PROGRAMMER AVEC DES ORIGAMIS

La mise au point de portes logiques à base de pliages de papier permet de concevoir des «ordinateurs origami».

RENDEZ-VOUS

L'AUTEUR

JEAN-PAUL DELAHAYE
professeur émérite à l'université de Lille et chercheur au laboratoire Cristal (Centre de recherche en informatique, signal et automatique de Lille)

En utilisant une règle et un compas, on peut mener des sortes de calculs – par exemple en partant d'un segment de longueur 1, on sait en construire un autre de longueur $\sqrt{2}$, ou en partant d'un segment de longueur bien choisie on peut en construire un de longueur $\sqrt{n}$ pour n'importe quel entier n positif. On dit que pour n'importe quel entier n positif, le nombre $\sqrt{n}$ est «constructible à la règle et au compas». On sait en revanche que les nombres π et $\sqrt[3]{2}$ ne pourront jamais être obtenus de cette manière. Pour π, cela provient du fait que ce nombre est transcendant – or, on sait qu'aucun nombre de cette catégorie n'est constructible à la règle et au compas. Pour $\sqrt[3]{2}$, ce sont des caractérisations fines des nombres constructibles à la règle et au compas, datant du XIXe siècle, qui ont permis d'établir qu'on ne pouvait pas l'obtenir ainsi. Par ailleurs, on montre facilement qu'en pliant du papier on peut aussi, partant d'une feuille carrée de côté bien choisi, déterminer par pliage des points distants l'un de l'autre de $\sqrt{n}$ pour tout entier n – on dit que ces nombres sont «constructibles par origami». Cette sorte de «calcul par origami» est aujourd'hui bien comprise et, avec une certaine surprise, on a découvert que cela donnait en fait accès à davantage de nombres que les constructions à la règle et au compas: $\sqrt[3]{2}$, par exemple, est constructible par origami. Notons toutefois que certaines impossibilités persistent, dont celle qui concerne la longueur π.

Un résultat remarquable concernant le calcul par origami vient d'être publié. Il a été établi en 2023 par Thomas Hull, du Franklin and Marshall College, aux États-Unis, et Inna Zakharevich, de l'université Cornell, également aux États-Unis. Un résultat analogue a été démontré presque en même temps par Michael Assis, de l'université de Melbourne, en Australie. Ce nouveau théorème assure que, pour tout algorithme de calcul A, en préparant convenablement des feuilles de papier pliées liées entre elles, ou une très grande feuille unique, on peut concevoir un dispositif qui, lorsqu'on l'actionnera «en le mettant à plat», effectuera les mêmes calculs que l'algorithme A.

Si, par exemple, on veut construire un dispositif qui calcule par pliage la fonction associant le n-ième nombre premier à l'entier n, on procédera de la façon suivante:

(a) On déterminera grâce à la méthode proposée par les chercheurs le bon dispositif en papier correspondant à cette fonction.

(b) Pour l'utiliser, on configurera une extrémité du dispositif pour qu'il représente l'entier n auquel on s'intéresse. Cela aura pour effet que le dispositif ne sera plus plat.

(c) L'opération de calcul consistera à «remettre à plat» le pliage. Pour cela on passera la main dessus, de la zone des données vers la zone des résultats: cela pliera et dépliera le dispositif, ce qui exécutera le calcul attendu.

(d) Une fois le pliage aplati, on récupérera à l'extrémité prévue le codage du n-ième nombre premier.

On dit que le pliage par origami est «computationnellement universel», ou plus simplement qu'il est «Turing-complet»: pour toute fonction

Jean-Paul Delahaye a également publié: **Au-delà du Bitcoin** (Dunod, 2022).

calculable f de l'ensemble des entiers vers lui-même telles qu'elles ont été définies par Alan Turing en 1936, il existe un pliage qui réalise le calcul de f. Rappelons qu'une «fonction calculable» est une fonction de l'ensemble des entiers naturels $\mathbb{N}$ vers lui-même (ou de $\mathbb{N}^k$ vers $\mathbb{N}^m$, pour des entiers k et m donnés), pour laquelle on peut écrire un algorithme qui calcule sa valeur pour n'importe quelle donnée. Rappelons aussi que toutes les fonctions ne sont pas calculables (en fait, Alan Turing avait même démontré que la plupart ne l'étaient pas). Le résultat sur le pliage par origami assure donc que si l'on peut écrire un algorithme effectuant le calcul d'une fonction, on peut aussi construire un pliage en papier exécutant ce même calcul: ce qu'un ordinateur peut faire, un origami le peut également.

