Universidad del Valle
Escuela de Ingeniería de Sistemas y Computación
Programación por Restricciones
Desarrollo: Taller Modelamiento e Implementación CSPs

Juan Marcos Caicedo Mejía (1730504-3743)

2020

PARTE 1 (CSPs)

1. Sudoku

- Variables:

  - `subcuadricula`

    Representa la dimensión de cada subcuadricula. En un Sudoku común y corriente esto tiene un valor de 3.

  - `cuadricula`

    Representa la dimensión de toda la cuadricula grande (la que contiene las subcuadriculas). Al igual que la variable anterior, en un Sudoku común esta variable tiene el valor de 9.

  - `entrada`

    Representa el tablero de Sudoku inicial que posee algunos números preestablecidos. En esta matriz, el 0 representa una celda vacía.

  - `salida`

    Representa el tablero variable sobre el cual las restricciones tomarán acción y construirán la solución del Sudoku.

  - `subcuadricula_uno`

    Es un array que contiene los 9 números correspondientes a la primera subcuadricula

  - `subcuadricula_dos`

    Es un array que contiene los 9 números correspondientes a la segunda subcuadricula

  - `subcuadricula_tres`

    Es un array que contiene los 9 números correspondientes a la tercera subcuadricula

  - `subcuadricula_cuatro`

    Es un array que contiene los 9 números correspondientes a la cuarta subcuadricula

  - `subcuadricula_cinco`

    Es un array que contiene los 9 números correspondientes a la quinta subcuadricula

- `subcuadricula_seis`

  Es un array que contiene los 9 números correspondientes a la sexta subcuadricula

- `subcuadricula_siete`

  Es un array que contiene los 9 números correspondientes a la séptima subcuadricula

- `subcuadricula_ocho`

  Es un array que contiene los 9 números correspondientes a la octava subcuadricula

- `subcuadricula_nueve`

  Es un array que contiene los 9 números correspondientes a la novena subcuadricula

- Dominios:

  - `dimension_cuadricula = 1..cuadricula`

    Este dominio es útil para poder acceder a todas las celdas de la cuadrícula grande, pues es el que demarca que los números que estén en el tablero solo sean del 1 al 9.

- Restricciones:

  - Una primera restricción que solo sirve para poder copiar los datos del tablero inicial del Sudoku consiste en que si alguna celda en el tablero de la variable de entrada contiene un número mayor que 0, este mismo número se copie en la misma ubicación en el tablero de salida, si no es así, es decir, hay una celda con 0, se omite su valor:

$$\forall i, j \in \{1, 9\} \mid entrada(i, j) > 0 \therefore salida(i, j) = entrada(i, j)$$

  - Luego, una restricción importante es para garantizar que los números de todas las filas sean distintos:

$$\forall i, j \in \{1, 9\} \mid$$

$$x_{1,1} \neq x_{1,2} \neq x_{1,3} \neq ... \neq x_{1,9}$$

$$\wedge$$

$$x_{2,1} \neq x_{2,2} \neq x_{2,3} \neq ... \neq x_{2,9}$$

$$\wedge$$

$$...$$

$$x_{9,1} \neq x_{9,2} \neq x_{9,3} \neq ... \neq x_{9,9}$$

- Luego, una restricción importante es para garantizar que los números de todas las columnas sean distintos:

$$\forall i, j \in \{1, 9\} \mid$$

$$x_{1,1} \neq x_{2,1} \neq x_{3,1} \neq ... \neq x_{9,1}$$

$$\wedge$$

$$x_{1,2} \neq x_{2,2} \neq x_{3,2} \neq ... \neq x_{2,9}$$

$$\wedge$$

$$...$$

$$x_{1,9} \neq x_{2,9} \neq x_{3,9} \neq ... \neq x_{9,9}$$

  Donde $x_{i,j}$ es el número de la celda en la posición (i, j).

- Luego, usando las variables de:

  `subcuadricula_uno`

  hasta

  `subcuadricula_nueve`

  Se impusieron las restricciones que permiten que en cada subcuadrícula no se repitan los números del 1 al 9. Por cada variable de subcuadricula (que contiene los numeros de dicha subcuadricula) se impuso la restricción de

  `alldifferent`

  Para la

  `subcuadricula_uno`

La restricción sería así:

$$x_{1,1} \neq x_{1,2} \neq x_{1,3}$$
$$\wedge$$
$$x_{2,1} \neq x_{2,2} \neq x_{2,3}$$
$$\wedge$$
$$x_{3,1} \neq x_{3,2} \neq x_{3,3}$$

Para la

`subcuadricula_dos`

La restricción sería así:

$$x_{1,4} \neq x_{1,5} \neq x_{1,6}$$
$$\wedge$$
$$x_{2,4} \neq x_{2,5} \neq x_{2,6}$$
$$\wedge$$
$$x_{3,4} \neq x_{3,5} \neq x_{3,6}$$

Para la

`subcuadricula_tres`

La restricción sería así:

$$x_{1,7} \neq x_{1,8} \neq x_{1,9}$$
$$\wedge$$
$$x_{2,7} \neq x_{2,8} \neq x_{2,9}$$
$$\wedge$$
$$x_{3,7} \neq x_{3,8} \neq x_{3,9}$$

