Análisis y Diseño de Algoritmos II

Jesús Alexander Aranda Ph.D   Robinson Duque, Ph.D
Juan Francisco Díaz, Ph. D

Universidad del Valle

jesus.aranda@correounivalle.edu.co
robinson.duque@correounivalle.edu.co
juanfco.diaz@correounivalle.edu.co
Programa de Ingeniería de Sistemas
Escuela de Ingeniería de Sistemas y Computación

Overview

1. Modelamiento Básico III
    - Restricciones Globales
    - SEND + MORE = MONEY
    - Sudoku
2. Ejercicio General
    - Problema de Máquinas

Restricciones Globales

- MiniZinc incluye una biblioteca de restricciones globales que también se puede utilizar para definir modelos.
- Un ejemplo es la restricción `alldifferent` que requiere que todas las variables que aparecen en su argumento sean diferentes.

Restricciones Globales

- La clase anterior vimos que es posible imponer restricciones de este tipo:

```
constraint forall (i,j in 1..3 where i < j)
  (a[i] != a[j]);
```

donde *a* es una matriz aritmética con índice establecido 1..3. Esta restricción logra la conjunción lógica a[1] != a[2] /\a[1] != a[3] /\a[2] != a[3].

- La restricción anterior también se logra con:

```
include "alldifferent.mzn";
%include "globals.mzn";

constraint alldifferent(a)
```

Restricciones Globales

MiniZinc cuenta con una amplia gama de restricciones globales (visite `www.minizinc.org/doc-2.2.3/en/lib-globals.html` para ver el listado completo). Algunas de ellas son:

- Relacionadas con All-Different:
  - **all_different(array [X] of var int: x)**: Restringe el arreglo de enteros x a ser diferente
  - **all_disjoint(array [int] of var set of int: S)**: Restringe el arreglo de conjuntos de enteros S a ser disyuntos.
  - **all_equal(array [X] of var int: x)**: Restringe el arreglo de enteros x a ser igual
  - **nvalue(var int: n, array [int] of var int: x)**: Requiere que el número de valores distintos en x sea n

Restricciones Globales

- Relacionadas con Ordenamiento:
  - **predicate increasing(array [int] of var int: x))** : Restringe los valores de x a estar ordenados ascendentemente (permite duplicados) tal que x[i] <= x[i+1].
  - **predicate decreasing(array [int] of var float: x)** : Restringe los valores de x a estar ordenados de forma descendente (permite duplicados) tal que x[i] <= x[i+1].

Restricciones Globales

- Relacionadas con Conteo:
  - **predicate among(var int: n, array [int] of var int: x, set of int: v)**: Requiere que exactamente n variables en x tomen uno de los valores en v
  - **predicate at_least(int: n, array [int] of var int: x, int: v)**: Requiere que por lo menos n variables en x tomen el valor v
  - **predicate at_most(int: n, array [int] of var int: x, int: v)**: Requiere que a lo sumo n variables en x tomen el valor v
  - **predicate count(array [int] of var int: x, var int: y, var int: c)**: Requiere que c sea el número de ocurrencias de y en x

SEND + MORE = MONEY

En esta última clase del curso exploraremos algunos ejemplos relacionados con problemas de satisfacción de restricciones. Se requiere resolver el siguiente acertijo:

$$\begin{array}{cccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

Si cada letra (D, E, M, N, O, R, S, Y) es un dígito distinto, adicionalmente S y M $\neq 0$:

SEND + MORE = MONEY

```
include "alldifferent.mzn";

var 1..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 1..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;
```

SEND + MORE = MONEY

```
constraint         1000 * S + 100 * E + 10 * N + D
                 + 1000 * M + 100 * O + 10 * R + E
  = 10000 * M + 1000 * O + 100 * N + 10 * E + Y;

constraint alldifferent([S,E,N,D,M,O,R,Y]);

solve satisfy;

output ["   \(S)\(E)\(N)\(D)\n",
        "+  \(M)\(O)\(R)\(E)\n",
        "= \(M)\(O)\(N)\(E)\(Y)\n"];
```

La solución es única: 9567+1085=10652.

Sudoku

Para estudiar un modelo que solucione el Sudoku clásico, primero introduciremos las **expresiones condicionales**. Éstas nos permitirán imponer restricciones al cargar nuestros datos si se cumple determinada condición. Un ejemplo sencillo de su uso:

```
int: r = if y != 0 then x div y else 0 endif;
```

Sudoku

Las expresiones condicionales tienen la forma:

```
if <bool-exp> then <exp-1> else <exp-2> endif
```

(frecuentemente se utiliza para asignar valores a variables) en lugar de una declaración de control de flujo. Se evalúa a $< exp - 1 >$ si la expresión booleana $< bool - exp >$ es verdadera y $< exp - 2 >$ de lo contrario.

Sudoku - Reglas Generales

- Hay una tabla de tamaño nueve por nueve.
- Cada campo de la tabla contiene un número en el rango de uno a nueve.
- En cada columna, todos los números son diferentes (esto obliga a que cada columna contenga todos los números en el rango de nueve).
- En cada fila todos los números son diferentes también.
- Finalmente, la misma regla que se aplica a las columnas y filas también restringe 9 cuadrados de tamaño 3x3 que están en el rompecabezas marcados con líneas más gruesas.
- Los Sudokus se rellenan con algunos valores iniciales. Estos valores ayudan al comienzo de la resolución y la dificultad se puede ajustar por su número y ubicación.