TROIS IDÉES PRINCIPALES

La démonstration s'appuie sur trois idées principales.

(1) Grâce à des pliages bien précis, que nous allons expliquer, on peut créer des portes logiques OU, ET, NOT (négation), XOR (OU exclusif), NAND (négation du ET), etc., et on peut également permettre à des données de «circuler» dans le pliage en se croisant, se démultipliant ou s'effaçant.

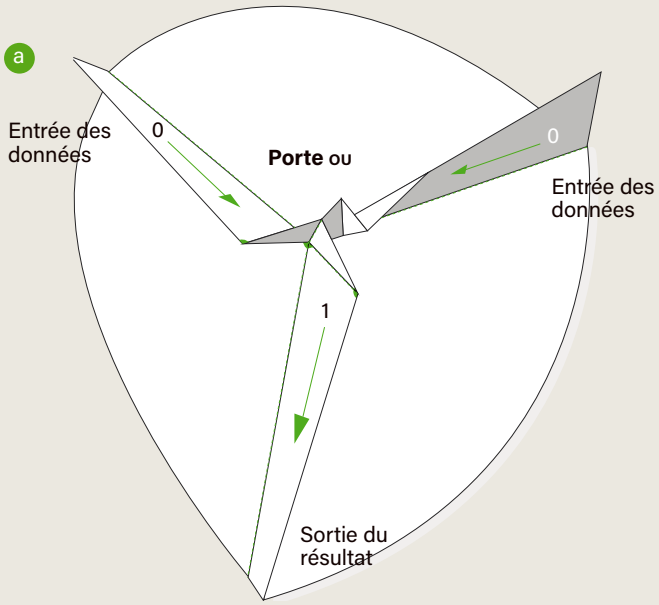
(2) En combinant ces portes logiques en papier et ces dispositifs faisant circuler les données, on en déduit comment fabriquer un grand pliage qui, quand on l'aplatit, se comporte comme un «automate cellulaire à une dimension» (on explique un peu plus loin ce dont il s'agit). Il est

PLIAGES DE PORTES LOGIQUES

1

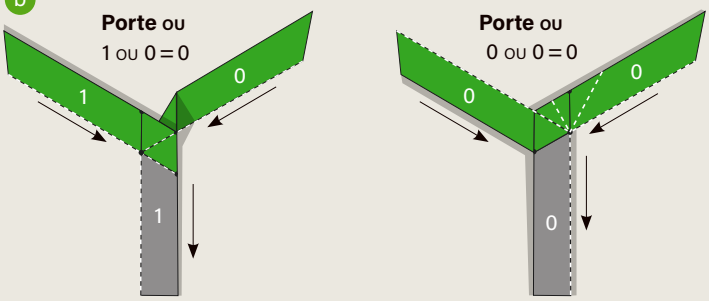
En passant la main du haut vers le bas sur le pliage représenté sur le dessin *(a)*, selon la façon dont sont couchées les deux bandes d'entrée (côté 0 ou côté 1 pour chacune), le pliage au centre provoque le basculement de la bande de sortie du côté 0 ou du côté 1. Le pliage est conçu pour que le résultat corresponde au OU logique entre les entrées : 0 OU 0 = 0 ; 0 OU 1 = 1 ; 1 OU 0 = 1 ; 1 OU 1 = 1.

a



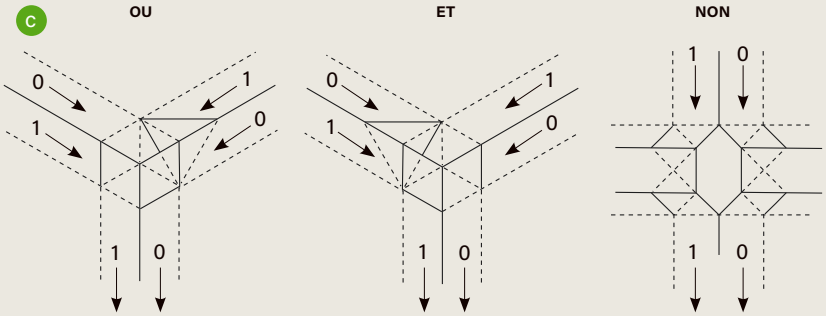
Le dessin *(b)* donne deux configurations (parmi les quatre possibles) de cette porte logique OU.

