Rapport du projet INF402: Takuzu

MIN1
Lucas Fontana
Mathias Rivet

26 avril 2024

Table des matières

Table des figures

Résumé

Ce projet a été réalisé dans le cadre de l'UE INF402. L'objectif est ici d'automatiser la traduction des règles du Takuzu en formules logiques afin de trouver un modèle via un SAT-solveur. L'intégralité de ce projet est le fruit de nos efforts, il ne contient de ce fait aucun plagiat provenant d'internet ou d'intelligence artificielle.

1 Présentation du problème

Le problème choisi possède plusieurs appellations, Binairo et Takuzu sont les deux plus répandues. Le Takuzu est un casse-tête logique qui s'effectue sur une grille de taille $n \times m$ pair avec $(n, m) \in \mathbb{N}^2, n \geq 2, m \geq 2$. Le but est alors de remplir entièrement une grille vite ou partiellement complétée. De ce fait une grille plus grande sera plus difficile à finir. La plupart du temps, le problème se joue sur une grille carrée. Les éléments du jeu sont représentés par un couple de symboles au choix, usuellement, 0 et 1 sont utilisés. Ainsi nous utiliserons des 0 et 1 pour nos représentations et traiterons des grilles carrées.

1.1 Règles

Le Takuzu possède trois règles :

Règle 1 Chaque ligne et chaque colonne doivent contenir le même nombre de 0 que de 1, ce qui justifie l'utilisation de grilles aux dimensions paires.

Règle 2 Il est proscrit d'avoir plus de deux fois le même symbole consécutif dans une ligne ou une colonne.

Règle 3 Une fois la grille entièrement complétée, toutes les lignes doivent être deux à deux différentes, tout comme les colonnes.

1.2 Exemples

Soit une grille incomplète de dimension 4×4 :

1	1	0	0
	0	0	
	1		0

Elle peut être remplie de la façon suivante :

1	1	0	0
0	0	1	1
1	0	0	1
0	1	1	0

Des exemples de gilles complètes se trouvent dans le dossier `test`.

1.3 Contraintes

La principale contrainte de ce jeu est la difficulté à modéliser la première règle en logique propositionnelle ou du premier ordre. En effet, il n'est pas aisé de vérifier qu'il y ait exactement le même nombre de 0 et 1 à l'aide de la logique. De plus, les SAT-solveurs possèdent une mauvaise gestion de l'arithmétique.

2 Traduction logique

Pour traduire les règles du Takuzu en logique, nous avons modélisé chaque case de la grille par une variable unique. Pour ce faire, les variables sont nommées en fonction de leur position dans la grille via la formule $l \times n + c$, avec n la dimension de la grille, l ligne et c colonne les coordonnées de la variable tel que $(l, c) \in [\![1; n]\!]^2$. Une négation arithmétique de la variable indiquera la négation logique de celle-ci. Le terme « combinaison » désignera une suite de n éléments (0 ou 1) représentant une ligne ou une colonne. Ainsi une telle « combinaison » sera traduite en logique par la conjonction des variables respectant le signe est la position de ses éléments. La conjonction et la disjonction serons respectivement représentées par . et $+$.

2.1 Logique propositionnelle

Pour une modélisation en logique propositionnelle, il est nécessaire de fixer une taille de grille. Les variables sont représentées sous la forme $x_{l,c}$.

Règle 1 Afin d'obtenir une FNC traduisant la première règle, nous construisons l'ensemble $\mathcal{E}$ des combinaisons contenant un nombre différent de 0 et 1 pour une grille de taille n. Cette

FNC est obtenue en prenant la négation de la disjonction des éléments de cet ensemble. Ainsi pour $n = 2$, l'ensemble des combinaisons est $\mathcal{E} = \{(0, 0), (1, 1)\}$. Ce qui nous donne :

$$\begin{array}{r}\overline{((-x_{1,1}. - x_{1,2}) + (x_{1,1}.x_{1,2})}\\ \overline{+(-x_{2,1}. - x_{2,2}) + (x_{2,1}.x_{2,2})}\\ \overline{+(-x_{1,1}. - x_{2,1}) + (x_{1,1}.x_{2,1})}\\ \overline{+(-x_{1,2}. - x_{2,2}) + (x_{1,2}.x_{2,2}))}\end{array}$$

