Programación III – Taller MiniZinc
David Fernando Benavides Calderón

1. Tres hermanos, Ana, Benito, y Carla, tienen edades diferentes. Carla es más joven que Benito. Ana no es la más joven. Carla no es la más vieja. Benito no es el más joven. ¿Cuál es la edad de cada uno?

SOLUCIÓN:

```
% Definimos las variables para las edades de Ana, Benito y Carla

var 1..3: Ana;

var 1..3: Benito;

var 1..3: Carla;


% Las edades son diferentes

constraint Ana != Benito /\ Ana != Carla /\ Benito != Carla;


% Carla es más joven que Benito

constraint Carla < Benito;


% Ana no es la más joven

constraint Ana != 1;


% Carla no es la más vieja

constraint Carla != 3;


% Benito no es el más joven

constraint Benito != 1;


% Solución

solve satisfy;


% Mostrar las edades

output ["Ana tiene: \(Ana)\n", "Benito tiene: \(Benito)\n", "Carla tiene: \(Carla)\n"];
```

2. En una fiesta de disfraces, cinco amigos se disfrazaron de diferentes animales: tigre, elefante, león, cebra y jirafa. A continuación, algunas pistas sobre quién se disfrazó de qué:

• Carlos no se disfrazó de tigre ni de jirafa.

• El león se disfrazó de María.

• Juan no se disfrazó de jirafa.

• Laura se disfrazó de elefante.

• El tigre no fue disfrazado por Pedro ni por María. ¿Quién se disfrazó de qué

animal?

SOLUCIÓN:

```
% Definimos el conjunto de amigos y disfraces

set of int: Personas = 1..5;

set of int: Animales = 1..5;


% Cada número representará un animal:

% 1: Tigre, 2: Elefante, 3: León, 4: Cebra, 5: Jirafa


% Variables que representan el disfraz de cada persona

var Animales: Carlos;

var Animales: Maria;

var Animales: Juan;

var Animales: Laura;

var Animales: Pedro;


% Restricción: Cada persona se disfraza de un animal diferente

constraint all_different([Carlos, Maria, Juan, Laura, Pedro]);


% Restricciones basadas en las pistas:


% Carlos no se disfrazó de tigre ni de jirafa

constraint Carlos != 1 /\ Carlos != 5;
```

```
% El león se disfrazó de María
constraint Maria = 3;

% Juan no se disfrazó de jirafa
constraint Juan != 5;

% Laura se disfrazó de elefante
constraint Laura = 2;

% El tigre no fue disfrazado por Pedro ni por María
constraint Pedro != 1 /\ Maria != 1;

% Solución
solve satisfy;

% Mostrar quién se disfrazó de qué
output [
  "Carlos se disfrazó de: ", show(Carlos), "\n",
  "Maria se disfrazó de: ", show(Maria), "\n",
  "Juan se disfrazó de: ", show(Juan), "\n",
  "Laura se disfrazó de: ", show(Laura), "\n",
  "Pedro se disfrazó de: ", show(Pedro), "\n"
];
```

---

3. Tienes tres vasos vacíos: uno de 8 litros, otro de 5 litros y otro de 3 litros. Puedes llenar los vasos o verter agua de un vaso a otro. El objetivo es medir exactamente 4 litros de agua en el vaso de 8 litros. ¿Cómo lo haces?