b



Le dessin *(c)* donne les canevas des portes OU, ET et NON.

c



Les segments en pointillé indiquent les « plis en creux » (comme un lit de rivière) et les segments en trait plein indiquent des « plis montagne » (comme une crête dans un massif montagneux).

possible de simuler de cette manière n'importe quel automate cellulaire unidimensionnel.

(3) On exploite le point précédent pour construire un pliage simulant le fonctionnement d'un automate cellulaire unidimensionnel bien particulier, appelé «automate défini par la règle 110». On utilise alors un résultat démontré en 2004 par Matthew Cook, qui indique que cet automate basé sur la règle 110 est Turing-complet, et on en déduit donc que l'origami est, lui aussi, Turing-complet.

Avant de présenter plus de détails, indiquons que le résultat est sans application pratique. L'article de Thomas Hull et Inna Zakharevich explique comment il faut s'y prendre pour effectuer le calcul de n'importe quelle fonction calculable avec du papier à aplatir, mais la réalisation d'un tel pliage, pour une fonction comme celle calculant le n-ième nombre premier, exigerait une telle quantité de papier et serait tellement délicate et longue à faire fonctionner que personne ne l'envisage. Le résultat est intéressant, il affine notre compréhension de la théorie du calcul et c'est une belle réussite d'avoir su en donner une démonstration, mais il n'aura très probablement aucun effet sur notre capacité à faire de meilleurs ordinateurs.

Notons que l'idée de mener cette démonstration est née dans l'esprit d'Inna Zakharevich, passionnée depuis longtemps d'origami, quand elle a découvert qu'une démonstration du même type existait pour le jeu de la vie, imaginé par John Conway. L'origami permettant de créer des systèmes assez riches et complexes, il devait donc sûrement exister un résultat similaire pour ce dernier... Restait à trouver comment le démontrer !

2 PORTES LOGIQUES

La première étape pour imaginer un ordinateur-origami est la mise au point d'une représentation des bits d'information que la machine en papier va manipuler. Ici on utilisera des bandes de papier qui, selon les manipulations faites, se coucheront soit d'un côté représentant le bit 0 (aussi appelé FAUX), soit de l'autre côté représentant le bit 1 (VRAI). Ces bandes de papier, selon qu'elles finissent couchées du côté 0 ou du côté 1 lors de la mise à plat des portes logiques en origami, produiront un résultat qui sera transmis aux autres portes logiques du dispositif.

La porte logique OU est par exemple connectée par son côté «entrée» à deux bandes, qui codent les données d'entrée de la porte. Une troisième bande correspond au côté «sortie»

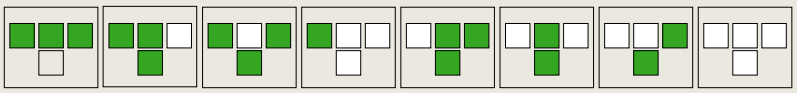
LA RÈGLE 110

L'automate cellulaire défini par la règle 110 peut être représenté sous la forme d'un tableau, donné dans la figure *(a)*, qui associe à chaque configuration d'un microcalculateur et de ses deux voisins à un instant *t* la configuration du microcalculateur central à l'instant suivant *t* + 1.