Para la

`subcuadricula_cuatro`

La restricción sería así:

$$x_{4,1} \neq x_{4,2} \neq x_{4,3}$$
$$\wedge$$
$$x_{5,1} \neq x_{5,2} \neq x_{5,3}$$
$$\wedge$$
$$x_{6,1} \neq x_{6,2} \neq x_{6,3}$$

Para la

`subcuadricula_cinco`

La restricción sería así:

$$x_{4,4} \neq x_{4,5} \neq x_{4,6}$$
$$\wedge$$
$$x_{5,4} \neq x_{5,5} \neq x_{5,6}$$
$$\wedge$$
$$x_{6,4} \neq x_{6,5} \neq x_{6,6}$$

Para la

`subcuadricula_seis`

La restricción sería así:

$$x_{4,7} \neq x_{4,8} \neq x_{4,9}$$
$$\wedge$$
$$x_{5,7} \neq x_{5,8} \neq x_{5,9}$$
$$\wedge$$
$$x_{6,7} \neq x_{6,8} \neq x_{6,9}$$

Para la

`subcuadricula_siete`

La restricción sería así:

$$x_{7,1} \neq x_{7,2} \neq x_{7,3}$$
$$\wedge$$
$$x_{8,1} \neq x_{8,2} \neq x_{8,3}$$
$$\wedge$$
$$x_{9,1} \neq x_{9,2} \neq x_{9,3}$$

Para la

`subcuadricula_ocho`

La restricción sería así:

$$x_{7,4} \neq x_{7,5} \neq x_{7,6}$$
$$\wedge$$
$$x_{8,4} \neq x_{8,5} \neq x_{8,6}$$
$$\wedge$$
$$x_{9,4} \neq x_{9,5} \neq x_{9,6}$$

Para la

`subcuadricula_nueve`

La restricción sería así:

$$x_{7,7} \neq x_{7,8} \neq x_{7,9}$$
$$\wedge$$
$$x_{8,7} \neq x_{8,8} \neq x_{8,9}$$
$$\wedge$$
$$x_{9,7} \neq x_{9,8} \neq x_{9,9}$$

2. Kakuro

- Variables:
  - `ancho`
    Constante entera que se encarga de definir cuál será el ancho (en cuadrículas) del Kakuro.
  - `alto`
    Constante entera que se encarga de definir cuál será el alto (en cuadrículas) del Kakuro.
  - `entrada_casillas`
    Matriz constante que hace parte del input del Kakuro: esta matriz denota las casillas disponibles para rellenar (casillas blancas) como 0, y las casillas inhabilitadas para rellenar (generalmente las grises o que tienen diagonal) como -1.
  - `entrada_casillas_left`
    Matriz constante que hace parte del input del Kakuro: esta matriz denota las sumas horizontales en su respectiva casilla a la que pertenecen en el Kakuro como tal.
  - `entrada_casillas_up`
    Matriz constante que hace parte del input del Kakuro: esta matriz denota las sumas verticales en su respectiva casilla a la que pertenecen en el Kakuro como tal.
  - `salida_casillas`
    Matriz de variables de decisión que funciona como apoyo: pues sirve para decirle en qué casillas puede poner valores y en qué no, en la matriz en la cual si funcionan las restricciones.
  - `salida_casillas_def`
    Matriz de variables de decisión que hace el papel del resultado del Kakuro: sobre esta matriz se imponen las verdaderas restricciones de las reglas del Kakuro.
- Dominios:
  - `dimension_ancho = 1..ancho`
    Dominio que denota desde 1 hasta cuánto van las cuadrículas de todo el ancho del Kakuro.

- `dimension_alto = 1..alto`

  Dominio que denota desde 1 hasta cuánto van las cuadrículas de todo el ancho del Kakuro.

- Restricciones:

  - La primera restricción consiste en rellenar la matriz de apoyo

    `salida_casillas`

    Con los mismos elementos del array de

    `entrada_casillas`

    Para poder saber qué casillas tenemos disponibles para rellenar y cuáles no (las inhabilitadas):

    $$\forall i \in dimension\_alto, j \in dimension\_ancho \wedge x \in \{-1, 0\}$$

    `entrada_casillas[i,j] = x`

    $$\Rightarrow$$

    `salida_casillas[i,j] = x`

  - La segunda restricción consiste en que, usando la información del segundo array del input

    `entrada_casillas_left`

    Si llega a encontrarse con un elemento que NO es -1, es decir, que significa que allí hay una suma horizontal, los elementos que se encuentren en la misma fila donde está dicho elemento, y que son 0, es decir, están disponibles para ser rellenados (esta información se consigue a partir de la matriz de apoyo salida_casillas), entonces en la matriz definitiva de la solución salida_casillas_def, dichas celdas de 0 son las que deben se rellenadas con números del 1 al 9.

    $$\forall i \in dimension\_alto, j \in dimension\_ancho$$

    `entrada_casillas_left[i,j] != -1`

    $$\Rightarrow$$

    $$\forall k \in dimension\_ancho$$

`salida_casillas[i,k] = 0`

$$\Rightarrow$$

`salida_casillas_def[i,k]`

$$\in \{1, 9\}$$

- La tercer restricción consiste en que, usando la información del tercer array del input

