### MiniZinc

Claude-Guy Quimper

#### MiniZinc

#### https://www.minizinc.org

- MiniZinc est un langage de modélisation, principalement développé par des chercheurs de Melbourne (Université de Melbourne, Université Monash, Data61 Decision Sciences).
- Il permet de facilement modéliser un problème et de soumettre ce problème à un solveur compatible avec MiniZinc.
- Un code en MiniZinc n'est pas un programme qui résout un problème, mais plutôt une description du problème que le solveur devra résoudre seul.

#### Documentation

- Ces diapositives ont pour objectif de démarrer votre apprentissage de MiniZinc.
- Pour maîtriser le language, vous devrez cependant vous référer à la documentation officielle.
  - Site web: <a href="https://www.minizinc.org/">https://www.minizinc.org/</a>
  - Consultez le « MiniZinc Handbook » qui inclut un tutoriel.

3

#### Les bonnes habitudes

- Dans votre code MiniZinc il est recommandé de définir les éléments d'un modèle dans cet ordre:
  - Les constantes
  - Les variables
  - Les contraintes
  - La fonction objectif
  - Les heuristiques de recherche
  - La sortie
- C'est dans cet ordre que nous allons les présenter.

4

#### Variable ou constante?

- Une variable est un inconnu dont on demande au solveur de trouver la valeur.
  - Ce concept provient donc des mathématiques où l'on cherche un inconnu plutôt que de l'informatique où une variable dans un code de programmation est toujours connue, mais peut changer de valeur.
- Une constante est un connu au moment de résoudre le problème. Il s'agit souvent d'une donnée du problème.

#### Déclaration d'une constante

Syntaxe: <type>: <nom> = <valeur>;

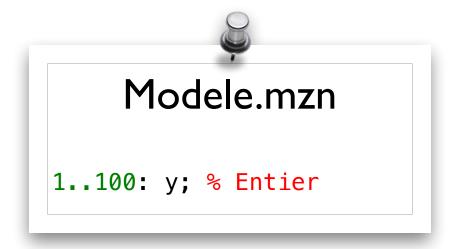
• Exemples:

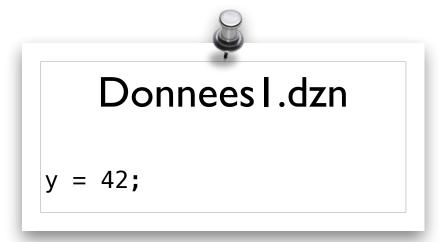
```
float: x = 10.0; % Nombre réel
int: y = 42; % Entier
1..3: z = 2; % Entier entre 1 et 3
array[1..3] of int: vecteur = [1, 2, 1]; % Tableau
array[1..4] of int: v = [1, 2] ++ [3, 4]; % Concaténation
array[1..3, 1..2] of int: matrice = [|1, 2, | 3, 4, | 5, 6|];
set of int: ensemble = {1, 2, 3}; % Ensemble
set of int: interval = 2..4; % Équivalent à {2, 3, 4}
% On peut utiliser des opérations mathématiques
% sur des constantes pour calculer une autre constante.
float: a = x + y;
```

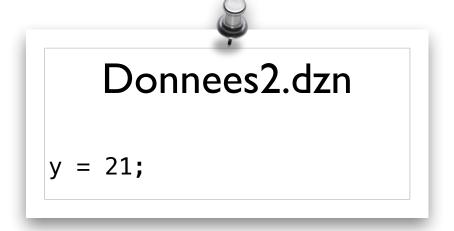
### Fichier de données

- L'architecture MiniZinc permet de séparer les données du modèle en deux fichiers.
- **Fichier.mzn**: Contient le modèle définition des: constantes, variables contraintes, objectifs, ...
- Fichier.dzn: Contient la valeur des constantes.
- Cette façon de diviser le code permet d'utiliser un modèle (modele.mzn) et de l'utiliser pour résoudre plusieurs problèmes (probl.dzn, prob2.dzn, ...)

## Exemple







#### Déclaration d'une variable

- Syntaxe: var <domaine>: <nom>;
- Le domaine doit être un ensemble.
- Exemple:

```
var int: x;  % Attention, le domaine est infiniment grand!
var 1..3: y;
% La dimension du tableau doit être connue
array[1..10] of var 0..1: tableau_bits;
```

9

#### Déclaration d'une contrainte

Syntaxe: constraint < logicial expression >;

```
% Contraintes de base
constraint x = y;
constraint x < y;
constraint x <= y;</pre>
constraint x > y;
constraint x >= y;
constraint x != y; % not equals
% Connecteurs logiques
constraint if x < y then y < z else y > z endif;
constraint x < y \ / \ y != z; % logical "or" constraint x < y \ / \ y != z; % logical "and"
constraint x < y \rightarrow y != z; % logical implication
constraint not (x < y / y > z); % logical negation
```

Source: MiniZinc cheat sheet

# Arithmétique

 La plupart des opérateurs arithmétiques usuels sont définis en MiniZinc.

```
var 0..1: x;
var 0..1: y;
var 0..1: z;
array[0..10] of var 0..5: vecteur;

constraint x + y = z;
constraint 2 * x - 3 * y = x * z;
constraint x = sum(vecteur);
```