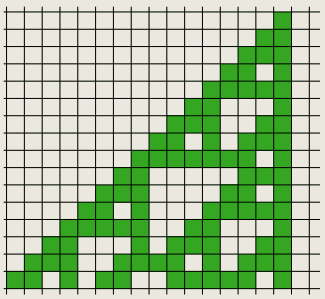
a

111	110	101	100	011	010	001	000
0	1	1	0	1	1	1	0

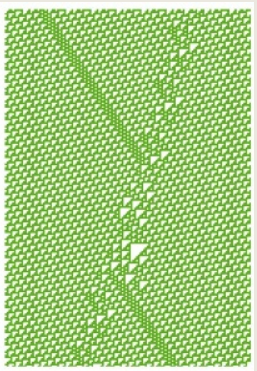
b

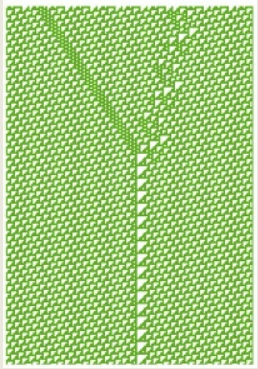


c



d

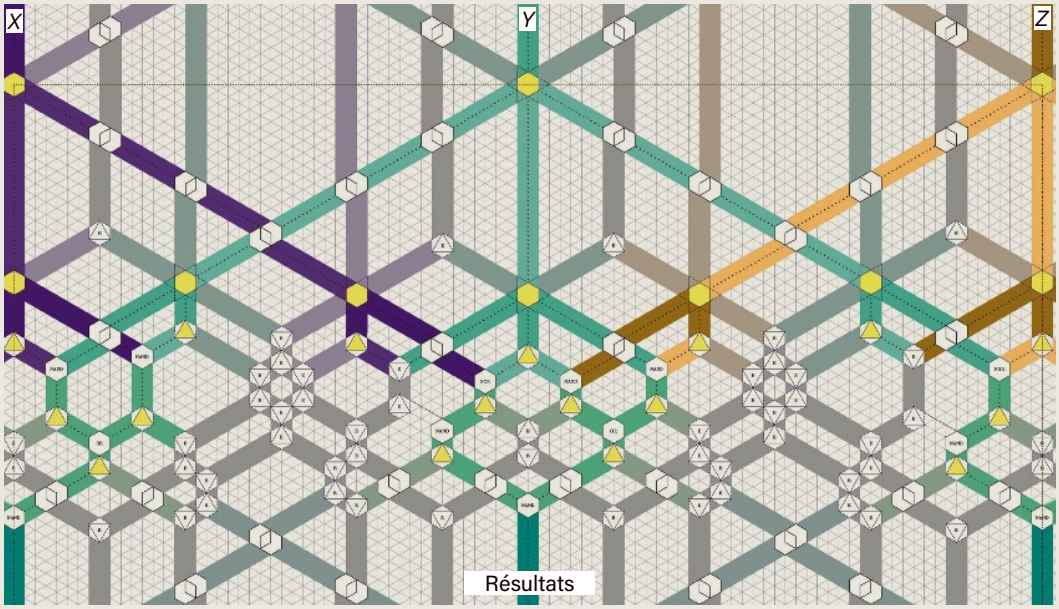




On peut aussi le représenter par un tableau où les bits d'information valant 1 sont remplacés par des cases vertes et les bits d'information valant 0 par des cases blanches, comme le montre la figure *(b)*.
La figure *(c)* détaille un calcul simple. On part d'une ligne de microcalculateurs (la ligne du haut) qui sont tous dans l'état 0, à l'exception d'un seul d'entre eux. Ces microcalculateurs à l'instant suivant sont représentés par la deuxième ligne du tableau, obtenue en appliquant la règle 110 à la première ligne. On fait de même pour la troisième ligne, qui est déterminée en appliquant la règle 110 à la deuxième, etc. La succession des lignes représente l'évolution de l'automate.
On constate sur la figure (d) qu'au cours de l'évolution d'une ligne de microcalculateurs suivant la règle 110, il semble y avoir une sorte de « circulation de données » sur la ligne : certains groupes de cases noires et blanches se déplacent, se croisent, fusionnent ou bifurquent, au fil du temps... C'est ce type d'observation qui donne l'intuition que cet automate peut être exploité pour générer une porte logique entre des flux de données.

3 DES PLIS POUR LA RÈGLE 110

Grâce aux outils présentés dans l'encadré 1, Thomas Hull et Inna Zakharevich ont conçu un système de pliages qui simule la règle 110. Cette étape du travail est assez délicate !
L'image ci-dessous est une représentation schématique du patron de ce pliage. Notez qu'il est présenté sur un maillage triangulaire du plan, pour plus de clarté, mais que ce maillage ne fait pas partie du pliage.