  `entrada_casillas_up`

  Si llega a encontrarse con un elemento que NO es -1, es decir, que significa que allí hay una suma vertical, los elementos que se encuentren en la misma columna donde está dicho elemento, y que son 0, es decir, están disponibles para ser rellenados (esta información se consigue a partir de la matriz de apoyo salida_casillas), entonces en la matriz definitiva de la solución salida_casillas_def, dichas celdas de 0 son las que deben se rellenadas con números del 1 al 9.

  $$\forall j \in dimension\_ancho, i \in dimension\_alto$$

  `entrada_casillas_up[i,j] != -1`

  $$\Rightarrow$$

  $$\forall k \in dimension\_alto$$

  `salida_casillas[k,i] = 0`

  $$\Rightarrow$$

  `salida_casillas_def[k,i]`

  $$\in \{1, 9\}$$

- La cuarta restricción es la que hace la magia para que el Kakuro corresponda en sus sumas horizontales: Revisa mediante la función de agregación $forall$ la información del segundo array de input:

  `entrada_casillas_left`

Buscando, de manera similar a dos for anidados para recorrer una matriz en un lenguaje común, elemento a elemento, un elemento DISTINTO a -1, lo cúal significa que dicho elemento hace referencia a una suma horizontal que debe ser cumplida como restricción.

De encontrar un elemento distinto a -1, va a verificar que la suma de los elementos que le siguen inmediatamente después en la fila a la ubicación del elemento que encontró (en el arreglo de output salida_casillas_def) sea igual al número que había encontrado que no era -1.

Ahora, los Kakuro tienen distintas posibilidades en este momento, y son dos:

- Que dentro de las casillas siguientes al número que se debe rellenar haya una casilla inhabilitada, con lo cual tendrá casillas habilitadas (blancas) para cumplir la suma inmediatamente antes de dicha casilla inhabilitada.
- Que todas las casillas siguientes al número que debe dar la suma horizontal (dicho número se encuentra en una casilla inhabilitada para rellenar) estén habilitadas para rellenar, hasta llegar al ancho del Kakuro, es decir, que entre todas las casillas siguientes al número que se debe rellenar no haya ninguna inhabilitada para rellenar y esto llegue hasta el ancho del Kakuro.

Lo que sucede en esta restricción es que, revisará que la suma de las casillas siguientes resulte en el número de la suma horizontal a cumplir, pero dependiendo de el caso que se de, revisará dicha suma de números en un rango distinto:

- **Caso 1: En las casillas siguientes al número que debe sumar, alguna casilla inhabilitada**

  Sea $\mathbb{S}$ el conjunto de todas las casillas inmediatamente siguientes a la casilla que contiene la suma horizontal:

  $$\mathbb{S} = \{s_0, s_1, s_2, ..., s_{ancho}\}$$

  Por ejemplo, si el número que debe dar la suma horizontal se encuentra en una posición de columna j, es obligatorio que s_0 esté en la posición $j + 1$.

  Luego, sea $\mathbb{M}$ un subconjunto de $\mathbb{S}$ tal que

  $$m_0, m_1, m_2...m_n \in \mathbb{M} \Leftrightarrow m_0 = m_1 = m_2 = ... = m_n$$

  $$\wedge m_0 = -1 \wedge m_0, m_1, m_2...m_n \in \mathbb{S}$$

  Es decir, $\mathbb{M}$ es el subconjunto de los elementos de el conjunto $\mathbb{S}$ tales que sean iguales a -1. Es importante recalcar que de dicho conjunto, m_0 es el primer elemento que coincide en ser -1 en $\mathbb{S}$, por ende, es el primer elemento de $\mathbb{M}$.

  De igual forma, el predicado $posicion(x)$ hace referencia a que, dado un elemento x de una matriz, $posicion(x)$ devuelve la posicion de columna del elemento x.

  $$\forall i \in dimension\_alto, j \in dimension\_ancho$$

  `entrada_casillas_left[i,j] != -1`

  $$\Rightarrow$$

  $$\sum_{j+1}^{posicion(m_0)-1} x_i = entrada\_casillas\_left(i, j)$$

  Dicha sumatoria denota que la sumatoria de todos los elementos siguientes en la fila al número que se encontró que no

era -1, hasta una posición anterior al siguiente -1, es decir, la última casilla blanca habilitada para rellenar, todos deben sumar dicho número que fue encontrado que no era -1 (suma horizontal). Además deben ser distintos:

$$\forall x \in \{j+1, posicion(m_0) - 1\}$$

$$x_{j+1} \neq x_{j+2} \neq x_{j+3} \neq ... \neq x_{posicion(m_0)-1}$$

- **Caso 2: En las casillas siguientes al número que debe sumar, todas habilitadas hasta el ancho del Kakuro**

$$\forall i \in dimension\_alto, j \in dimension\_ancho$$

`entrada_casillas_left[i,j] != -1`

$$\Rightarrow$$

$$\sum_{j+1}^{ancho} x_i = entrada\_casillas\_left(i,j)$$

Dicha sumatoria denota que la sumatoria de todos los elementos siguientes en la fila al número que se encontró que no era -1, hasta una la última casilla habilitada que está ubicada en el ancho del Kakuro, deben sumar el número que se encontró que no era -1. Además deben ser distintos:

$$\forall x \in \{j+1, ancho\}$$

$$x_{j+1} \neq x_{j+2} \neq x_{j+3} \neq ... \neq x_{ancho}$$

- La quinta restricción es la que hace la magia para que el Kakuro corresponda en sus sumas verticales: Revisa mediante la función de agregación $forall$ la información del tercer array de input:

  `entrada_casillas_up`

  Buscando, de manera similar a dos for anidados para recorrer una matriz en un lenguaje común, elemento a elemento, un elemento DISTINTO a -1, lo cúal significa que dicho elemento hace referencia a una suma vertical que debe ser cumplida como restricción.

