Travail suggéré au candidat

Dégager les idées forces du texte. Décrire le fonctionnement des machines de Turing et des automates finis. Pour illustrer ses explications, le candidat pourra notamment écrire une machine de Turing différente de celle présentée dans le texte (par exemple f(n) = n+2) et poursuivre le calcul des générations successives de l'automate de Conway présenté à la figure 5.

Problèmes indécidables, machines de Turing, automates finis et paradis artificiels.

Grâce aux travaux des logiciens, les mathématiciens savent précisément ce qu'est une méthode de calcul. Ils savent assez précisément quels problèmes ces méthodes peuvent traiter et mieux encore ils savent que certains problèmes n'auront jamais de solution.

Ces problèmes pour lesquels on démontre qu'il n'existe aucune méthode de résolution sont appelés problèmes "indécidables". Dans les années 1930, les premiers exemples de tels problèmes semblaient artificiels, mais on a rapidement découvert que des problèmes simples sont de ce type. Notamment en informatique, de nombreuses questions sont des problèmes indécidables.

Est-il utile de savoir si un problème est oui ou non indécidable ? Certainement si un problème est indécidable, il ne faut plus perdre son temps à en chercher une solution, et il est préférable de chercher un problème dérivé du premier mais plus simple, dont on doit ensuite démontrer la décidabilité.

Il ne faut pas confondre les problèmes indécidables avec les énoncés indécidables. Si l'indécidabilité d'un problème est absolue, l'indécidabilité d'un énoncé est toujours relative à un système de démonstration. En 1931, Kurt Gödel démontra le premier théorème important concernant les énoncés indécidables : il les caractérisa comme les énoncés impossible à démontrer ainsi que leur négation

Quelques exemples de problèmes décidables et indécidables

Avant de présenter quelques problèmes indécidables célèbres, nous allons donner quelques exemples de problèmes décidables. Si l'on considère deux nombres entiers m et n supérieurs à 1, m est -il un multiple de n ? On sait que 12 est un multiple de 2 et que 16 n'est pas un multiple de 5. On sait comment déterminer de manière systématique les cas où m est un multiple de n:

- Faire la division entière de m par n
- Calculer le reste r
- Si r est nul alors m est un multiple de n, sinon m n'est pas un multiple de n

Ce procédé général et systématique constitue un algorithme informel de décision pour le problème des multiples. C'est un procédé d'une sûreté absolue, efficace pour tous les

couples m et n, et qui donne toujours la bonne réponse : le problème des multiples est décidable.

D'autres problèmes simples sont également décidables. Ainsi on sait déterminer si un nombre est premier, on sait déterminer la nième décimale du nombre pi, on sait vérifier si un nombre est racine d'une équation… Les énoncés sont simples et les problèmes décidables.

Passons maintenant à un problème dont l'énoncé est un petit peu plus compliqué : soient F1, F2, F3,.., Fn une liste de polygones. Peut-on paver le plan sans recouvrement et espace vide avec des exemplaires de ces polygones, et uniquement de ceux ci ? En 1966, Robert Berger a démontré que ce problème est indécidable : aucun algorithme ne permet par un calcul fini d'établir pour tout ensemble de formes polygonales si oui ou non on peut paver le plan de la façon indiquée. Affirmer que le problème est indécidable est beaucoup plus fort qu'avouer ne pas savoir le résoudre, ce qui marquerait simplement notre ignorance. L'indécidabilité résulte-t-elle du fait que les listes des formes géométriques pavantes sont trop nombreuses ? On est intuitivement amené à considérer des sous problèmes du problème initial. Ainsi on pourrait limiter les formes pavantes à des carrés juxtaposés, que l'on nomme aussi polyominos. Comme il existe beaucoup moins de polyominos que de formes géométriques quelconques, le nouveau problème est beaucoup plus simple que le problème initial. Pourtant les logiciens démontrent à nouveau que le problème du pavage du plan par des polyominos est indécidable. Faudrait-il dans ce cas restreindre les formes à une seule ? Si on considère une forme élémentaire unique composée de carrés juxtaposés : peut-on paver le plan sans recouvrement ni espaces vides avec des exemplaires de la forme élémentaire et sans la faire tourner ? Cette fois-ci le problème est si simple que l'on imagine aisément un algorithme qui traite le problème : sa formulation précise et sa programmation sont peut-être pénibles mais réalisables. Le problème du pavage du plan sans rotation par un unique polyomino est décidable.