Les bandes *X*, *Y* et *Z* encodent les entrées de la règle 110, et la sortie correspondante est produite et encodée dans la bande « Résultat ».
En plus des portes logiques OU et NAND (négation du ET), indiquées sur le schéma, pour réussir à simuler la règle 110, il a fallu introduire les pliages « intersecteurs », « négation-2 », « négation-3 », et « dévoreur » :
– Les pliages « intersecteurs », représentés par des doubles hexagones dans le schéma, permettent à deux fils de se croiser sans perdre leurs signaux.
- Les pliages « négation-2 », représentés par des triangles jaunes, calculent la négation de leur entrée et envoient la réponse dans deux directions.
– Les pliages « négation-3 », dessinés par des hexagones jaunes, calculent la négation de leur entrée et envoient la réponse dans trois directions.
– Les pliages « dévoreurs-duplicateurs », représentés par un triangle contenant la lettre E (pour *eater* en anglais). Selon la position de la pointe du triangle, vers le haut ou vers le bas, ils fusionnent deux signaux d'entrée ou dupliquent un signal d'entrée unique.

et produit le résultat. Si les bandes en entrée sont couchées de façon à coder 0 et 0, la mise à plat de la porte logique couchera la bande de sortie du côté 0. Pour des entrées 0 et 1, après aplatissement, la bande de sortie indiquera 1, de même que pour les entrées 1 et 0 ou encore 1 et 1. Tout ceci est conforme à la définition du OU logique : 0 OU 0=0, 0 OU 1=1, 1 OU 0=1, 1 OU 1=1. En passant la main sur la porte logique pour la mettre à plat celle-ci calcule donc un OU entre les données de ses entrées et affiche le résultat grâce à la position couchée de sa bande de sortie (*voir l'encadré 1*). On pourra voir fonctionner cette porte OU en consultant la vidéo réalisée par Peter Whitehouse indiquée dans la bibliographie.

AUTOMATES CELLULAIRES UNIDIMENSIONNELS

Le deuxième point de la démonstration consiste à appliquer ce qu'on sait des automates cellulaires unidimensionnels.

4 TURING-COMPLET

La notion de système Turing-complet est aujourd'hui centrale en informatique, car elle concerne tous les systèmes pouvant effectuer des calculs… même s'ils ne sont pas, initialement, prévus pour ça ! Dès qu'on peut exploiter un système pour mener certaines opérations de base sur les nombres entiers ou sur des bits d'information, on sait en déduire qu'en s'y prenant convenablement ces systèmes permettent de mener tous les calculs exécutables par algorithme. Ils sont dans ce cas qualifiés de « systèmes Turing-complets ». Sans surprise, la plupart des langages de programmation sont Turing-complets : c'est le cas de C, Basic, Lisp, Prolog, Java, Python, etc. Cette notion a aussi du sens pour les automates cellulaires à une ou deux dimensions : il a par exemple été démontré que l'automate cellulaire défini par la règle 110 (qui est unidimensionnel) et le jeu de la vie (qui est un automate cellulaire à deux dimensions), de John Conway, étaient Turing-complets. Mais de manière plus surprenante, c'est également le cas du jeu de cartes à jouer et collectionner Magic (*Magic : The Gathering* en anglais). Ce jeu a été inventé par Richard Garfield en 1993, et les trois chercheurs Alex Churchill, Stella Biderman et Austin Herrick ont démontré en 2019 qu'il était Turing-complet – c'est-à-dire qu'on peut faire correspondre les manipulations de cartes autorisées dans le jeu à des calculs sur les entiers, et démontrer que ces calculs définissent un système Turing-complet. Parmi d'autres exemples étonnants, on peut aussi citer le jeu Minecraft.