Sudoku - Reglas Generales

	6	8	4		1		7	
				8	5		3	
	2	6	8		9		4	
		7				9		
	5		1		6	3	2	
	4		6	1				
	3		2		7	6	9	

Sudoku

Asuma la siguiente entrada:

```
start=[|
0, 0, 0, 0, 0, 0, 0, 0, 0|
0, 6, 8, 4, 0, 1, 0, 7, 0|
0, 0, 0, 0, 8, 5, 0, 3, 0|
0, 2, 6, 8, 0, 9, 0, 4, 0|
0, 0, 7, 0, 0, 0, 9, 0, 0|
0, 5, 0, 1, 0, 6, 3, 2, 0|
0, 4, 0, 6, 1, 0, 0, 0, 0|
0, 3, 0, 2, 0, 7, 6, 9, 0|
0, 0, 0, 0, 0, 0, 0, 0, 0|];
```

Sudoku

Asuma que se cargan los datos de la siguiente forma:

```
include "alldifferent.mzn";

array[1..9,1..9] of 0..9: inicio; %Tablero inicial
    0 = vacio
array[1..9,1..9] of var 1..9: Sdk;

%Llenar el tablero inicial
constraint forall(i,j in 1..9)(
    if inicio[i,j] > 0 then
            Sdk[i,j] = inicio[i,j]
    else true endif );
```

Sudoku

Implementemos las restricciones del Sudoku...

Sudoku

Ahora una solución más especializada:

```
% Todas las filas son diferentes
constraint forall (i in 1..9) (
  alldifferent([ Sdk[i,j] | j in 1..9]));

% Todas las columnas son diferentes
constraint forall (j in 1..9) (
  alldifferent([ Sdk[i,j] | i in 1..9]));
```

Sudoku

```
% Los sub-cuadros son diferentes
constraint
 forall (a, o in 1..3)(
    alldifferent(
    [Sdk[(a-1)*3 + a1,(o-1)*3+o1] | a1,o1 in 1..3]
    ));


solve satisfy;
```

Sudoku

```
output  [ show_int(1,Sdk[i,j]) ++ " " ++
 if j mod 3 == 0 then " " else "" endif ++
 if j == 9 then
     if i != 9 then
        if i mod 3 == 0 then "\n\n" else "\n" endif
 else "" endif else "" endif
 | i,j in 1..9 ] ++ ["\n"];
```

Problema de Máquinas

El capataz del taller de máquinas desea programar la producción de dos tipos de partes, cada una de las cuales debe someterse a operaciones de torneado, fresado y rectificado en tres máquinas diferentes. El tiempo por lote para las dos partes junto con los tiempos disponibles por máquina y márgenes de ganancia se proporcionan en la tabla de la siguiente manera:

Part	Turning	Milling	Grinding	Profit $ per lot
Part 1	12 hrs/lot	8 hrs/lot	15 hrs/lot	120
Part 2	6 hrs/lot	6.5 hrs/lot	10 hrs/lot	90
Machine time available hrs	60 hrs	75 hrs	140 hrs	

¿Cuántos lotes por cada parte se deben producir para maximizar la ganancia?

Problema de Máquinas

Actividad en clase: Determine cuántos lotes por cada parte debe producir para maximizar las ganancias.

- Proponga un modelo para la instancia dada.
- Generalice el problema. Utilice notación formal para proponer un modelo que soporte cualquier número de partes.
- Implemente los dos modelos en MiniZinc.

Problema de Máquinas

Modelo para la instancia dada:

$$
\begin{array}{ll}
\texttt{maximize} & : \quad 120x_1 + 90x_2 \\
\texttt{subject to} & : \\
& 12x_1 + 6x_2 \leq 60 \\
& 8x_1 + 6{,}5x_2 \leq 75 \\
& 15x_1 + 10x_2 \leq 140 \\
& x_1 \geq 0, x_2 \geq 0
\end{array}
$$

Ahora proponga el modelo genérico...

Variables:

var var

1 2

$Lote_i$ = cantidad de lotes por cada parte i $(1 \leq i \leq N)$

Restricciones

$\forall i, j \in \{1, \dots N\},\ i \neq j \quad Lote_i \neq Lote_j$

* Horas de toneado:

$$\left(\sum_{i=1}^{N} Torneado_i * Lote_i \right) \leq TT$$

* Horas de Fresado:

$$\left(\sum_{i=1}^{N} Fresado_i * Lote_i \right) \leq TM$$

* Horas de Rectificado

$$\left(\sum_{i=1}^{N} Rectificado_i * Lote_i \right) \leq TR$$

* No Negatividad

$\forall i \in \{1, \dots, N\},\ Lote_i \geq 0$

Modelo Genérico:

Función Objetivo:

$$Max: \sum_{i=1}^{N} Ganancia_i * Lote_i$$

Parámetros:

N = número de partes.

TT = Tiempo de torneado

TM = Tiempo de Fresado

TR = Tiempo de Rectificado

12	6

$Torneado_i$ = horas de torneado de la parte i $(1 \leq i \leq N)$

$Fresado_i$ = horas de Fresado de la parte i $(1 \leq i \leq N)$

$Rectificado_i$ = horas de rectificado de la parte i $(1 \leq i \leq N)$

$Ganancia_i$ = Ganancia por cada lote de la parte i $(1 \leq i \leq N)$

Fin de la Presentación

Lecturas recomendadas:

Modelamiento básico en MiniZinc:
https://www.minizinc.org/doc-2.2.3/en/modelling.html

Modelos más complejos:
https://www.minizinc.org/doc-2.2.3/en/modelling2.html

Fin de la Presentación

¿Preguntas?