UNIVERSIDAD DE GRANADA

Técnicas y Sistemas Inteligentes

Práctica 2

Programación de Restricciones con Minizinc

Autor: Francisco Javier Bolívar Expósito Grupo: 3 Curso: 2019-2020

1. Puzzle Cripto-aritmético

Nuestro objetivo es averiguar un conjunto de valores para las letras que verifique la suma.

La codificación usada consiste en una variable del 0 al 9 (dígito) por cada letra y un array de 5 componentes con el acarreo resultante de cada suma intermedia.

Las restricciones necesarias es que todas las variables de letras sean diferentes para que cada uno de ellas represente un dígito y que se cumpla la suma de cada columna, consistente en que la suma del acarreo de la suma anterior y la suma de las letras en la parte superior de la columna sean iguales a la letra resultado más diez por el acarreo de la suma actual.

```
A = 1;
D = 9;
E = 0;
F = 8;
I = 7;
K = 2;
N = 4;
R = 3;
S = 5;
T = 6;
acarreo = array1d(1..5, [0, 1, 1, 0, 2]);
----------
```

Resultado

2. Encontrar un número X de 10 dígitos que satisfaga que el primer dígito de X representa el número de 0s en X, el segundo dígito de X representa el número de 1s en X, etc.

X se codifica como un array de tamaño 10 en el que cada componente es un dígito de x, siendo la primera el primer dígito y la última el último.

Siendo el primer dígito X_0 y el último X_9 la restricción necesaria es: Para cada X_i el número de apariciones de i en X tiene que ser igual a X_i.

```
x = array1d(0..9, [6, 2, 1, 0, 0, 0, 1, 0, 0, 0]);
```

Resultado

3. Encontrar una asignación de horarios de profesores en una aula.

La codificación consiste en un vector de tamaño número de profesores en el que se guarda cuando da hora cada profesor. Simplemente guardamos la hora de inicio de 0 a 23, ya que todos los profesores dan clases de una hora, la solución 11 por ejemplo se interpretaría como 11:00-12:00.

Como restricción establecemos que las horas en las que dan clases los profesores tienen que ser diferentes y que la hora de cada profesor tiene que estar dentro de su horario.

```
prof = array1d(1..6, [14, 12, 13, 10, 11, 9]);
```

Resultado Minizinc

El horario obtenido quedaría de la siguiente manera.

	Aula 1
9:00 - 10:00	Profesor 6
10:00 - 11:00	Profesor 4
11:00 - 12:00	Profesor 5
12:00 - 13:00	Profesor 2
13:00 - 14:00	Profesor 3
14:00 - 15:00	Profesor 1

4. Encontrar una asignación de horarios de asignaturas según grupos de alumnos, horario de profesores y aulas disponibles.

Para resolver el ejercicio se ha querido obtener a que hora empieza cada clase cumpliendo las restricciones.

La codificación consiste en una matriz ‘aula_sel’ que para cada [asignatura, grupo] indica que aula se ha escogido para la clase y otra matriz ‘hora_sel’ de la misma forma pero que indica a que hora se da la clase. Las horas codificadas permiten valores desde las 9 hasta las 12, indicado la hora de inicio de la clase.

Un grupo de alumnos no puede tener clase en la misma hora, para cumplir esto establecemos que para cada columna de la matriz ‘hora_sel’ sus componentes tienen que ser todos diferentes.

Para cumplir que un profesor no puede dar varias clases a la vez establecemos que las horas de todas las clases que imparte deben ser diferentes.

Para cumplir el horario de los profesores establecemos que las horas de las clases que imparte no puede ocupar su horario no disponible.

Finalmente una asignatura no puede compartir el mismo lugar / horario que otra, para esto indicamos que cada asignatura tiene que tener una hora diferente para cada una del resto de asignaturas o impartirse en una clase diferente.

```
IA-G1: 10:00-11:00 A3
IA-G2: 12:00-13:00 A2
IA-G3: 11:00-12:00 A4
IA-G4: 10:00-11:00 A2
TSI-G1: 9:00-10:00 A3
TSI-G2: 11:00-12:00 A3
TSI-G3: 12:00-13:00 A1
TSI-G4: 11:00-12:00 A2
FBD-G1: 11:00-12:00 A1
FBD-G2: 9:00-10:00 A2
FBD-G3: 10:00-11:00 A1
FBD-G4: 9:00-10:00 A1
```

Resultado Minizinc

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

5. Encontrar una asignación de horarios de asignaturas de diferente duración, con horarios de los profesores, recreo, etc.