### Méta-contrainte

 Une méta-contrainte est une contrainte qui prend en arguments d'autres contraintes.

```
var 0..1: x;
var 0..1: y;
var 0..1: z;

constraint if z = 0 then y = 0 endif;
constraint if x > 0 then y = z else z > y endif;
```

# Liste en compréhension

- Les listes en compréhension permettent de facilement créer des vecteurs de constantes, de variables ou de contraintes.
- Le mot clé **forall** permet d'ajouter plusieurs contraintes au modèle.

```
% Construction du tableau [2, 4, 6]
array[int] of int: a = [ 2 * i | i in 1..3];
% Construction du tableau [6, 12, 18]
array[int] of int: b = [ 2 * i | i in 1..10 where i mod 3 = 0];
% Crée une contrainte de différence entre chaque paire
% de variables du vecteur x
constraint forall (i, j in 1..n where i<j) (x[i] != x[j]);</pre>
```

# Fonction objectif

 Les problèmes de satisfaction n'ont pas de fonction objectif. On les résouts avec solve satisfy.



 Les problèmes d'optimisation peuvent minimiser ou maximiser une variable ou une expression.

```
solve maximize y;
solve minimize sum(x);
```

## Heuristiques de recherche

- L'annotation int\_search permet de spécifier l'ordre avec lequel les variables et les valeurs sont choisies.
- L'annotation **seq\_seach** permet d'appliquer une stratégie de recherche avant une autre.

### Choix de variables

Heuristique	Critère de sélection
input_order	L'ordre donné à la commande int_search
first_fail	La variable avec le plus petit domaine au moment du branchement.
max_regret	La variable ayant la plus grande différence entre les deux plus petites valeurs de son domaine.
smallest	La variable avec la plus petite valeur dans son domaine.

• <u>D'autres heuristiques</u> de choix de variables existent dans MiniZinc.

### Choix de valeurs

Heuristique	Critère de sélection
indomain_min	La plus petite valeur du domaine
indomain_max	La plus grande valeur du domaine
indomain_median	La valeur médiane
indomain_split	Conserve les valeurs plus petites que la médiane.

 <u>D'autres heuristiques</u> de choix de valeurs existent dans MiniZinc.

#### Afficher la solution

- La commande output affiche une liste de chaînes de caractères.
- La fonction show convertie une variable ou une expression en chaînes de caractères.
- L'opérateur ++ concatène deux chaînes de caractères.
- La fonction join insère une chaîne entre les chaînes d'une liste et concatène le tout.

```
% valeur de x: 42
output ["valeur de x: ", show(x), "\n"];
% 1, 2, 3, 4
output [join(", ", [show(x) | x in vecteur]) ++ "\n"];
```

# Exemple complet

 Les carrés magique (voir le code sur le site web du cours)

4	14	15	1
9	7	6	12
5	11	10	8
16	2	3	13

#### FlatZinc

- Lorsque l'on compile un fichier MiniZinc (mzn), nous obtenons un fichier FlatZinc (fzn).
- Ce dernier est un format d'échange simple, de bas niveau, pouvant être compris par un solveur spécifique.
- Cette compilation permet d'optimiser le modèle avant même que le solveur commence la résolution.
- Elle permet aussi de réécrire certaines contraintes qui ne sont pas prises en charge par le solveur.

## MiniZinc avant compilation

 Exemple tiré du MiniZinc Handbook d'un modèle générant deux cercles qui ne se touchent pas et qui sont imbriqués dans un rectangle.

```
float: width = 10.0; % width of rectangle to hold circles float: height = 8.0; % height of rectangle to hold circles float: r1 = 2.0; var r1..width-r1: x1; % (x1,y1) is center of circle of radius r1 var r1..height-r1: y1; float: r2 = 3.0; var r2..width-r2: x2; % (x2,y2) is center of circle of radius r2 var r2..height-r2: y2; % centers are at least r1 + r2 apart constraint (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) >= (r1+r2)*(r1+r2); solve satisfy;
```

# FlatZinc après la compilation

Remarquez:

```
le retrait des constantes
% Variables

    la création de variables

var 2.0 .. 8.0: x1;
var 2.0 .. 6.0: y1;
                                     • Le calcul automatique des
var 3.0 .. 7.0: x2;
var 3.0 .. 5.0: y2;
                                        domaines de ces variables.
var -5.0..5.0: FLOAT01;

    L'utilisation d'un ensemble

var -25.0..25.0: FLOAT02;
var -3.0..3.0: FLOAT03;
                                        restraint de contraintes
var - 9.0..9.0: FLOAT04;
var 25.0..34.0: FLOAT05;

    L'élimination de sous-

% Constraints
                                        expressions communes.
constraint float plus(FLOAT01, x2, x1);
constraint float plus (FLOAT03, y2, y1);
constraint float plus (FLOAT02, FLOAT04, FLOAT05);
constraint float times (FLOAT01, FLOAT01);
constraint float times (FLOAT03, FLOAT04);
solve satisfy;
```