SOLUCIÓN:

```
% Definimos las capacidades de los vasos
int: max_v8 = 8;
int: max_v5 = 5;
int: max_v3 = 3;
```

```
% Variables que representan el contenido de los vasos después de cada paso
var 0..max_v8: v8;
var 0..max_v5: v5;
var 0..max_v3: v3;

% Inicializamos los vasos: todos vacíos
int: init_v8 = 0;
int: init_v5 = 0;
int: init_v3 = 0;

% Pasos del algoritmo para llegar a 4 litros en el vaso de 8 litros
constraint
  % Paso 1: Llenar el vaso de 5 litros
  init_v5 = 5 /\ init_v3 = 0 /\ init_v8 = 0 /\

  % Paso 2: Verter 5 litros en el vaso de 3 litros
  v5 = 2 /\ v3 = 3 /\ v8 = 0 /\

  % Paso 3: Vaciar el vaso de 3 litros
  v3 = 0 /\

  % Paso 4: Verter los 2 litros restantes del vaso de 5 litros en el vaso de 3 litros
  v5 = 0 /\ v3 = 2 /\ v8 = 0 /\

  % Paso 5: Llenar el vaso de 5 litros nuevamente
  v5 = 5 /\ v3 = 2 /\ v8 = 0 /\

  % Paso 6: Verter el vaso de 5 litros en el vaso de 3 litros hasta llenarlo (3 litros)
  v5 = 4 /\ v3 = 3 /\ v8 = 0 /\

  % Paso 7: Verter los 4 litros restantes en el vaso de 8 litros
```

```
   v8 = 4 /\ v5 = 0 /\ v3 = 3;

% Mostrar los valores finales de cada vaso
solve satisfy;

output [
   "Vaso de 8 litros contiene: \(v8) litros\n",
   "Vaso de 5 litros contiene: \(v5) litros\n",
   "Vaso de 3 litros contiene: \(v3) litros\n"
];
```

---

4. Cuatro amigos están hablando acerca de los días de la semana. Cada uno hace una afirmación:

• Antonio dice: "Hoy no es lunes ni martes."

• Beatriz dice: "Ayer era domingo."

• Carlos dice: "Mañana será jueves."

• Daniel dice: "Pasado mañana será sábado." ¿Qué día es hoy?

SOLUCIÓN:

```
% Definimos los días de la semana, donde 1 = Lunes, 2 = Martes, ..., 7 = Domingo
enum Dias = {Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo};

% Variable para representar el día de hoy
var Dias: Hoy;

% Restricciones basadas en las afirmaciones de los amigos

% Antonio dice: "Hoy no es lunes ni martes."
constraint Hoy != Lunes /\ Hoy != Martes;

% Beatriz dice: "Ayer era domingo."
% Si ayer era domingo, hoy es lunes
constraint (Hoy = Lunes) -> (Hoy = Lunes);
```

```
% Carlos dice: "Mañana será jueves."
% Si mañana es jueves, hoy es miércoles
constraint (Hoy = Miercoles) -> (Hoy = Miercoles);

% Daniel dice: "Pasado mañana será sábado."
% Si pasado mañana es sábado, hoy es jueves
constraint (Hoy = Jueves) -> (Hoy = Jueves);

% Solución
solve satisfy;

% Mostrar el resultado
output ["Hoy es: ", show(Hoy)];
```

---

5. Cinco personas, A, B, C, D y E, se paran en una fila. Cada una de ellas lleva un sombrero de un color: rojo, azul, verde, amarillo o blanco. No pueden ver el color de su propio sombrero, pero pueden ver los sombreros de las personas delante de ellas. Se sabe que:

- A ve a B y C, y sus sombreros son de diferentes colores.
- B ve a C, y sus sombreros son del mismo color.
- C no ve ningún sombrero.
- D ve a C y A, y sus sombreros son de diferentes colores.
- E ve a D y B, y sus sombreros son de diferentes colores. ¿Qué color de sombrero lleva cada persona?

```
% Definimos los colores de los sombreros
enum Colores = {Rojo, Azul, Verde, Amarillo, Blanco};

% Variables que representan el color del sombrero de cada persona
var Colores: A;
var Colores: B;
var Colores: C;
var Colores: D;
var Colores: E;
```

```
% Restricciones basadas en las pistas

% A ve a B y C, y sus sombreros son de diferentes colores
constraint B != C;

% B ve a C, y sus sombreros son del mismo color
constraint B = C;

% C no ve ningún sombrero, no impone restricciones

% D ve a C y A, y sus sombreros son de diferentes colores
constraint D != C /\ D != A;

% E ve a D y B, y sus sombreros son de diferentes colores
constraint E != D /\ E != B;

% Solución
solve satisfy;

% Mostrar el color de cada sombrero
output [
  "A tiene sombrero de color: ", show(A), "\n",
  "B tiene sombrero de color: ", show(B), "\n",
  "C tiene sombrero de color: ", show(C), "\n",
  "D tiene sombrero de color: ", show(D), "\n",
  "E tiene sombrero de color: ", show(E), "\n"
];
```

PARTE 2

1. Seis amigos desean pasar sus vacaciones juntos y deciden, cada dos, utilizar diferentes medios de transporte; sabemos que Alejandro no utiliza el coche ya que éste acompaña a Benito que no va en avión. Andrés viaja en avión. Si Carlos no va acompañado de Darío ni hace uso del avión.