Para resolver este problema se ha seguido un método similar al ejercicio anterior.

Tenemos un array ‘dia_sel’ que indica que día de la semana se da cada clase y un array ‘hora_sel’ que indica a que hora.

Cada bloque de cada asignatura es una clase con su valor en estos vectores. El orden seguido se puede observar en el array ‘strAsig’, donde en $strAsig_i$ podemos ver de que asignatura es el bloque i.

strAsig = ["A1","A2","A3","A4","A5","A6","A7","A8","A9","A1","A2","A3","A4","A5","A6","A7"];

Los días pueden tomar un valor del 1 al 5, representado por orden del Lunes al Viernes.

Las horas de un bloque pueden tomar el valor 8, 9, 10, 12 o 13. Representando la hora de inicio de una clase, el valor 11 no se incluye ya que a esta hora se encuentra el recreo.

Con la codificación definida aplicamos las siguientes restricciones.

Para cada bloque cada uno del resto de bloques debe impartirse en una hora diferente o un día diferente, ya que no puede darse una clase a la misma hora el mismo día.

El valor de ‘dia_sel’ para un bloque no puede ser igual al de otro bloque que pertenezca a la misma asignatura, ya que dos bloques de una misma asignatura no pueden darse en el mismo día.

Para los bloque de 2 horas se restringe el conjunto de horas de inicio elegibles, no pudiendo establecer su hora a las 10 porque entonces ocuparía el recreo de las 11 ni a las 13 por salirse del horario de clases (ocuparía 13:00-15:00 y las clases finalizan a las 14:00). Además, para cada bloque de 2h cada uno del resto de bloques debe impartirse en un día diferente o tener su hora de inicio 2h después o tener una hora de inicio más pequeña que el bloque de 2h.

Finalmente establecemos que los dias seleccionados para los bloques que imparte un profesor (excepto el cuarto) no pueden ser el mismo.

```
dia_sel = array1d(1..16, [4, 1, 3, 2, 1, 3, 5, 1, 5, 2, 5, 5, 4, 3, 4, 2]);
hora_sel = array1d(1..16, [12, 10, 12, 12, 12, 10, 10, 8, 13, 8, 12, 8, 8, 8, 10, 10]);
```

Resultado Minizinc

	Lunes	Martes	Miércoles	Jueves	Viernes
8:00 - 9:00	A8	A1	A5	A4	A3
9:00 - 10:00	A8	A1	A5	A4	A3
10:00 - 11:00	A2	A7	A6	A6	A7
11:00 - 12:00	Recreo	Recreo	Recreo	Recreo	Recreo
12:00 - 13:00	A5	A4	A3	A1	A2
13:00 - 14:00	A5	A4	A3	A1	A9

6. ¿Dónde está la cebra y quién bebe agua?

Para lo codificación de este ejercicio se ha utilizado una matriz ‘calle’ 5x5. Cada fila corresponde a una casa de la calle, siendo la fila 1 la primera casa a la izquierda y 5 la última casa a la derecha. Cada columna corresponde a una característica, guardándose en la primera la región de la persona que vive en la casa, el color de la casa en la segunda, el animal en la 3a, la profesión en la 4a y finalmente que bebe en la 5a.

Los valores de cada característica van del 1 al 5 y están definidos de la siguiente forma:

Columna 1. 1 - Andaluz, 2 - Catalán, 3 - Gallego, 4 - Navarro, 5 - Vasco
Columna 2. 1 - Roja, 2 - Azul, 3 - Verde, 4 - Amarilla, 5 - blanca
Columna 3. 1 - Perro, 2 - Caracoles, 3 - Zorro, 4 - Caballo, 5 - Cebra
Columna 4. 1 - Pintor, 2 - Escultor, 3 - Diplomático, 4 - Violinista, 5 - Médico
Columna 5. 1 - Té, 2 - Café, 3 - Leche, 4 - Zumo, 5 – Agua

De restricciones primero establecemos que para cada columna todos sus valores deben ser diferentes, ya que el enunciado establece que las casas no comparten ninguna característica.