Le dixième problème de Hilbert fut posé par David Hilbert lors du Congrès International de Mathématiques à Paris en 1900. L'énoncé était le suivant: soit une équation diophantienne (à coefficients entiers). Peut -on trouver un procédé qui détermine par un nombre fini d'opérations si cette équation possède des solutions en nombres entiers ? Cet énoncé mérite plusieurs remarques: tout d'abord on nomme équation diophantienne une équation de la forme $P(x) = 0$ où P est un polynôme à coefficients entiers. Ainsi $x^2 + y^2 = 1$ est une équation diophantienne à deux inconnues x et y. Elle possède 2 solutions entières qui sont $x = 1$, $y = 0$ et $x = 0$, $y = 1$ (on ne s'intéresse ici qu'aux solutions entières positives, mais

la prise en compte des solutions négatives ne change rien au problème). L'équation diophantienne $x^2 - 991.y^2 = 1$ possède aussi des solutions mais celles ci sont beaucoup moins simples à calculer que précédemment : il s'agit de x = 379 516 400 906 811 930 638 014 896 080 et y = 12 055 735 790 331 359 447 442 538 767 ! Dans son énoncé, Hilbert mentionne "…un procédé qui détermine par un nombre fini d'opérations….". Aujourd'hui nous comprenons qu'il signifiait un algorithme, et que l'énoncé de Hilbert est la question de la décidabilité des équations diophantiennes.

Hilbert pressentait peut - être que son problème n'avait pas de solution: "Parfois il arrive que l'on recherche une solution sous des hypothèses insatisfaites ou inappropriées en un certain sens, et on se trouve donc incapable d'atteindre son but. Naît alors l'objectif de prouver l'impossibilité de la solution sous les hypothèses données et dans le sens envisagé. De telles preuves d'impossibilité on déjà été obtenues par les anciens, comme l'irrationalité de la racine de 2. Dans les mathématiques modernes, la question de l'impossibilité de certaines questions a joué un rôle clef. C'est ainsi que nous avons acquis la connaissance que de vieux et difficiles problèmes comme prouver l'axiome des parallèles, la quadrature du cercle, ou résoudre des équations du cinquième degré par radicaux n'ont pas de solution dans le sens envisagé initialement."

L'histoire se répéta pour les équations diophantiennes. Le problème résista jusqu'en 1970 où il fut définitivement résolu par Yuri Matijasevic à l'Institut Mathématique de Steklov à Leningrad, et la solution a été conforme à ce que l'on pressentait: il n'existe aucun algorithme qui indique, pour chaque équation diophantienne, si elle a ou non des solutions en nombres entiers. Ce résultat, en même temps que la solution d'un problème ancien, est particulièrement intéressant car il marque la maturité des techniques de démonstration en théorie de la décidabilité

La décidabilité d'un problème peut, comme nous l'avons vu, être associée à l'existence d'un algorithme qui permet de résoudre le problème. Dans la suite de ce document, nous allons nous intéresser à deux manière de formuler de façon universelle des algorithmes : les machines de Turing et les automates finis

Les machines de Turing

Le mécanisme de machine de Turing dû au mathématicien britannique Alan Turing est à la fois très simple et très puissant. C'est même le plus simple des mécanismes de calcul universel que l'on puisse envisager. Qu'il soit simple, cela apparaîtra dans sa définition, qu'il

soit universel, c'est à dire qu'il permette effectivement de programmer n'importe quel algorithme, cela constitue la thèse de Church - Turing qui est universellement acceptée car on n'a jamais trouvé d'algorithme qui ne puisse être codé sur une machine de Turing.