Un tel automate est une ligne infinie de microcalculateurs. Chacun d'eux peut se trouver dans un état pris parmi une liste finie, et chaque microcalculateur de la ligne évolue en temps discret en déterminant son état à l'instant $t+1$ en fonction, d'une part, de son propre état à l'instant t, et, d'autre part, de l'état de ses voisins à l'instant t. Les plus simples de ces automates sont constitués de microcalculateurs qui n'ont que deux états possibles, 0 ou 1, et dont l'évolution ne dépend que d'eux-mêmes et de leurs deux voisins immédiats. Fixer un tel automate, c'est donc déterminer, par exemple, que « si mon voisin de droite est dans l'état 1, que je suis dans l'état 0 et que mon voisin de gauche est dans l'état 1, alors je passe dans l'état 1 », ce qu'on note : $101 \to 1$. Il y a huit configurations possibles pour un microcalculateur et ses deux voisins, ce qui donne donc huit possibilités pour la partie gauche de la règle. Définir complètement un tel automate exige donc de fixer huit règles, c'est-à-dire de choisir, pour chacune des huit configurations si la partie droite de la règle sera 0 ou 1. Il en résulte qu'au total il y a $2^8 = 256$ automates cellulaires de ce type. L'automate 110 est celui défini par les huit règles suivantes : $111 \to 0$; $110 \to 1$; $101 \to 1$; $100 \to 0$; $011 \to 1$; $010 \to 1$; $001 \to 1$; $000 \to 0$.

La suite des parties droites des règles, lues dans l'ordre indiqué ci-dessus, est 01101110. Cette suite vue comme un nombre binaire correspond au nombre 110 en décimal : c'est lui qui donne son nom à la règle.

COMPORTEMENTS COMPLEXES

La règle 110 engendre des comportements complexes : on perçoit une sorte de structure non triviale dans les résultats produits (*voir l'encadré 2*). La question a été longtemps posée de savoir si on pouvait l'utiliser pour mener des calculs intéressants. Stephen Wolfram, le célèbre créateur du logiciel Mathematica et auteur du livre sur les automates cellulaires *A New Kind of Science* (Wolfram Media, 2002), avait même émis la conjecture que l'automate défini par la règle 110 était Turing-complet. L'idée était d'encoder des données sous la forme d'une liste de 0 et de 1 sur la ligne de départ, et de récupérer les résultats au bout d'un certain nombre d'étapes d'évolution de cette ligne en suivant la règle 110, cela de façon à simuler n'importe quelle fonction calculable.

X Y Z	NON-*X* ET *Y*	NON-*Y* ET *Z*	*Y* ET NON-*Z*	((NON-*Y* ET *Z*) OU (*Y* ET NON-*Z*))	r_{110} (*X*, *Y*, *Z*)
0 0 0	0	0	0	0	0
0 0 1	0	1	0	1	1
0 1 0	1	0	1	1	1
0 1 1	1	0	0	0	1
1 0 0	0	0	0	0	0
1 0 1	0	1	0	1	1
1 1 0	0	0	1	1	1
1 1 1	0	0	0	0	0

La règle 110 transforme chaque triplet de valeurs booléennes (*X*, *Y*, *Z*) en une nouvelle valeur, donnée par l'expression logique : r_{110} (*X*, *Y*, *Z*) = (NON-*X* ET *Y*) OU ((NON-*Y* ET *Z*) OU (*Y* ET NON-*Z*)).

Cette conjecture a été démontrée en 2004 par Matthew Cook, qui travaillait alors pour Stephen Wolfram et est aujourd'hui chercheur à l'Institut de neuro-informatique à Zurich, en Suisse.

Affirmer que l'automate défini par la règle 110 est Turing-complet signifie que pour tout algorithme A, on peut simuler l'action de A en configurant une ligne initiale de 0 et de 1 et en la faisant évoluer en suivant la règle 110. Une partie de cette ligne initiale codera l'algorithme A, et une autre partie codera la donnée – par exemple, un entier n. En laissant la ligne de 0 et de 1 évoluer en suivant la règle 110, on obtiendra au bout d'un temps fini une configuration de 0 et de 1 donnant le résultat que A produit pour la donnée n. Bien sûr coder de cette manière des algorithmes et des données est extrêmement difficile : la démonstration de Matthew Cook est un exploit ! Ce résultat est important, car il montre que des mécanismes d'une grande simplicité – la règle 110 est élémentaire – peuvent détenir un important potentiel comme outils de calcul… pour qui sait les utiliser. Notez que, bien qu'il y ait 256 automates cellulaires unidimensionnels, la preuve qu'ils engendrent un système de calcul Turing-complet n'a été apportée que pour celui défini par la règle 110 et pour ceux qui lui sont directement liés par symétrie.

