resultado que obtenemos es el mismo que en el ejemplo: $\mathbf{X} =$ **6210001000**

Ejercicio 3

Tenemos que encontrar un horario para distribuir a 6 profesores en un aula durante 6 horas. Cada clase dura 1 hora, pero cada profesor tiene distintas restricciones en sus horarios. Se ha representado también como un array, donde cada posición representa a un profesor y sus valores serán las horas disponibles.

La primera restricción la vamos a solventar con la función "*all_ different()*" vista anteriormente, para que ningún profesor pueda estar en el aula a la misma hora que otro. Las siguientes líneas del código corresponderán con las expresiones lógicas que satisfacen los distintos horarios de los profesores.

El resultado final que obtendríamos es el siguiente:

Profesor 1 da 1h de clase a las 14:00h

Profesor 2 da 1h de clase a las 12:00h

Profesor 3 da 1h de clase a las 13:00h

Profesor 4 da 1h de clase a las 10:00h

Profesor 5 da 1h de clase a las 11:00h

Profesor 6 da 1h de clase a las 9:00h

Ejercicio 4

Este ejercicio también consiste en la creación de un horario para los profesores, pero esta vez con distintas condiciones. Dispondremos de 4 aulas, 4 profesores y 3 asignaturas con 4 grupos cada una, por lo que la representación en este ejercicio será distinta al anterior.

Utilizaremos una matriz donde las filas serán los periodos de tiempo, y las

columnas los profesores. Los valores que tomará esta matriz vienen determinados por otro array que representa a las distintas asignaturas con sus grupos correspondientes; sin embargo, como tenemos menos asignaturas que horas disponibles, he añadido 4 más, llamadas *'EMPTY'*, y serán horas libre para el profesor al que le corresponda.

También he implementado otra matriz igual que la anterior, pero que sus valores representan el grupo que está dando una clase. Gracias a esto, y a la funnción "*alldifferent()*", que en este caso la aplico por filas, evitamos que el mismo grupo este dando dos clases a la misma hora.

En cuanto a las restricciones, he utilizado una estructura similar a la del ejercicio anterior, con expresiones lógicas. Al tener dos matrices, vamos asignando asignatura y grupo paralelamente; y también he utilizado "*forall()*" para recorrerlas más fácil, por ejemplo:

```
forall(h in HOURS) ( (schedule[h,1] == 2 ∩ groups[h,1] == 2)
```

Para todas las horas, el profesor 1 puede cursar la asignatura 2 ('IA') en el grupo 2. Finalmente, el resultado de la matriz es el siguiente:

	1	**2**	**3**	**4**
9:00-10:00	IA-G1	FBD-G2	FBD-G3	Empty
10:00-11:00	TSI-G1	Empty	TSI-G3	IA-G4
11:00-12:00	TSI-G2	FBD-G1	FBD-G4	Empty
12:00-13:00	IA-G2	Empty	TSI-G4	IA-G3

Ejercicio 5

Ejercicio 6

Ejercicio 7

Ejercicio 8

Ejercicio 9

Ejercicio 10