Une machine de Turing est un mécanisme idéal, destiné à effectuer des calculs comme n+2 ou 3.n, lorsque l'on fournit la valeur n. Pour effectuer ces calculs, la machine de Turing utilise un ruban illimité, composé de cases jointives, et d'une tête de lecture - écriture, avec laquelle elle lit, écrase, ou écrit sur le ruban. Cette tête se déplace sur le ruban conformément aux instructions de la machine. Une machine possède plusieurs états qui avec les données lues sur la bande, définissent le fonctionnement ultérieur de la machine. Décrire une machine de Turing, c'est décrire comment l'état de la machine évolue et quels sont les déplacements de la tête. Une instruction est par exemple: "si je suis dans l'état E1 et que je lis le symbole 0 sur le ruban, alors je passe dans l'état E2, j'écris 1, et je me déplace d'une case vers la droite". Une telle instruction peut se coder facilement par (E1,0 -> E2,1,D). Chaque instruction n'est déterminée que par l'état courant et par le symbole du ruban placé sous la tête de lecture de la machine. La machine est déterministe, c'est-à-dire que chaque couple (état courant, symbole sur le ruban) ne détermine qu'une seule instruction.

A titre d'exemple, considérons la machine dont les 2 instructions sont (E1,0 -> E2,0,D) et (E1,1 -> E2,0,D). Cette machine, chaque fois qu'on la place sur le ruban ne comportant que des 0 et des 1, se déplace vers la droite jusqu'à trouver un 1 qu'elle transforme en 0 et s'arrête.

La machine de Turing qui calcule la fonction f(n) = 3.n peut être représentée par un graphe (figure 1) qui résume la liste des instructions. Chaque instruction, déterminée par un état et par une valeur lue sur la bande, est représenté par une flèche joignant l'état de départ à l'état d'arrivée, avec des indications d'écriture et de déplacement. Dans un souci de simplicité, la machine représentée sur la figure 2 a été écrite pour un codage naturel des entiers: l'entier n est codé par une suite de n symboles 1 (ce qui est différent du codage binaire usuel).

Dans le tableau de la figure 2, nous avons indiqué le détail des états successifs de la machine et de son ruban pour la donnée initiale n = 2. Le bilan de cet algorithme est que quatre 1 supplémentaires ont été rajoutés. Les n symboles 1 sont devenus 3.n symboles 1.

L'importance du concept de la machine de Turing tient en ce que le type de calculs qu'elle effectue est absolument général. Pour tout algorithme, il existe une machine de Turing qui exécute les mêmes opérations que l'algorithme.