¿Podrías decirnos en qué medio de transporte llega a su destino Tomás?

SOLUCIÓN:

```
% Definimos los medios de transporte

enum Transporte = {Coche, Avion, Otro};


% Variables para representar el medio de transporte de cada persona

var Transporte: Alejandro;

var Transporte: Benito;

var Transporte: Andres;

var Transporte: Carlos;

var Transporte: Dario;

var Transporte: Tomas;


% Restricciones basadas en la información proporcionada


% Alejandro no utiliza el coche

constraint Alejandro != Coche;


% Benito no va en avión y está acompañado por Alejandro

constraint Benito != Avion /\ Benito = Alejandro;


% Andrés viaja en avión

constraint Andres = Avion;


% Carlos no va acompañado de Darío ni utiliza el avión

constraint Carlos != Avion /\ Carlos != Dario;
```

```
% Resolver el medio de transporte de Tomás
solve satisfy;

% Mostrar el medio de transporte de cada amigo
output [
  "Alejandro viaja en: ", show(Alejandro), "\n",
  "Benito viaja en: ", show(Benito), "\n",
  "Andrés viaja en: ", show(Andres), "\n",
  "Carlos viaja en: ", show(Carlos), "\n",
  "Darío viaja en: ", show(Dario), "\n",
  "Tomás viaja en: ", show(Tomas), "\n"
];
```

---

2. Imagina que estás planeando una expedición de senderismo y tienes una mochila que puede llevar hasta 15 kg. Tienes varios artículos que puedes llevar, cada uno con su propio peso y valor. Tu objetivo es maximizar el valor total de los artículos que llevas sin exceder la capacidad de la mochila.

Aquí están los artículos disponibles:

1. Saco de dormir: Peso = 3 kg, Valor = 15
2. Tienda de campaña: Peso = 5 kg, Valor = 20
3. Cantimplora: Peso = 1 kg, Valor = 5
4. Comida: Peso = 4 kg, Valor = 10
5. Ropa: Peso = 2 kg, Valor = 2
6. Botiquín: Peso = 1 kg, Valor = 8

SOLUCIÓN:

```
% Definimos los pesos y valores de los artículos
int: n = 6; % Número de artículos
array[1..n] of int: pesos = [3, 5, 1, 4, 2, 1]; % Pesos de los artículos
array[1..n] of int: valores = [15, 20, 5, 10, 2, 8]; % Valores de los artículos
int: capacidad = 15; % Capacidad máxima de la mochila en kg

% Variables binarias para representar si un artículo es seleccionado o no
array[1..n] of var 0..1: seleccion;
```

```
% Restricción: El peso total de los artículos seleccionados no debe exceder la capacidad
constraint sum(i in 1..n)(seleccion[i] * pesos[i]) <= capacidad;

% Objetivo: Maximizar el valor total de los artículos seleccionados
solve maximize sum(i in 1..n)(seleccion[i] * valores[i]);

% Mostrar los resultados
output [
  "Artículos seleccionados:\n",
  if seleccion[1] = 1 then "Saco de dormir\n" else "" endif,
  if seleccion[2] = 1 then "Tienda de campaña\n" else "" endif,
  if seleccion[3] = 1 then "Cantimplora\n" else "" endif,
  if seleccion[4] = 1 then "Comida\n" else "" endif,
  if seleccion[5] = 1 then "Ropa\n" else "" endif,
  if seleccion[6] = 1 then "Botiquín\n" else "" endif,
  "\nValor total: ", show(sum(i in 1..n)(seleccion[i] * valores[i])), "\n",
  "Peso total: ", show(sum(i in 1..n)(seleccion[i] * pesos[i])), " kg\n"
];
```

---

2. En una granja de pollos se tiene una dieta para engordar, la cual consiste de una composición mínima de 15 unidades de una sustancia A y otras 15 de una sustancia B. En el mercado sólo se encuentra dos clases de compuestos: el tipo X con una composición de 1 unidad de A y 5 de B, y el otro tipo, Y, con una composición de 5 unidades de A y 1 de B. El precio del tipo X es de 10 pesos y del tipo Y es de 25 pesos ¿Qué cantidades se han de comprar de cada tipo para cubrir las necesidades con un costo mínimo?