Tras esto para cada apartado del enunciado establecemos unas restricción. Hay fundamentalmente tres tipos de restricciones.

El primer tipo de restricción es que si una casa tiene una característica también tiene otra.
Veamos un ejemplo, 'El Vasco vive en la casa Roja', nos indica que si una casa tiene la columna 1 == 5 (Vasco) la columna 2 == 1 (Roja). Representamos esta restricción con 'constraint exists(i in ncasas) (calle[i, 1] == 5 /\ calle[i, 2] == 1)';

El segundo tipo es que la casa de una posición tiene una característica. Esta restricción es especialmente fácil de representar, solo tenemos que establecer la restricción de que calle[casa, tipocaracteristica] = característica. Por ejemplo 'En la casa central se bebe leche' lo representamos con 'constraint calle[3, 5] == 3;'

El último tipo es que una casa con una característica 'a' se encuentra al lado de otra con una característica 'b'. Establecemos que si una casa i tiene la característica 'a' además se tiene que cumplir que la casa i-1 o la i+1 cumpla la característica 'b'. Hay una variación en la que tiene que estar obligatoriamente a la derecha, por lo que i+1 tiene que cumplir obligatoriamente la característica 'b'.

```
Bebe agua el Andaluz
Cebra en casa 5 con el Gallego
```

Resultado Minizinc

7. Asignación de tiempos de inicio a tareas con duraciones y precedencia.

La representación usada es un array 'tiempo_inicio' y uno 'tiempo_fin', ambos de enteros, para los que $tiempo\_inicio_i$ y $tiempo\_fin_i$ representan el tiempo en el que se inicia y finaliza respectivamente la tarea i.

Guardamos en un array 'duracion' la duración de la tarea i en $duracion_i$.

De restricciones limitamos el rango de los tiempos a números iguales o mayores que 0, establecemos que $tiempo\_fin_i == tiempo\_inicio_i + duracion_i$ y finalmente tenemos que establecer las restricciones de precedencia.

Para representar que la tarea 2 va después de la 1 tenemos que establecer la restricción de que el tiempo de inicialización de la tarea 2 sea mayor o igual que el tiempo de finalización de la tarea 1.

Finalmente minimizamos el máximo valor de los tiempos de finalización.

```
tiempo_inicio = array1d(1..9, [0, 7, 10, 7, 15, 15, 15, 7, 16]);
tiempo_fin = array1d(1..9, [7, 10, 11, 15, 17, 16, 16, 10, 18]);
duracion_total = 18;
```

Resultado Minizinc

10. Problema de la mochila.

Guardamos los datos de los problemas en un array de pesos en el que en la posición i se indica el peso del objeto i y un array de preferecias en el que en la posición i se indica la preferencia del objeto i. Ambos tienen un tamaño igual al número de objetos.

Para codificar la solución tenemos un array binario con un tamaño igual al número de objetos. En el si la posición i tiene valor 0 significa que no guardamos en la mochila el objeto i y si tiene valor 1 que si lo guardamos.

Como restricciones tenemos que la suma del peso de los elementos seleccionados debe ser menor o igual al peso máximo (275). Queda programado en Minizinc con ‘constraint sum([pesos[i] * seleccion[i] | i in nobjetos]) <= 275;’ ya que si el objeto es seleccionado pesos[i] * seleccion[i] resultará en su peso, y si no es seleccionado al multiplicar por 0 no se tendrá en cuenta.

Finalmente indicamos que queremos maximizar la suma de la preferencia de los elementos seleccionados, programándolo en Minizinc de manera similar a la suma de pesos de elementos seleccionados pero cambiando el array de pesos por el array de preferencias.

```
seleccion = array1d(1..12, [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0]);
preferencia_obtenida = 705;
```

Resultado Minizinc

Obtenemos como resultado guardar en la mochila el Mapa, el Compás, el Agua, el Sandwich, el Azúcar, el Queso y el Protector Solar con una preferencia total de 705.