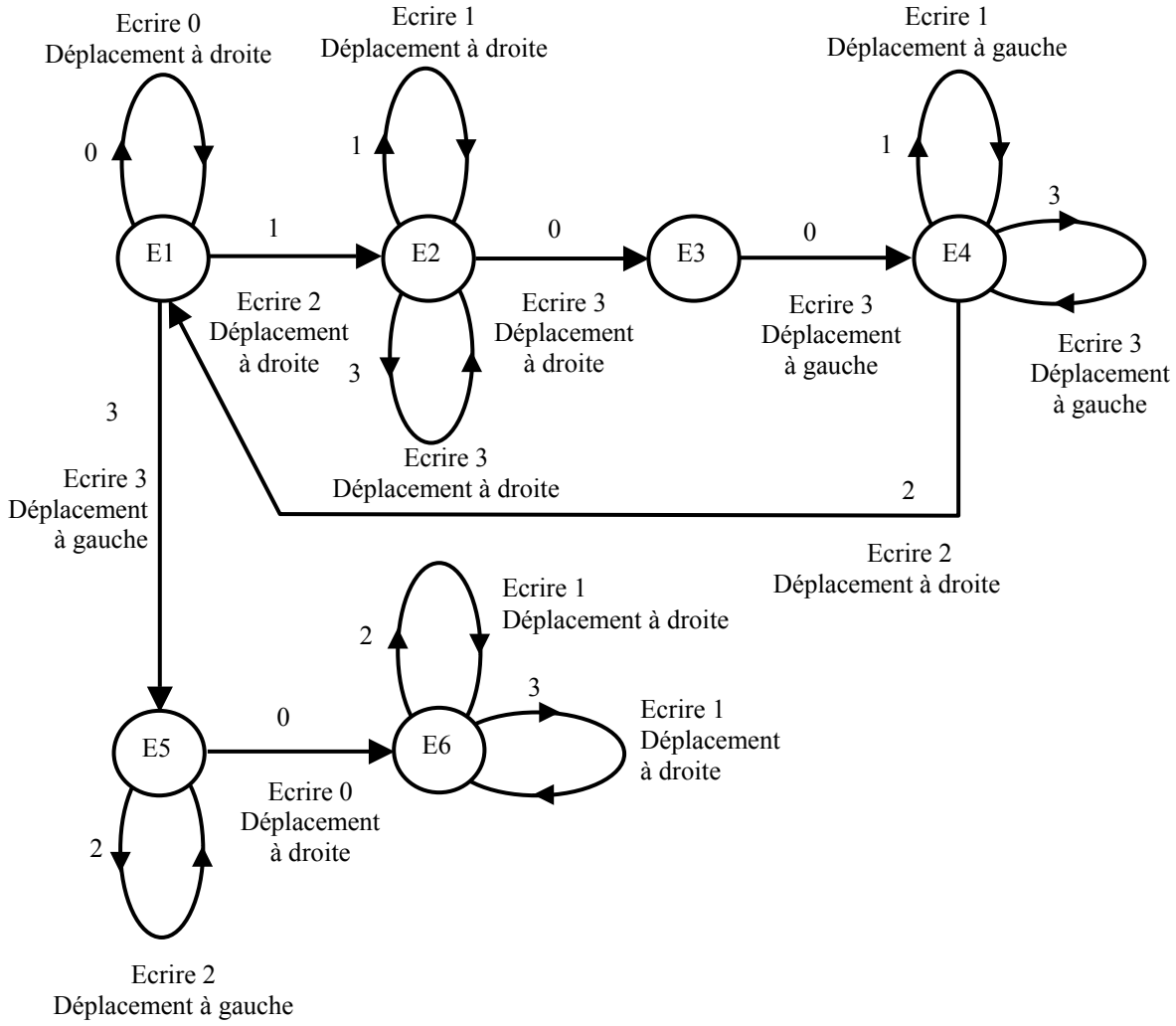

Figure 1 : Machine de Turing correspondant à f(n) = 3.n

Une seconde raison qui a rendu les machines de Turing célèbres est qu'elles sont l'objet d'une des premières démonstrations d'indécidabilité. Le problème étudié est celui de l'arrêt des machines de Turing. Quelles sont les machines qui s'arrêtent et quelles sont celles qui ne s'arrêtent jamais ? Plus précisément, le problème se pose de la façon suivante : les instructions d'une machine de Turing étant données, ainsi qu'un nombre n, la machine de Turing s'arrête-t-elle pour la donnée n ? Comme pour l'exemple précédent, on convient que la donnée n est codée par une suite de n symboles 1 sur le ruban, le reste du ruban étant initialement rempli de 0.

Pour traiter le problème de l'arrêt des machines de Turing, on peut utiliser la thèse de Church - Turing, selon laquelle toute fonction programmable peut l'être avec une machine de Turing, associée à un raisonnement par l'absurde. Supposons que le problème de l'arrêt soit décidable. Alors il existerait une machine de Turing MA qui, chaque fois que l'on écrit sur son ruban les instructions d'une machine quelconque M, et un entier n, effectue les calculs puis

s'arrête en écrivant "oui" sur le ruban si la machine M s'arrête pour la donnée n, ou "non" si la machine M ne s'arrête pas pour la donnée n. Remarquons que les machines de Turing peuvent être numérotées, puisqu'elles sont toutes décrites à l'aide d'une suite finie de symboles. Nous pouvons donc transformer la machine hypothétique MA en une machine MB qui lorsqu'on lui donne un nombre entier n, le transforme en la définition de la machine n, suivi de la donnée n, puis fonctionne comme MA et enfin s'arrête si il y avait écrit "non" ou se met à calculer indéfiniment s'il y avait écrit "oui" à la fin de la deuxième phase. La transformation de la machine MA en machine MB n'est pas difficile car on avait admis que ce que faisait la machine MA était réalisable par algorithme. Comme MB ne fait pas beaucoup plus compliqué que MA, si MA existe, alors MB existe aussi. La machine MB a la propriété remarquable suivante. Par construction, elle ne s'arrête pas lorsqu'on lui donne le numéro n d'une machine

