

# Projet INF402

## Sommaire

### 1) Intro du projet

- Règles
- Exemples

### 2) Résoudre le jeu (formalization)

- Signature des formules
- Liste des règles
- Règles en logique 1er ordre

### 3) Déroulement du programme

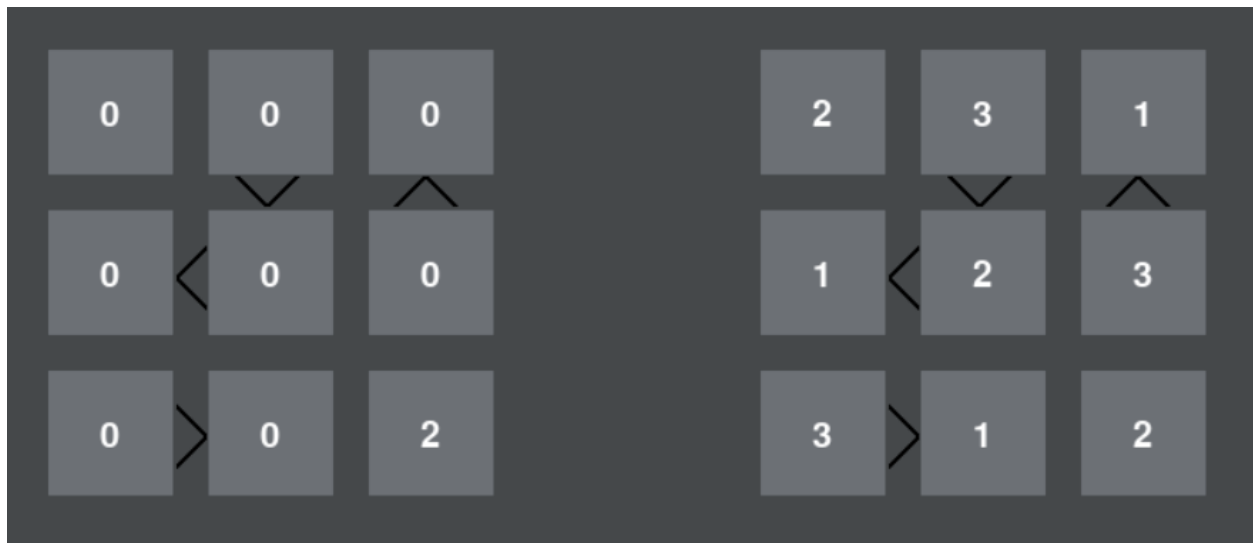
- Introduction
- Entrée du prog
- Implémentation des formules des règles du jeu
- Affichage de la réponse

## 1) Introduction

Le projet a pour but d'utiliser une des les formalisations de la logique appris dans cette UE pour résoudre une jeux forme tableau. Le choix du jeu est Futoshiki.

Futoshiki est une version du Sudoku, mais avec des règles supplémentaires. Comme Sudoku il faut un nombre pas cas par ligne, par contre il n'y a pas des zones comme en Sudoku. D'autre part, il existe une « signe » inferieure qui explique la relation entre deux cases. Le signe peut être dans tout direction. La taille du tableau n'est pas rigide comme Sudoku, mais il peut varier en fonction de la difficulté choisie de l'utilisateur.

Pour arriver à bien résoudre le puzzle, il faut compléter toutes les cases, en suivant les règles imposées. Ici on a une configuration initiale, et la solution.



## 2) Formalisation

Pour résoudre le problème Futushiki il faut utiliser un solveur. Donc il faut créer une liste de clauses qui modélise les règles. Pour créer ces clauses on formalise les règles en Logique du premier ordre.

Dans la logique qu'on a décidé d'utiliser, on utilise :

$X(n, l, c)$  une fonction qui retourne vrai si le nombre  $n$  est placé dans la ligne  $l$  et colonne  $c$ , et il retourne faux sinon.