Puis en développant, on obtient finalement la FNC suivante :

$$\begin{array}{c}(x_{1,1} + x_{1,2}).(-x_{1,1} + -x_{1,2})\\ .(x_{2,1} + x_{2,2}).(-x_{2,1} + -x_{2,2})\\ .(x_{1,1} + x_{2,1}).(-x_{1,1} + -x_{2,1})\\ .(x_{1,2} + x_{2,2}).(-x_{1,2} + -x_{2,2})\end{array}$$

Règle 2 Pour décrire la seconde règle nous vérifions si trois variables consécutives d'une ligne ou d'une colonne possèdent la même valeur. La négation du résultat nous fournit une FNC. Pour $n = 4$ cela nous donne :

$$\begin{array}{r}\overline{((x_{1,1}.x_{1,2}.x_{1,3}) + (x_{1,2}.x_{1,3}.x_{1,4})}\\ \overline{+(-x_{1,1}. - x_{1,2}. - x_{1,3}) + (-x_{1,2}. - x_{1,3}. - x_{1,4})}\end{array}$$

$$\begin{array}{r}\overline{+(x_{2,1}.x_{2,2}.x_{2,3}) + (x_{2,2}.x_{2,3}.x_{2,4})}\\ \overline{+(-x_{2,1}. - x_{2,2}. - x_{2,3}) + (-x_{2,2}. - x_{2,3}. - x_{2,4})}\end{array}$$

$$\begin{array}{r}\overline{+(x_{3,1}.x_{3,2}.x_{3,3}) + (x_{3,2}.x_{3,3}.x_{3,4})}\\ \overline{+(-x_{3,1}. - x_{3,2}. - x_{3,3}) + (-x_{3,2}. - x_{3,3}. - x_{3,4})}\end{array}$$

$$\begin{array}{r}\overline{+(x_{4,1}.x_{4,2}.x_{4,3}) + (x_{4,2}.x_{4,3}.x_{4,4})}\\ \overline{+(-x_{4,1}. - x_{4,2}. - x_{4,3}) + (-x_{4,2}. - x_{4,3}. - x_{4,4})}\end{array}$$

$$\begin{array}{r}\overline{+(x_{1,1}.x_{2,1}.x_{3,1}) + (x_{2,1}.x_{3,1}.x_{4,1})}\\ \overline{+(-x_{1,1}. - x_{2,1}. - x_{3,1}) + (-x_{2,1}. - x_{3,1}. - x_{4,1})}\end{array}$$

$$\begin{array}{r}\overline{+(x_{1,2}.x_{2,2}.x_{3,2}) + (x_{2,2}.x_{3,2}.x_{4,2})}\\ \overline{+(-x_{1,2}. - x_{2,2}. - x_{3,2}) + (-x_{2,2}. - x_{3,2}. - x_{4,2})}\end{array}$$

$$\begin{array}{r}\overline{+(x_{1,3}.x_{2,3}.x_{3,3}) + (x_{2,3}.x_{3,3}.x_{4,3})}\\ \overline{+(-x_{1,3}. - x_{2,3}. - x_{3,3}) + (-x_{2,3}. - x_{3,3}. - x_{4,3})}\end{array}$$

$$\begin{array}{r}\overline{+(x_{1,4}.x_{2,4}.x_{3,4}) + (x_{2,4}.x_{3,4}.x_{4,4})}\\ \overline{+(-x_{1,4}. - x_{2,4}. - x_{3,4}) + (-x_{2,4}. - x_{3,4}. - x_{4,4}))}\end{array}$$