E1	0	1	1	0	0	0	0	0	0	0	…
E1	0	1	1	0	0	0	0	0	0	0	…
E2	0	2	1	0	0	0	0	0	0	0	…
E2	0	2	1	0	0	0	0	0	0	0	…
E3	0	2	1	3	0	0	0	0	0	0	…
E4	0	2	1	3	3	0	0	0	0	0	…
E4	0	2	1	3	3	0	0	0	0	0	…
E4	0	2	1	3	3	0	0	0	0	0	…
E1	0	2	1	3	3	0	0	0	0	0	…
E2	0	2	2	3	3	0	0	0	0	0	…
E2	0	2	2	3	3	0	0	0	0	0	…
E2	0	2	2	3	3	0	0	0	0	0	…
E3	0	2	2	3	3	3	0	0	0	0	…
E4	0	2	2	3	3	3	3	0	0	0	…
E4	0	2	2	3	3	3	3	0	0	0	…
E4	0	2	2	3	3	3	3	0	0	0	…
E4	0	2	2	3	3	3	3	0	0	0	…
E1	0	2	2	3	3	3	3	0	0	0	…
E5	0	2	2	3	3	3	3	0	0	0	…
E5	0	2	2	3	3	3	3	0	0	0	…
E5	0	2	2	3	3	3	3	0	0	0	…
E6	0	2	2	3	3	3	3	0	0	0	…
E6	0	1	2	3	3	3	3	0	0	0	…
E6	0	1	1	3	3	3	3	0	0	0	…
E6	0	1	1	1	3	3	3	0	0	0	…
E6	0	1	1	1	1	3	3	0	0	0	…
E6	0	1	1	1	1	1	3	0	0	0	…
E6	0	1	1	1	1	1	1	0	0	0	…

Figure 2 : Fonctionnement de la machine de Turing qui calcule f(n) = 3.n

M qui s'arrête pour une donnée n. Inversement, MB s'arrête lorsqu'on lui donne le numéro n d'une machine M qui ne s'arrête pas pour la donnée n. Or, MB est aussi une machine de Turing avec un numéro k associé. Que se passe-t-il lorsque l'on donne le nombre entier k à traiter par la machine MB ? Si MB s'arrête pour la donnée k, alors la machine dont le numéro est k ne s'arrête pas pour la donnée k. Comme la machine dont le numéro est k est précisément la machine MB, on doit en déduire que MB ne s'arrête pas pour k ce qui est contradictoire. Inversement, si MB ne s'arrête pas pour la donnée k, alors MB s'arrête pour la donnée k ! Les deux cas possibles conduisent à des contradictions. Comme la seule hypothèse concerne l'existence de la machine MA, la contradiction montre que l'hypothèse est fausse : il n'existe pas de machine de Turing capable de prédire si n'importe quelle machine de Turing s'arrête pour n'importe quelle donnée.

Les automates finis

En 1982 J. Conway, E. Berkelamps et R. Guy proposèrent un autre moyen universel de calcul obtenu à partir d'automates finis.

Comme nous allons le voir, les automates finis illustrent à merveille les problèmes de décidabilité liés à la prédiction en environnement déterministe. Même quand on connaît parfaitement un système physique et toutes les lois qui le régissent, même si de plus ce système ne répond qu'à des lois déterministes, il se peut que son comportement à long terme ne soit pas prévisible. Même dans un univers simplifié, non quantique, que l'on connaîtrait parfaitement, l'avenir continuerait de nous échapper…