> **Des mécanismes d'une grande simplicité peuvent détenir un important potentiel comme outils de calcul… pour qui sait les utiliser**

Si le système de calcul engendré par la règle 110 permet bien de simuler n'importe quel algorithme, il faut noter qu'il le fait lentement. En réalité, le surcoût de calcul est même exponentiel : un calcul qu'un algorithme A effectue en n étapes demande un nombre d'étapes de l'ordre de r^n à la simulation de A par la règle 110, avec r une constante plus grande que 1. Heureusement, une variante de la méthode de Matthew Cook a permis de montrer qu'on pouvait mener des simulations évitant ce surcoût déraisonnable pour le transformer en un surcoût polynomial – autrement dit, si l'algorithme A exécute un calcul en n étapes, la simulation effectue ce même calcul en $P(n)$ étapes, où P est un polynôme. Cette variante est due aux chercheurs Turlough Neary, de l'institut de neuro-informatique de Zurich, et Damien Woods, de l'université de Maynooth, en Irlande.

PLIER LA RÈGLE 110

Disposant de portes logiques en origami et de systèmes pour faire circuler les données dans un pliage, et sachant que l'automate défini par la règle 110 est Turing-complet, il ne restait plus qu'à prouver que la règle 110 pouvait être programmée avec les outils de calculs de l'origami. C'est cette démonstration qu'ont proposée Thomas Hull et Inna Zakharevich.

La chose n'est pas simple. Sans entrer dans les détails, vraiment très complexes, notons qu'il existe une formule ramenant la règle 110 à une formule logique. Si l'on note les trois entrées de la règle 110 par X, Y et Z, le résultat est donné par : $r_{110}(X, Y, Z) = (\text{NON-}X \text{ ET } Y) \text{ OU } ((\text{NON-}Y \text{ ET } Z) \text{ OU } (Y \text{ ET NON-}Z))$ (*voir le tableau ci-contre*).

Pour terminer indiquons que cette démonstration repose sur une découverte qualifiée de « miracle » par Kurt Gödel, et qu'on doit à Alonzo Church et à Alan Turing. Tout le monde devrait l'avoir en tête, car elle est au cœur de l'informatique. Cette découverte assure que rien ne sert de concevoir des mécanismes de calcul compliqués comportant de nombreuses primitives de base (« ALLER À », boucles « POUR », « TANT QUE », « SI ALORS SINON », etc.), car dès qu'un minimum, pas très important, est atteint, ajouter de nouvelles primitives n'améliore plus la capacité de calcul. Cette vérité mathématique permet de concevoir des langages de programmation basés sur un très petit nombre d'éléments, elle permet de dessiner des puces assez simples qui pourront faire tout ce qui est faisable par calcul… et elle permet ici de démontrer que le pliage est un outil de calcul aussi puissant que n'importe quel gros ordinateur. Bien sûr ce résultat n'interdit pas de perfectionner les outils de programmation (langages, puces, etc.) pour qu'ils travaillent plus vite et que leur utilisation soit plus commode pour les usagers, mais ces perfectionnements ne concernent jamais l'ensemble des fonctions calculables par ces dispositifs.

Thomas Hull, l'un des chercheurs à qui l'on doit le résultat sur le calcul par origami, conclut à propos de son travail avec Inna Zakharevich que : « Plus nous ferons de choses comme celles-ci, plus je pense que nous aurons de chances d'établir des croisements profonds entre l'origami et des branches bien établies des mathématiques. » ■

BIBLIOGRAPHIE

M. Assis, An origami Universal Turing Machine design, *arXiv preprint*, 2024.

P. Whitehouse, Origami Logic Gates, *Youtube video*, 2024.

J. Cepelewicz, How to build an origami computer, *Quanta Magazine*, 2024.

T. Hull et I. Zakharevich, Flat origami is Turing-complete, *arXiv preprint*, 2023.

T. Hull, *Origametry : Mathematical Methods in Paper Folding*, Cambridge University Press, 2020.

A. Churchill *et al.*, Magic : The Gathering is Turing Complete, *arXiv preprint*, 2019.

T. Neary et D. Woods, P-completeness of Cellular Automaton Rule 110, *in* M. Bugliesi, B. Preneel, V. Sassone, I. Wegener (dir.), *Automata, Languages and Programming,* Icalp 2006 – Lecture Notes in Computer Science, 2006.

M. Cook *et al.*, Universality in elementary cellular automata, *Complex Systems*, 2004.