En développant, on obtient alors la FNC ci-après :

$$
\begin{aligned}
&(-x_{1,1} + -x_{1,2} + -x_{1,3}).(-x_{1,2} + -x_{1,3} + -x_{1,4})\\
&.(x_{1,1} + x_{1,2} + x_{1,3}).(x_{1,2} + x_{1,3} + x_{1,4})\\
&.(-x_{2,1} + -x_{2,2} + -x_{2,3}).(-x_{2,2} + -x_{2,3} + -x_{2,4})\\
&.(x_{2,1} + x_{2,2} + x_{2,3}).(x_{2,2} + x_{2,3} + x_{2,4})\\
&.(-x_{3,1} + -x_{3,2} + -x_{3,3}).(-x_{3,2} + -x_{3,3} + -x_{3,4})\\
&.(x_{3,1} + x_{3,2} + x_{3,3}).(x_{3,2} + x_{3,3} + x_{3,4})\\
&.(-x_{4,1} + -x_{4,2} + -x_{4,3}).(-x_{4,2} + -x_{4,3} + -x_{4,4})\\
&.(x_{4,1} + x_{4,2} + x_{4,3}).(x_{4,2} + x_{4,3} + x_{4,4})\\
&.(-x_{1,1} + -x_{2,1} + -x_{3,1}).(-x_{2,1} + -x_{3,1} + -x_{4,1})\\
&.(x_{1,1} + x_{2,1} + x_{3,1}).(x_{2,1} + x_{3,1} + x_{4,1})\\
&.(-x_{1,2} + -x_{2,2} + -x_{3,2}).(-x_{2,2} + -x_{3,2} + -x_{4,2})\\
&.(x_{1,2} + x_{2,2} + x_{3,2}).(x_{2,2} + x_{3,2} + x_{4,2})\\
&.(-x_{1,3} + -x_{2,3} + -x_{3,3}).(-x_{2,3} + -x_{3,3} + -x_{4,3})\\
&.(x_{1,3} + x_{2,3} + x_{3,3}).(x_{2,3} + x_{3,3} + x_{4,3})\\
&.(-x_{1,4} + -x_{2,4} + -x_{3,4}).(-x_{2,4} + -x_{3,4} + -x_{4,4})\\
&.(x_{1,4} + x_{2,4} + x_{3,4}).(x_{2,4} + x_{3,4} + x_{4,4})
\end{aligned}
$$

Règle 3 Enfin, pour construire la troisième règle sous la forme d'une FNC, nous prenons la négation de l'énoncé suivant pour une grille de dimension n, appliqué aux lignes, puis aux colonnes : Soit $\mathcal{S}$ une suite finie de combinaisons, existe-il deux éléments distincts identiques ? Ceci se traduit comme suit pour $n = 2$:

$$
\begin{aligned}
&\overline{((x_{1,1}.x_{1,2}.x_{2,1}.x_{2,2})}\\
&\overline{+(x_{1,1}.x_{1,2}. - x_{2,1}. - x_{2,2})}\\
&\overline{+(-x_{1,1}. - x_{1,2}.x_{2,1}.x_{2,2})}\\
&\overline{+(-x_{1,1}. - x_{1,2}. - x_{2,1}. - x_{2,2})}\\
&\overline{+(x_{1,1}.x_{2,1}.x_{1,2}.x_{2,2})}\\
&\overline{+(x_{1,1}.x_{2,1}. - x_{1,2}. - x_{2,2})}\\
&\overline{+(-x_{1,1}. - x_{2,1}.x_{1,2}.x_{2,2})}\\
&\overline{+(-x_{1,1}. - x_{2,1}. - x_{1,2}. - x_{2,2}))}
\end{aligned}
$$

La FNC finale s'obtient en développant :

$$\begin{aligned}(-x_{1,1} + -x_{1,2} + -x_{2,1} + -x_{2,2})\\ .(-x_{1,1} + -x_{1,2} + x_{2,1} + x_{2,2})\\ .(x_{1,1} + x_{1,2} + -x_{2,1} + -x_{2,2})\\ .(x_{1,1} + x_{1,2} + x_{2,1} + x_{2,2})\end{aligned}$$

$$\begin{aligned}.(-x_{1,1} + -x_{2,1} + -x_{1,2} + -x_{2,2})\\ .(-x_{1,1} + -x_{2,1} + x_{1,2} + x_{2,2})\\ .(x_{1,1} + x_{2,1} + -x_{1,2} + -x_{2,2})\\ .(x_{1,1} + x_{2,1} + x_{1,2} + x_{2,2})\end{aligned}$$