### Liste des règles

On a concentré toutes les règles différentes dans 5 règles principales

**1ère règle** : chaque nombre doit apparaître au moins 1 fois par ligne

Traduction en logique :  $\forall n, \forall l, \exists c \ X(n, l, c)$

**2ème règle** : chaque nombre doit apparaître au moins 1 fois par colonne

Traduction en logique :  $\forall n, \forall c, \exists l \ X(n, l, c)$

**3ème règle** : Il ne peut pas avoir 2 nombres dans le même cas

Traduction en logique :  $\forall c, \forall l, \forall n1, \forall n2 \ X(n1, l, c) \Rightarrow \neg X(n2, l, c)$

**4ème règle** : Il n'y a pas de répétition du même nombre dans la ligne

Traduction en logique :  $\forall n, \forall l, \forall c1, \forall c2 \ X(n, l, c1) \Rightarrow \neg X(n, l, c2)$

**5ème règle** : Il n'y a pas de répétition du même nombre dans la colonne

Traduction en logique :  $\forall n, \forall c, \forall l1, \forall l2 \ X(n, l1, c) \Rightarrow \neg X(n, l2, c)$

Exemple concret 1<sup>er</sup> règle :

Si je fais l'extension de  $\forall n, \forall l, \exists c \ X(n, l, c)$  avec  $n$  appartient à  $[1, 3]$  et  $l, c$  appartient à  $[0, 2]$

$$\forall n, \forall l, \exists c \ X(n, l, c)$$

$$\forall n, \forall l \ X(n, l, 0) \vee X(n, l, 1) \vee X(n, l, 2)$$

$$\forall n \ X(n, 0, 0) \vee X(n, 0, 1) \vee X(n, 0, 2)$$

$$\wedge X(n, 1, 0) \vee X(n, 1, 1) \vee X(n, 1, 2)$$

$$\wedge X(n, 2, 0) \vee X(n, 2, 1) \vee X(n, 2, 2)$$

$$X(1, 0, 0) \vee X(1, 0, 1) \vee X(1, 0, 2)$$

$$\wedge X(1, 1, 0) \vee X(1, 1, 1) \vee X(1, 1, 2)$$

$$\wedge X(1, 2, 0) \vee X(1, 2, 1) \vee X(1, 2, 2)$$

$$\wedge X(2, 0, 0) \vee X(2, 0, 1) \vee X(2, 0, 2) \dots$$

La liste des clauses dit qu'il y a un 1 dans la première ligne, ou la deuxième ligne ou la troisième. Pareil pour 2 et 3. La liste des clauses crée après l'extension des ces règles vont faire la solution « sudoku ».

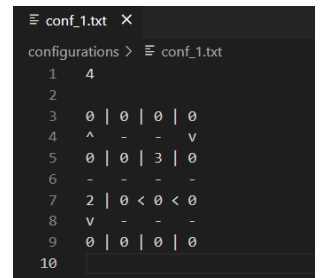
Pour but de sauver espace, on n'a pas fait l'extension des autres règles. Le fichier .dimacs a la fin du traitement va sembler à la liste qui début par l'extension ci-dessus.

### 3) Déroutement du programme

Pour bien utiliser la logique dans notre programme, le nom des variables est **ncl**. sans le X devant. Il y a des limites avec cette style de nommer les variables, mais il existe des solutions différentes.

Le programme écrit en python va prendre ou lire un fichier de configuration de notre choix. Le fichier va contenir tout l'info pour spécifier l'instance de notre Futoshiki.

Ensuite, le programme fait la traduction de ce fichier dans une objet interne. Il faut spécifier que dans cette étape il faut écrire un fichier .dimacs pour ensuite la résoudre.



```
conf_1.txt
configurations > conf_1.txt
1 4
2
3 0 | 0 | 0 | 0
4 ^ - - v
5 0 | 0 | 3 | 0
6 - - - -
7 2 | 0 < 0 < 0
8 v - - -
9 0 | 0 | 0 | 0
10
```

Le fichier .dimacs est créé en se basent sur les règles décrite dans la formalisation. Chaque règle est traduite dans une liste de for loops. La liste des clauses est comporte de 2 parties, la partie des règles sudoku et le data concret. Ce « data concret » consiste des nombres et des signes entre les nombres.

Le fichier .dimacs est donne a un SAT solver (dans notre cas z3 du python) pour enfin l'afficher.

Le programme est une interface graphique très simple avec des boutons. Il y a un bouton qui « importe » la configuration et une autre qui le ressoudé.