SOLUCIÓN:

```
% Definir las variables de decisión
var 0..100: x; % Cantidad de compuesto X a comprar
var 0..100: y; % Cantidad de compuesto Y a comprar

% Restricciones para las sustancias A y B
constraint x * 1 + y * 5 >= 15;  % Unidades de A
```

```
constraint x * 5 + y * 1 >= 15;  % Unidades de B


% Función objetivo: minimizar el costo total

var int: cost = x * 10 + y * 25;

solve minimize cost;


% Mostrar las soluciones

output ["Cantidad de X: \(x)\n", "Cantidad de Y: \(y)\n", "Costo total: \(cost)\n"];
```

---

3. Se dispone de 600 g de un determinado fármaco para elaborar pastillas grandes y pequeñas. Las grandes pesan 40 g y las pequeñas 30 g. Se necesitan al menos tres pastillas grandes, y al menos el doble de pequeñas que de las grandes. Cada pastilla grande proporciona un beneficio de $2 y la pequeña de $1. ¿Cuántas pastillas se han de elaborar de cada clase para que el beneficio sea máximo?

SOLUCIÓN:

```
% Definir las variables de decisión

var 3..15: g; % Cantidad de pastillas grandes (mínimo 3)

var 0..30: p; % Cantidad de pastillas pequeñas


% Restricciones

constraint g * 40 + p * 30 <= 600;   % Disponibilidad de fármaco

constraint p >= 2 * g;              % El número de pastillas pequeñas debe ser al menos el doble que las grandes


% Función objetivo: maximizar el beneficio total

var int: beneficio = g * 2 + p * 1;

solve maximize beneficio;


% Mostrar las soluciones

output ["Cantidad de pastillas grandes: \(g)\n",

    "Cantidad de pastillas pequeñas: \(p)\n",

    "Beneficio total: \(beneficio)\n"];
```

---

4. Se dispone de 600 g de un determinado fármaco para elaborar pastillas grandes y pequeñas. Las grandes pesan 40 g y las pequeñas 30 g. Se necesitan al menos tres

pastillas grandes, y al menos el doble de pequeñas que de las grandes. Cada pastilla grande proporciona un beneficio de $2 y la pequeña de $1. ¿Cuántas pastillas se han de elaborar de cada clase para que el beneficio sea máximo?

SOLUCIÓN:

```
% Definir las variables de decisión

var 3..15: g; % Cantidad de pastillas grandes (mínimo 3)

var 0..30: p; % Cantidad de pastillas pequeñas


% Restricciones

constraint g * 40 + p * 30 <= 600;   % Restricción de la cantidad de fármaco disponible

constraint p >= 2 * g;               % Al menos el doble de pastillas pequeñas que grandes


% Función objetivo: maximizar el beneficio total

var int: beneficio = g * 2 + p * 1;

solve maximize beneficio;


% Mostrar la solución

output ["Cantidad de pastillas grandes: \(g)\n",
    "Cantidad de pastillas pequeñas: \(p)\n",
    "Beneficio total: \(beneficio)\n"];
```

---

5. Consideremos el problema de generar un horario. Hay 7 clases y 4 espacios. Un espacio puede acomodar un máximo de 2 lecciones. Existen las siguientes restricciones:

– El curso 4 debe ser anterior al curso 6.

– El curso 5 debe ser anterior al curso 7.

– El curso 6 debe ser anterior al curso 2.

– El curso 1 no puede estar en paralelo con los cursos 2, 3, 4 y 7.

– El curso 2 no puede estar en paralelo con los cursos 3 y 6.

– El curso 3 no puede estar en paralelo con los cursos 4, 5 y 6.

– El curso 4 no puede estar en paralelo con los cursos 5 y 6.

– El curso 5 no puede estar en paralelo con el curso 7.