2.2 Logique du premier ordre

À partir de la traduction en logique propositionnelle des trois règles il est possible d'en faire une généralisation au premier ordre. Pour cela, les variables $x_{l,c}$ sont modélisées par le prédicat $P(l, c, n)$. Ce prédicat vaut 1 pour une variable positive et 0 pour une variable négative. Dans cette section, le symbole $+$ représente malheureusement deux opérateurs binaires différents, l'addition d'entiers et la disjonction logique.

Règle 1 Soit $\mathcal{E}$ l'ensemble des combinaisons de prédicats P qui possèdent un nombre différent de 0 et 1 pour une grille de taille n, on a :

$$\overline{\sum_{e\in\mathcal{E}}(\sum_{l=1}^{n} e_l + \sum_{c=1}^{n} e_c)}$$

Règle 2

$$\begin{aligned}&\forall l_1 \in [\![1; n]\!], \forall c_1 \in [\![1; n-2]\!], \forall l_2 \in [\![1; n-2]\!], \forall c_2 \in [\![1; n]\!],\\ &(P(l_1, c_1, n) + P(l_1, c_1 + 1, n) + P(l_1, c_1 + 2, n))\\ &.(\overline{P(l_1, c_1, n)} + \overline{P(l_1, c_1 + 1, n)} + \overline{P(l_1, c_1 + 2, n)})\\ &.(P(l_2, c_2, n) + P(l_2 + 1, c_2, n) + P(l_2 + 2, c_2, n))\\ &.(\overline{P(l_2, c_2, n)} + \overline{P(l_2 + 1, c_2, n)} + \overline{P(l_2 + 2, c_2, n)})\end{aligned}$$

Règle 3

$$\overline{(\sum_{c_1=1}^{n-1}\sum_{c_2=c_1+1}^{n}\prod_{l=1}^{n} P(l, c_1, n) = P(l, c_2, n))} + \overline{(\sum_{l_1=1}^{n-1}\sum_{l_2=l_1+1}^{n}\prod_{c=1}^{n} P(l_1, c, n) = P(l_2, c, n))}$$

Ou alors :

$$\overline{(\sum_{c_1=1}^{n-1}\sum_{c_2=c_1+1}^{n}\prod_{l=1}^{n} P(l, c_1, n) = P(l, c_2, n))}.\overline{(\sum_{l_1=1}^{n-1}\sum_{l_2=l_1+1}^{n}\prod_{c=1}^{n} P(l_1, c, n) = P(l_2, c, n))}$$

3 Modélisation en FNC

Les traductions en logique propositionnelle et du premier ordre nous ont permis de créer de meilleurs modélisations en FNC. En effet, le développement en FNC puis la factorisation de ces développement à facilité l'implémentation de ces algorithmes.

Règle 1 Soit $\mathcal{E}$ l'ensemble des combinaisons de variable $x_{l,c}$ qui possèdent un nombre différent de 0 et 1 pour une grille de taille n, on a :

$$\prod_{e\in\mathcal{E}}(\prod_{l=1}^{n}\sum_{c=1}^{n}e_c.\prod_{c=1}^{n}\sum_{l=1}^{n}e_l)$$

Règle 2

$$(\prod_{l=1}^{n}\prod_{c=1}^{n-2}(\sum_{k=0}^{2}x_{l,c+k}.\sum_{k=0}^{2}-x_{l,c+k})).(\prod_{c=1}^{n}\prod_{l=1}^{n-2}(\sum_{k=0}^{2}x_{l+k,c}.\sum_{k=0}^{2}-x_{l+k,c}))$$

Règle 3 Soit $\mathcal{K}$ l'ensemble des combinaisons de variables $x_{l,c}$ pour une grille de taille n.