Pour illustrer cette idée nous allons considérer les automates finis. Les automates finis sont des entités qui ne possèdent qu'un nombre fini d'états (dans ce document on se limitera à 2 états) et qui sont représentés dans un univers donné (dans ce document il s'agira des cases d'un damier de coté infini). Pour évoluer, un automate fini regarde les automates qui l'entourent, se souvient de l'état dans lequel il est et change éventuellement d'état en respectant les règles invariables qui le caractérisent. Ces conventions sont assimilables à un programme. Le changement porte sur toutes les cases du damier et détermine de façon synchrone une nouvelle génération d'états des cases. En appliquant à nouveau le processus décrit, on obtient une nouvelle génération. Le terme automate désigne ainsi à la fois le mécanisme de calcul associé à une case et cette case sur laquelle il opère en fonction de l'état des cases environnantes. Un automate très simple est l'automate "Déplacement Est": chaque case peut avoir deux états 0 ou 1; l'automate regarde l'état de la case Ouest, s'en souvient, et agit en le

prenant pour nouvel état de la case. Un réseau d'automates "Déplacement Est" sur un plan à pour effet, d'une génération à l'autre, de déplacer d'une case vers l'Est le motif initial.

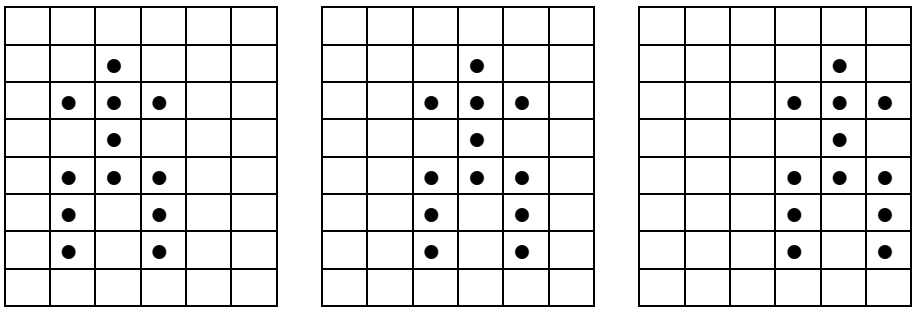

Figure 3 : Automate fini "Déplacement Est"

Un automate fini particulier est celui proposé par John Conway de Cambridge dans les années 1970 et connu aussi sous le nom de Jeu de la Vie. Il se déroule dans un univers extrêmement simple pour lequel la non prévisibilité du système a été prouvée.

L'espace de l'automate de Conway est un plan illimité sur lequel une grille est dessinée. Un seul type de particules élémentaires existe. Le comportement de ces particules suit une loi déterministe très simple:

- Si à un instant t une case est occupée par une particule, et qu'elle possède plus de 3 voisins alors à l'instant suivant t+1 la particule disparaît par étouffement.
- Si à un instant t une case est occupée par une particule, et qu'elle possède moins de 2 voisins alors à l'instant suivant t+1 la particule disparaît par isolement.
- Lorsque une case vide à un instant t possède exactement 3 voisins, alors une nouvelle particule naît dans cette case à l'instant t+1.

L'algorithme correspondant à l'automate de Conway est donné sur la figure 4. Cet algorithme permet de calculer les générations successives d'automates à partir d'une configuration initiale donnée et dans une portion de plan comportant n lignes et p colonnes. Les variables Etat_k et Etat_kplus1 représentent respectivement l'état de l'univers aux générations k et k+1. L'état de chaque automate est codé par 0 pour mort et 1 pour vivant. L'algorithme se déroule horizontalement de la gauche vers la droite. La profondeur est donnée verticalement. Il fonctionne de la manière suivante :

Début

Initialisation avec la configuration initiale codée dans Etat_k

Tant que le nombre de générations souhaité n’est pas atteint

Pour chaque case (i,j) de l'Etat_k qui n'est pas une case de bord

Si (i,j) = 1

Si (i,j) a 2 ou 3 voisins alors écrire 1 dans (i,j) de l'Etat_kplus1

Sinon écrire 0 dans (i,j) de l'Etat_kplus1

Sinon

Si (i,j) a 3 voisins alors écrire 1 dans (i,j) de l'Etat_kplus1

Sinon écrire 0 dans (i,j) de l'Etat_kplus1

Actualiser l'Etat_k avec l'Etat_kplus1 (et afficher éventuellement l'Etat_k)