De encontrar un elemento distinto a -1, va a verificar que la suma de los elementos que le siguen inmediatamente después en la columna a la ubicación del elemento que encontró (en el arreglo de output salida_casillas_def) sea igual al número que había encontrado que no era -1.

Ahora, los Kakuro tienen distintas posibilidades en este momento, y son dos:

- Que dentro de las casillas siguientes al número que se debe rellenar haya una casilla inhabilitada, con lo cual tendrá casillas habilitadas (blancas) para cumplir la suma inmediatamente antes de dicha casilla inhabilitada.
- Que todas las casillas siguientes al número que debe dar la suma vertical (dicho número se encuentra en una casilla inhabilitada para rellenar) estén habilitadas para rellenar, hasta llegar al alto del Kakuro, es decir, que entre todas las casillas siguientes al número que se debe rellenar no haya ninguna inhabilitada para rellenar y esto llegue hasta el alto del Kakuro.

Lo que sucede en esta restricción es que, revisará que la suma de las casillas siguientes resulte en el número de la suma vertical a cumplir, pero dependiendo de el caso que se de, revisará dicha suma de números en un rango distinto:

- **Caso 1: En las casillas siguientes al número que debe sumar, alguna casilla inhabilitada**

  Sea $\mathbb{S}$ el conjunto de todas las casillas inmediatamente siguientes a la casilla que contiene la suma vertical:

  $$\mathbb{S} = \{s_0, s_1, s_2, ..., s_{alto}\}$$

  Por ejemplo, si el número que debe dar la suma horizontal se encuentra en una posición de fila i, es obligatorio que s_0 esté en la posición $i + 1$.

  Luego, sea $\mathbb{M}$ un subconjunto de $\mathbb{S}$ tal que

  $$m_0, m_1, m_2...m_n \in \mathbb{M} \Leftrightarrow m_0 = m_1 = m_2 = ... = m_n$$

  $$\wedge m_0 = -1 \wedge m_0, m_1, m_2...m_n \in \mathbb{S}$$

  Es decir, $\mathbb{M}$ es el subconjunto de los elementos de el conjunto $\mathbb{S}$ tales que sean iguales a -1. Es importante recalcar que de dicho conjunto, m_0 es el primer elemento que coincide en ser -1 en $\mathbb{S}$, por ende, es el primer elemento de $\mathbb{M}$.

  De igual forma, el predicado $posicion(x)$ hace referencia a que, dado un elemento x de una matriz, $posicion(x)$ devuelve la posicion de fila del elemento x.

  $$\forall j \in dimension\_ancho, i \in dimension\_alto$$

  `entrada_casillas_up[i,j] != -1`

  $$\Rightarrow$$

  $$\sum_{i+1}^{posicion(m_0)-1} x_i = entrada\_casillas\_up(i, j)$$

Dicha sumatoria denota que la sumatoria de todos los elementos siguientes en la columna al número que se encontró que no era -1, hasta una posición anterior al siguiente -1, es decir, la última casilla blanca habilitada para rellenar, todos deben sumar dicho número que fue encontrado que no era -1 (suma vertical). Además deben ser distintos:

$$\forall x \in \{i+1, posicion(m_0) - 1\}$$

$$x_{i+1} \neq x_{i+2} \neq x_{i+3} \neq ... \neq x_{posicion(m_0)-1}$$

- **Caso 2: En las casillas siguientes al número que debe sumar, todas habilitadas hasta el alto del Kakuro**

$$\forall j \in dimension\_ancho, i \in dimension\_alto$$

`entrada_casillas_up[i,j] != -1`

$$\Rightarrow$$

$$\sum_{i+1}^{alto} x_i = entrada\_casillas\_up(i,j)$$

Dicha sumatoria denota que la sumatoria de todos los elementos siguientes en la columna al número que se encontró que no era -1, hasta una la última casilla habilitada que está ubicada en el ancho del Kakuro, deben sumar el número que se encontró que no era -1. Además deben ser distintos:

$$\forall x \in \{i+1, ancho\}$$

$$x_{i+1} \neq x_{i+2} \neq x_{i+3} \neq ... \neq x_{ancho}$$

Adicional: Dentro de la carpeta comprimida encontrará las imágenes **ejemplo_kakuro_sin_resolver.png** y **ejemplo_kakuro_resuelto.png**, que corresponden al Kakuro que está puesto en el archivo **kakuro.mzn**, para su verificación de la solución y ver cómo es el Kakuro en imagen.