$$(\prod_{l_1=1}^{n-1}\prod_{l_2=l_1+1}^{n}\prod_{k\in\mathcal{K}}\sum_{c=1}^{n}k_{l_1,c}+k_{l_2,c}).(\prod_{c_1=1}^{n-1}\prod_{c_2=c_1+1}^{n}\prod_{k\in\mathcal{K}}\sum_{l=1}^{n}k_{l,c_1}+k_{l,c_2})$$

4 Points critiques de l'implémentation

La deuxième règle nécessite d'avoir l'ensemble des combinaisons de 0 et 1 tel que le nombre de 0 et 1 soit différent. Pour cela nous avons cherché une suite sur un tableur et avons découvert le code de Grey, mais cette approche ne nous à pas permis de trouver une suite. De ce fait, nous avons élaboré un algorithme pour « construire » cet ensemble.

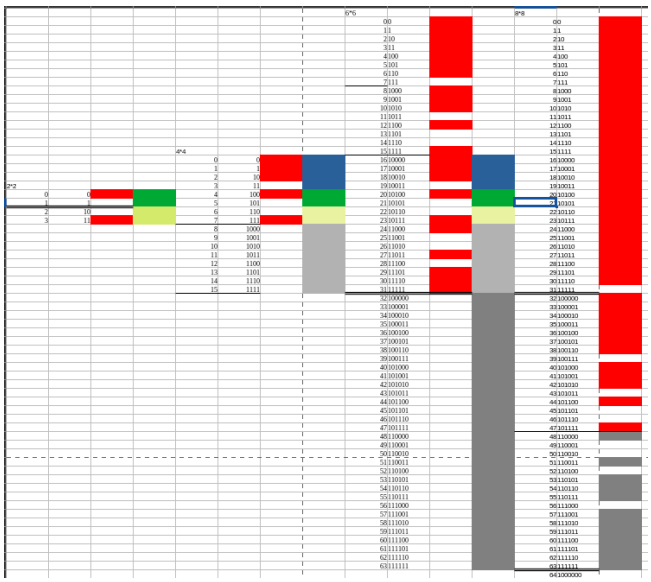

FIGURE 1 – Premier tableur de recherche

Nous avons choisis d'implémenter notre projet en langage C pour sa rapidité d'exécution et son contrôle sur la mémoire. Une première implémentation du projet se fut à l'aide d'une pile, d'une liste chaînée et d'une représentation en style Polonaise inversée. La difficulté d'implémentation des priorités avec des parenthèses et l'aspect trop généraliste de notre implémentation nous ont poussé à reconsidérer notre vision des choses. Ainsi nous avons pu mettre au point des algorithmes ciblées sur nos besoins et les implémentés aisément. Cette simplicité nous à permis de rapidement identifier les failles dans nos algorithmes et de ce fait les corrigés.

5 Exemple d'utilisation

Toutes les informations concernant l'utilisation se trouvent dans le fichier `readme.md`.

2*2		4*4		6*6	
0	0	0	0	0	0
1	1	1	1	1	1
2	10	2	10	2	10
3	11	3	11	3	11
		4	100	4	100
		5	101	5	101
		6	110	6	110
		7	111	7	111
		8	1000	8	1000
		9	1001	9	1001
		10	1010	10	1010
		11	1011	11	1011
		12	1100	12	1100
		13	1101	13	1101
		14	1110	14	1110
		15	1111	15	1111
				16	10000
				17	10001
				18	10010
				19	10011
				20	10100
				21	10101
				22	10110
				23	10111
				24	11000
				25	11001
				26	11010
				27	11011
				28	11100
				29	11101
				30	11110
				31	11111
				32	100000
				33	100001
				34	100010
				35	100011
				36	100100
				37	100101
				38	100110
				39	100111
				40	101000
				41	101001
				42	101010
				43	101011
				44	101100
				45	101101
				46	101110
				47	101111
				48	110000
				49	110001
				50	110010
				51	110011
				52	110100
				53	110101
				54	110110
				55	110111
				56	111000
				57	111001
				58	111010
				59	111011
				60	111100
				61	111101
				62	111110
				63	111111

FIGURE 2 – Second tableur avec le code de Grey