Fin

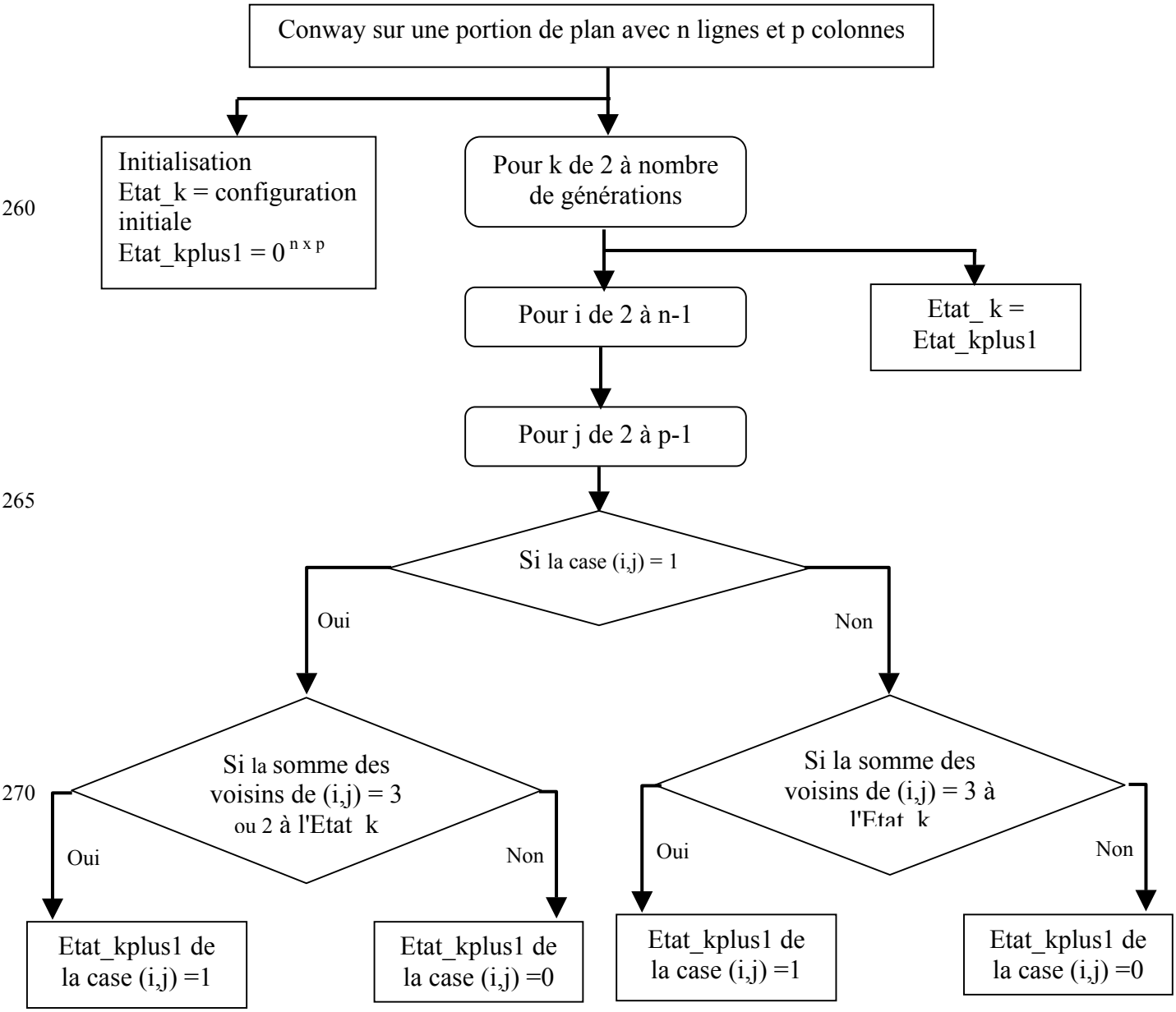

Figure 4 : Algorithme de l'automate de Conway

Pour illustrer l'automate de Conway, nous pouvons étudier l'évolution de générations en générations de la configuration initiale de la figure 5.