SOLUCIÓN:

```
% Parámetros
int: num_cursos = 7;  % Número total de cursos
int: num_espacios = 4; % Número total de espacios

% Variables de decisión: espacio asignado a cada curso
array[1..num_cursos] of var 1..num_espacios: horario;

% Restricciones de precedencia
constraint horario[4] < horario[6];  % El curso 4 debe ser antes que el curso 6
constraint horario[5] < horario[7];  % El curso 5 debe ser antes que el curso 7
constraint horario[6] < horario[2];  % El curso 6 debe ser antes que el curso 2

% Restricciones de paralelismo (no pueden estar en el mismo espacio)
constraint horario[1] != horario[2] /\ horario[1] != horario[3] /\ horario[1] != horario[4] /\
horario[1] != horario[7];
constraint horario[2] != horario[3] /\ horario[2] != horario[6];
constraint horario[3] != horario[4] /\ horario[3] != horario[5] /\ horario[3] != horario[6];
constraint horario[4] != horario[5] /\ horario[4] != horario[6];
constraint horario[5] != horario[7];

% Restricción de capacidad: un máximo de 2 lecciones por espacio
constraint
   forall(e in 1..num_espacios)(
      sum([if horario[c] == e then 1 else 0 endif | c in 1..num_cursos]) <= 2
   );

% Mostrar la solución
output ["Horario: \n"] ++
     [ "Curso \(c): espacio \(horario[c])\n" | c in 1..num_cursos ];
```

6. Cuadrados mágicos. Dado un cuadrado de 1 a N, colocar los números 1,2....N^2 de forma que todas las filas y columnas sumen la misma cantidad.

SOLUCIÓN:

```
int: N = 3; % Tamaño del cuadrado (3x3 en este caso)

% Definir el rango de los números a usar
set of int: Valores = 1..N*N;

% Matriz para el cuadrado mágico
array[1..N, 1..N] of var Valores: cuadrado;

% Suma mágica
var int: suma_magica = N * (N*N + 1) div 2;

% Restricciones
% 1. Cada valor entre 1 y N^2 debe aparecer exactamente una vez
constraint alldifferent([cuadrado[i, j] | i in 1..N, j in 1..N]);

% 2. Las filas deben sumar la suma mágica
constraint forall(i in 1..N) (
   sum(j in 1..N)(cuadrado[i, j]) = suma_magica
);

% 3. Las columnas deben sumar la suma mágica
constraint forall(j in 1..N) (
   sum(i in 1..N)(cuadrado[i, j]) = suma_magica
);

% 4. Las dos diagonales deben sumar la suma mágica
constraint sum(i in 1..N)(cuadrado[i, i]) = suma_magica;
constraint sum(i in 1..N)(cuadrado[i, N+1-i]) = suma_magica;
```

% Mostrar la solución

output [show(cuadrado)];

---

7. Se quiere encontrar un vector de tamaño MxM con valores de 1..M en el cual:

- Si una casilla de una fila>1 contiene el valor M, la casilla de arriba (misma columna, fila una menos) tiene que ser un 1.

- Si una casilla de una columna c, 1<c<M contiene un valor par p, entonces las casillas de la izquierda y de la derecha contienen los valores p-1 y p+1, respectivamente

- Todos los números 1..M aparecen al menos 2 veces en el array

SOLUCIÓN:

```
int: M; % Tamaño del vector MxM

set of int: Valores = 1..M;


% Definir la matriz MxM

array[1..M, 1..M] of var Valores: matriz;


% Restricciones


% 1. Si una casilla de una fila mayor que 1 contiene el valor M, la casilla de arriba (misma columna, fila una menos) tiene que ser un 1

constraint forall(i in 2..M, j in 1..M)(

   if matriz[i,j] = M then matriz[i-1,j] = 1 else true endif

);


% 2. Si una casilla en una columna c, 1<c<M contiene un valor par p, entonces las casillas de la izquierda y de la derecha contienen los valores p-1 y p+1, respectivamente

constraint forall(i in 1..M, j in 2..M-1)(

   if matriz[i,j] mod 2 = 0 then

      matriz[i,j-1] = matriz[i,j] - 1 /\ matriz[i,j+1] = matriz[i,j] + 1

   else true endif

);
```

```
% 3. Todos los números de 1 a M deben aparecer al menos 2 veces en la matriz
constraint forall(v in Valores)(
   sum(i in 1..M, j in 1..M)(if matriz[i,j] = v then 1 else 0 endif) >= 2
);

% Mostrar la solución
output [show(matriz)];
```