3. Secuencia Mágica

- Variables:
  - `n`

    La variable (ingresada por el usuario mediante el IDE o proveída por un archivo de datos .dzn), representa la longitud de la secuencia mágica.
- Dominios:
  - `dominio = 0..n-1`

    Este dominio caracteriza dos cosas: el tamaño de la secuencia mágica (el tamaño del arreglo que la representa) y los posibles valores que toman los números dentro de la secuencia mágica.
- Restricciones:
  - Llamemos *ocurre* a un predicado que recibe una lista l y un número x, así, $ocurre(l, x)$ arroja el número de ocurrencias del número x en la lista l. Otro predicado puede ser *posicion* que dada una lista l y un número x, retorna la posición del número x en la lista l (indexando desde 0). La restricción principal del problema consiste en que la secuencia mágica se caracteriza porque el número i ocurre exactamente x_i veces en la secuencia. Así, la restricción podría modelarse:

    $$\forall x \in l, ocurre(l, posicion(l, x)) = x$$

    Esto quiere decir que el número de la posición de x en la lista l debe ocurrir x veces en la lista l.
  - Una restricción redundante adicional, dicta que la suma de todos los números en la secuencia debe sumar el número n (longitud de la secuencia):

    $$\sum_{i=0}^{n-1} x_i = n$$

    Donde x_i es el i-ésimo elemento de la secuencia mágica.

- Otra restricción adicional, dice que la suma de cada elemento de la secuencia multiplicada por $i-1$ debe ser igual a 0:

$$\sum_{i=0}^{n-1} x_i * (i-1) = 0$$

Donde x_i es el i-ésimo elemento de la secuencia mágica.

- La primera restricción redundante se deduce a partir de que cada número en la secuencia cuenta el número de ocurrencias de un número, y que de el número total de ocurrencias en la secuencia debe ser n, pues en la secuencia no pueden haber números distintos a los comprendidos entre 0 y $n-1$.
- La segunda restricción tiene que ver con que, si multiplicamos el índice de cada valor en la secuencia por el valor correspondiente:

$$0 * x_0 + 1 * x_1 + ... + (n-1) * x_{n-1}$$

Efectivamente esto es igual que simplemente sumarlos:

$$x_0 + x_1 + ... + x_{n-1}$$

Finalmente:

$$0 * x_0 + 1 * x_1 + ... + (n-1) * x_{n-1} = x_0 + x_1 + ... + x_{n-1} = n$$

Esto debido a que la secuencia es mágica.

También podríamos apoyarnos en la siguiente pequeña demostración:

$$\sum_{i=0}^{n-1} x_i = n$$

$$x_0 + x_1 + ... + x_{n-1} = n$$

$$\wedge$$

$$\sum_{i=0}^{n-1} x_i * i = n$$

$$\sum_{i=0}^{n-1} x_i * i = \sum_{i=0}^{n-1} x_i$$

$$0 * x_0 + 1 * x_1 + ...(n-1) * x_{n-1} = x_0 + x_1 + ... + x_{n-1}$$

$$(0 * x_0 - x_0) + (1 * x_1 - x_1) + ... + ((n-1) * x_{n-1} - x_{n-1}) = 0$$

$$-1 * x_0 + 0 * x_1 + ... + (n-2) * x_{n-1} = 0$$

$$\sum_{i=0}^{n-1} x_i * (i-1) = 0$$

4. Acertijo Lógico

- Variables:
  - `apellido_juan, apellido_oscar, apellido_dario`
    Son las variables que corresponden a los apellidos de cada persona: Juan, Oscar y Dario. Los posibles valores que pueden tomar dichas variables son los elementos del dominio (tipo enumerado)

    `APELLIDOS`
  - `musica_juan, musica_oscar, musica_dario`
    Son las variables que corresponden a la musica preferida de cada persona: Juan, Oscar y Dario. Los posibles valores que pueden tomar dichas variables son los elementos del dominio (tipo enumerado)

    `MUSICA`
  - `edad_juan, edad_oscar, edad_dario`
    Son las variables que corresponden a la edad de cada persona: Juan, Oscar y Dario. Los posibles valores que pueden tomar dichas variables son los elementos del dominio (tipo enumerado)

    `EDADES`
- Dominios:
  - `APELLIDOS = {Gonzalez, Garcia, Lopez}`
    Los posibles valores de los apellidos, dados por el enunciado del problema.
  - `MUSICA = {Clasica, Pop, Jazz}`
    Los posibles valores de la música preferida, dados por el enunciado del problema.
  - `EDADES = 24..26`
    Los posibles valores de las edades, de 24 a 26 años.
- Restricciones: Antes podríamos considerar los siguientes predicados:
  1. $apellido(x, y)$ = La persona x se apellida y

2. $musica(x, y)$ = La música favorita de la persona x es y
3. $edad(x, y)$ = La persona x tiene y años

→ Restricciones:

- $\neg apellido(Juan, Gonzalez)$
  (Juan no se apellida González, esto se infiere al decir que Juan es mayor que González)
- $\neg apellido(Oscar, Lopez)$
  (Oscar no se apellida López)
- $\forall x \mid x \in APELLIDOS$
  $apellido(Juan, x_i) \wedge apellido(Oscar, x_{i+1}) \wedge apellido(Dario, x_{i+2}) \Rightarrow x_i \neq x_{i+1} \neq x_{i+2}$
  (Todos se deben apellidar distinto)
- $\neg musica(Dario, Jazz)$
  (A Dario no le gusta el Jazz)
- $\exists x \in APELLIDOS \wedge \exists y \in MUSICA \mid apellido(Oscar, x) \wedge x = Gonzalez \Rightarrow (musica(Oscar, y) \wedge y = Clasica)$
- $\exists x \in APELLIDOS \wedge \exists y \in MUSICA \mid apellido(Dario, x) \wedge x = Gonzalez \Rightarrow (musica(Dario, y) \wedge y = Clasica)$
  (Como Juan no puede ser González, solo Oscar y Dario pueden apellidarse González, y si alguno de los dos se apellida González, su música favorita debe ser la clásica)
- $\forall y \mid y \in MUSICA$
  $musica(Juan, y_i) \wedge musica(Oscar, y_{i+1}) \wedge musica(Dario, y_{i+2}) \Rightarrow y_i \neq y_{i+1} \neq y_{i+2}$
  (A cada uno le gusta un estilo musical distinto)
- $edad(Oscar, 25)$
  (Oscar tiene 25 años)
- $\exists x, y \in EDADES \mid$
  $apellido(Oscar, Gonzalez) \Rightarrow edad(Juan, x) \wedge edad(Oscar, y) \wedge x > y$
  (Si Oscar se apellida Gonzalez, como Juan debe ser mayor que Gonzalez, Juan debería ser mayor que Oscar)

- $\exists x, y \in EDADES \mid$
  $apellido(Dario, Gonzalez) \Rightarrow edad(Juan, x) \wedge edad(Dario, y) \wedge x > y$
  (Si Dario se apellida Gonzalez, como Juan debe ser mayor que Gonzalez, Juan debería ser mayor que Dario)
- $musica(Juan, Pop) \Rightarrow \exists x \in APELLIDOS \wedge \exists y \in EDADES \mid$
  $apellido(Juan, x) \wedge edad(Juan, y) \wedge x \neq Garcia \wedge y \neq 24$
  (Si Juan llegase a ser el fan de la música Pop, no puede apellidarse García ni tener 24 años)
- $musica(Oscar, Pop) \Rightarrow \exists x \in APELLIDOS \wedge \exists y \in EDADES \mid$
  $apellido(Oscar, x) \wedge edad(Oscar, y) \wedge x \neq Garcia \wedge y \neq 24$
  (Si Oscar llegase a ser el fan de la música Pop, no puede apellidarse García ni tener 24 años)
- $musica(Dario, Pop) \Rightarrow \exists x \in APELLIDOS \wedge \exists y \in EDADES \mid$
  $apellido(Dario, x) \wedge edad(Dario, y) \wedge x \neq Garcia \wedge y \neq 24$
  (Si Dario llegase a ser el fan de la música Pop, no puede apellidarse García ni tener 24 años)

5. Ubicación de personas en una reunión

- Variables:

  - `n`

    Denota el número n de personas en la reunión

  - `cantidad_nexts`

    Denota el número de restricciones de vecindad «next», es decir, cuantas duplas hay en el array de next

  - `cantidad_separates`

    Denota el número de restricciones de no vecindad «separate», es decir, cuantas duplas hay en el array de separate

  - `cantidad_distancias`

    Denota el número de restricciones de distancia entre personas «distancia», es decir, cuantas tripletas hay en el array de distancia

  - `entrada`

    Es el arreglo que contiene las personas que saldrán en la foto. Todos sus elementos deben pertenecer al dominio del conjunto enumerado

    `PERSONAS`

  - `next`

    Es el arreglo que contiene las restricciones de proximidad entre una persona en la foto y otra. (cada dupla significa que una persona debe estar al lado de la otra)

  - `separate`

    Es el arreglo que contiene las restricciones de NO proximidad entre una persona en la foto y otra. (cada dupla significa que una persona no puede estar al lado de la otra)

  - `distancia`

    Es el arreglo que contiene las restricciones sobre distancia de separación entre una persona u otra. (cada tripleta significa que la primera persona debe estar separada de la otra persona por máximo un determinado número de personas)

- `salida`

  Es el arreglo que contiene las variables de decisión sobre las cuales se aplican las restricciones apropiadas. Sus elementos también pertenecen al conjunto enumerado de

  `PERSONAS`

- Dominios:

  - `PERSONAS`

    El conjunto de las posibles personas que saldrán en la foto
  - `DOMINIO1 = 1..n`

    Dominio de 1 hasta el número n de personas en la foto
  - `DOMINIO2 = 1..cantidad_nexts`

    Dominio que es útil para saber cuántas restricciones de next hay
  - `DOMINIO3 = 1..cantidad_separates`

    Dominio que es útil para saber cuántas restricciones de separate hay
  - `DOMINIO4 = 1..cantidad_distancias`

    Dominio que es útil para saber cuántas restricciones de distancia hay

- Restricciones:

  - Todas las personas en la fila de la foto deben ser distintas:

    $$\forall x \mid x \in PERSONAS \land x_0, x_1, ..., x_{i-2}, x_{i-1} \in salida$$

    $$x_0 \neq x_1 \neq ... \neq x_{i-2} \neq x_{i-1}$$

  - Restricciones next:

    $$\forall i \in DOMINIO1, j \in DOMINIO2$$

    $$salida(i) = entrada(next(j,1)) \land i = 1 \Rightarrow salida(i+1) = entrada(next(j,2))$$

    $$salida(i) = entrada(next(j,1)) \land i = 8 \Rightarrow salida(i-1) = entrada(next(j,2))$$

    $$salida(i) = entrada(next(j,1)) \land (i \neq 1 \land i \neq 8) \Rightarrow$$

    $$salida(i+1) = entrada(next(j,2)) \oplus salida(i-1) = entrada(next(j,2))$$

- Restricciones separate:

$$\forall i \in DOMINIO1, j \in DOMINIO3$$

$$salida(i) = entrada(separate(j,1)) \wedge i = 1 \Rightarrow salida(i{+}1) \neq entrada(separate(j,2))$$

$$salida(i) = entrada(separate(j,1)) \wedge i = 8 \Rightarrow salida(i{-}1) \neq entrada(separate(j,2))$$

$$salida(i) = entrada(separate(j,1)) \wedge (i \neq 1 \wedge i \neq 8) \Rightarrow$$

$$salida(i{+}1) \neq entrada(separate(j,2)) \oplus salida(i{-}1) \neq entrada(separate(j,2))$$

- Restricciones distancia:

$$\forall i, k \in DOMINIO1, j \in DOMINIO4$$

$$salida(i) = entrada(distancia(j,1)) \wedge salida(k) = entrada(distancia(j,2)) \Rightarrow$$

$$|i - k| \leq distancia(j,3) + 1$$

PARTE 2 (COPs)

7. Construcción de Transformadores

a Tabla que representa los datos del problema:

	Toneladas de material ferromagnético	Horas de trabajo	Precio Venta
Transformador #1	2	7	120.000 USD
Transformador #2	1	8	80.000 USD

b Modelo COP para el problema utilizando los valores presentados en la tabla:

- Variables:
  - `cantidad_toneladas_material_ferromagnetico = 6;`
    Es una constante, corresponde a la cantidad de toneladas de material ferromagnético disponibles, que en el caso del problema, equivale a 6.
  - `cantidad_horas_de_trabajo = 28;`
    Es una constante, corresponde a la cantidad de horas de trabajo disponibles, que en el caso del problema, equivale a 28.
  - `cantidad_toneladas_t1 = 2;`
    Es una constante, corresponde a la cantidad de toneladas de material ferromagnético que se requieren para fabricar una unidad del Transformador #1, en el problema equivale a 2.
  - `cantidad_toneladas_t2 = 1;`
    Es una constante, corresponde a la cantidad de toneladas de material ferromagnético que se requieren para fabricar una unidad del Transformador #2, en el problema equivale a 1.
  - `cantidad_horas_t1 = 7;`
    Es una constante, corresponde a la cantidad de horas de trabajo que se requieren para fabricar una unidad del Transformador #1, en el problema equivale a 7.
  - `cantidad_horas_t2 = 8;`
    Es una constante, corresponde a la cantidad de horas de trabajo que se requieren para fabricar una unidad del Transformador #2, en el problema equivale a 8.

- `precio_venta_t1 = 120000;`
  Es una constante, corresponde al precio de venta al público para el Transformador #1, que en el problema equivale a 120.000
- `precio_venta_t2 = 80000;`
  Es una constante, corresponde al precio de venta al público para el Transformador #2, que en el problema equivale a 80.000
- `t1`
  Variable de decisión que corresponde a cuántas unidades de Transformador #1 se fabricarán.
- `t2`
  Variable de decisión que corresponde a cuántas unidades de Transformador #2 se fabricarán.
- `cantidad_toneladas_material_ferromagnetico_utilizadas = cantidad_toneladas_t1*t1 + cantidad_toneladas_t2*t2;`
  Variable de decisión que corresponde al número total de toneladas de material ferromagnético utilizadas, según el número de transformadores de cada tipo fabricados.
- `cantidad_horas_de_trabajo_utilizadas = cantidad_horas_t1*t1 + cantidad_horas_t2*t2;`
  Variable de decisión que corresponde al número total de horas de trabajo utilizadas, según el número de transformadores de cada tipo fabricados.

- Dominio:
  - $$t_1, t_2 \in \mathbb{Z}^+ \cup \{0\}$$
    Dominio para la cantidad de transformadores a fabricar t_1 y t_2 (cada uno). La cantidad de transformadores a fabricar debe ser entera y no negativa (incluyendo el cero).
- Restricciones:
  - Restricciones de no negatividad:
    $$t_1 \geq 0 \wedge t_2 \geq 0$$

- La cantidad de toneladas de material ferromagnético utilizadas no debe superar la cantidad con la que contamos al inicio:

$$cantidad\_toneladas\_material\_ferromagnetico\_utilizadas \leq$$

$$cantidad\_toneladas\_material\_ferromagnetico$$

- La cantidad de horas de trabajo utilizadas no debe superar la cantidad con la que contamos al inicio:

$$cantidad\_horas\_de\_trabajo\_utilizadas \leq$$

$$cantidad\_horas\_de\_trabajo$$

- Función objetivo:
  - La función objetivo a **maximizar** es multiplicar el precio de venta el Transformador #1 por la cantidad de transformadores t_1 fabricadas, más la multiplicación del precio de venta el Transformador #2 por la cantidad de transformadores t_2:

$$f = precio\_venta\_t_1 * t_1 + precio\_venta\_t_2 * t_2$$

c Presente la implementación del modelo anterior utilizando código MiniZinc: **se encuentra en el archivo transformadores.mzn**

d Proponga un modelo COP genérico para n transformadores asumiendo que los valores en la tabla pueden cambiar:

n transformadores:

- **Entradas**:
  - Una lista de tamaño n que denota los precios de cada transformador:

$$< p_0, p_1, p_2, ..., p_n >$$

Donde p_i denota el precio a la venta del público del transformador i-ésimo

  - Una lista de tamaño n que denota la cantidad de toneladas de material ferromagnético necesarias para fabricar cada transformador:

$$< m_0, m_1, m_2, ..., m_n >$$

Donde m_i denota la cantidad de toneladas de material ferromagnético necesario para fabricar el transformador i-ésimo

- Una lista de tamaño n que denota la cantidad de horas de trabajo necesarias para fabricar cada transformador:

  $$< h_0, h_1, h_2, ..., h_n >$$

  Donde h_i denota la cantidad de horas de trabajo necesarias para fabricar el transformador i-ésimo
- Cantidad de toneladas de material ferromagnético inicial disponibles para utilizar M
- Cantidad de horas de trabajo disponibles para utilizar H

- **Salida**:
  - Lista de tamaño n cuyos elementos $x_0, x_1, x_2, ..., x_n \in \mathbb{Z}^+ \cup \{0\}$ tal que:

    $$\sum_{i=0}^{n} x_i * p_i$$

    sea **máximo** sujeto a

    $$\sum_{i=0}^{n} x_i * m_i \leq M$$

    $$\wedge$$

    $$\sum_{i=0}^{n} x_i * h_i \leq H$$

8. Producción de Gasolina

a Proponga un modelo COP para el problema utilizando los valores presentados en la tabla:

- Entradas:
  Butano = 0, Catalítico Reformado = 1, Nafta Pesado = 2
  - Lista de tamaño 3 que denota la cantidad de Octano de cada ingrediente:
    $$O_i =< 120, 100, 74 >$$
  - Lista de tamaño 3 que denota la cantidad de Valor de presión de cada ingrediente:
    $$P_i =< 60, 2{,}5, 4{,}0 >$$
  - Lista de tamaño 3 que denota la cantidad de Volatilidad de cada ingrediente:
    $$V_i =< 105, 3, 12 >$$
  - Lista de tamaño 3 que denota los costos por galón de cada ingrediente:
    $$C_i =< 0{,}58, 1{,}55, 0{,}85 >$$
- Salida:
  - Lista de tamaño 3 que denota la cantidad de galones a usar de cada ingrediente:
    $$X_i =< x_0, x_1, x_2 >$$
    Tal que:
    $$\sum_{i=0}^{2} x_i = 12000$$

Sujeto a:

$$(\sum_{i=0}^{2} x_i * o_i) \div 12000 \geq 89$$

$$\wedge$$

$$(\sum_{i=0}^{2} x_i * p_i) \div 12000 \leq 11$$

$$\wedge$$

$$(\sum_{i=0}^{2} x_i * v_i) \div 12000 \geq 17$$

Donde:

$$\sum_{i=0}^{2} x_i * c_i$$

Sea **mínimo**.

b Presente la implementación del modelo anterior utilizando código MiniZinc: **se encuentra en el archivo produccion_gasolina_instancia.mzn**

c Proponga un modelo genérico asumiendo que los valores en la tabla pueden cambiar dependiendo del proveedor de cada uno de los ingredientes:

- Entradas:
  Butano = 0, Catalítico Reformado = 1, Nafta Pesado = 2
  - Lista de tamaño 3 que denota la cantidad de Octano de cada ingrediente:
    $$O_i =< o_0, o_1, o_2 >$$
  - Lista de tamaño 3 que denota la cantidad de Valor de presión de cada ingrediente:
    $$P_i =< p_0, p_1, p_2 >$$
  - Lista de tamaño 3 que denota la cantidad de Volatilidad de cada ingrediente:
    $$V_i =< v_0, v_1, v_2 >$$

- Lista de tamaño 3 que denota los costos por galón de cada ingrediente:

$$C_i =< c_0, c_1, c_2 >$$

- Salida:
  - Lista de tamaño 3 que denota la cantidad de galones a usar de cada ingrediente:

$$X_i =< x_0, x_1, x_2 >$$

Tal que:

$$\sum_{i=0}^{2} x_i = 12000$$

Sujeto a:

$$(\sum_{i=0}^{2} x_i * o_i) \div 12000 \geq 89$$

$$\wedge$$

$$(\sum_{i=0}^{2} x_i * p_i) \div 12000 \leq 11$$

$$\wedge$$

$$(\sum_{i=0}^{2} x_i * v_i) \div 12000 \geq 17$$

Donde:

$$\sum_{i=0}^{2} x_i * c_i$$

Sea **mínimo**.

d Presente la implementación del modelo anterior utilizando código MiniZinc: **se encuentra en el archivo produccion_gasolina_generico.mzn** Una instancia con valores distintos a la tablas se encuentra en **produccion_gasolina_generico.dzn**, para ser corrida con el modelo generico