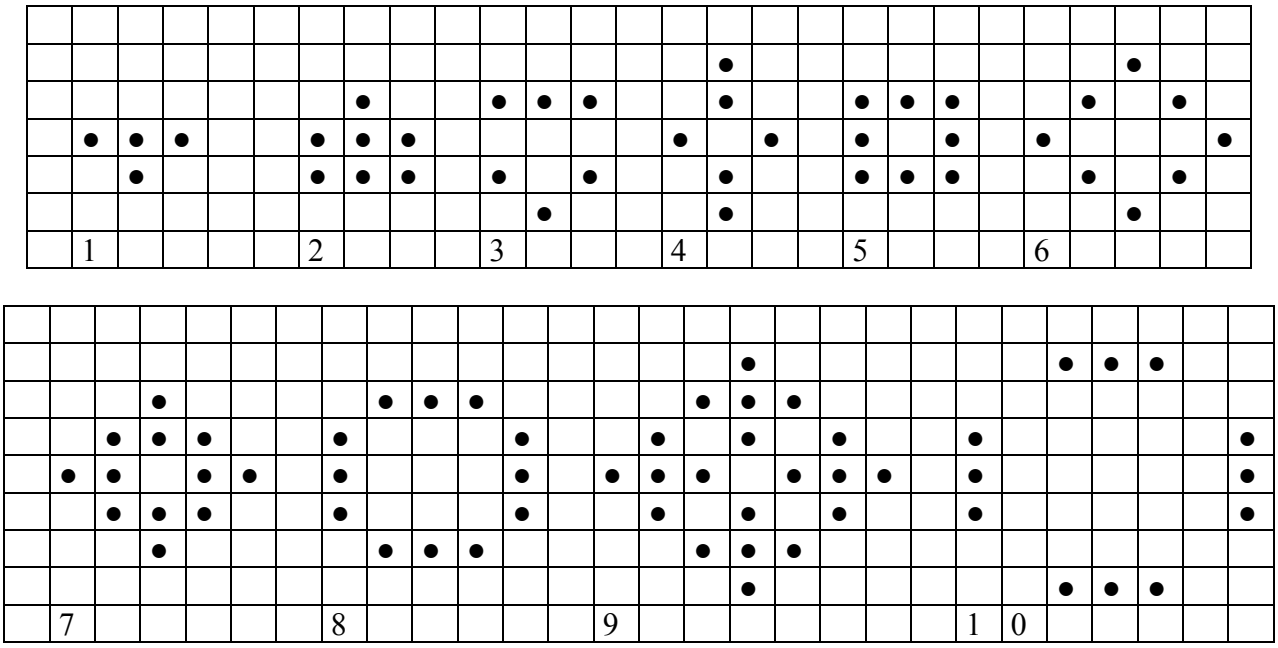

Figure 5 : Exemple d'évolution obtenue pour l'automate de Conway

Mois après mois, l'étude de l'automate de Conway se développa et finalement le nombre de résultats accumulés permit d'établir qu'aussi simple qu'il soit, ce modèle d'univers était malgré tout un objet d'indécidabilité.

Le problème le plus simple que l'on puisse se poser à propos d'un automate fini est: une configuration finie étant donnée, finit-elle par disparaître ou, au contraire, persiste-t-elle indéfiniment ? Pour l'automate "Déplacement Est", la réponse est immédiate, la configuration initiale est reproduite à l'identique de générations en générations: il y a donc persistance. D'autre part, en poursuivant l'exemple proposé sur la figure 5, on peut faire apparaître une $11^{ième}$ génération (que le candidat pourra calculer), puis si l'on itère encore une fois l'algorithme, on se rend compte que la $12^{ième}$ génération est identique à la génération 10: à partir de la génération 11, il y a donc oscillation entre 2 configurations. Dans ce cas précis, la configuration initiale ne persiste pas, mais il y a apparition d'un attracteur, c'est à dire d'un ensemble limite de configurations qui se répètent aussi loin que l'on veut dans une succession de générations. Un attracteur est donc composé d'une ou plusieurs configurations périodiques. Le chercheur finlandais Jarkko Kari a démontré que toute propriété de l'ensemble limite qui est vraie pour certains automates et fausse pour d'autres est indécidable. Parmi les

conséquences de ce résultat, on peut montrer que savoir pour un automate donné s'il possède un attracteur est indécidable.

Dans le cas général, la question de la persistance est elle aussi indécidable: aucun algorithme ne pourra jamais être suffisamment général pour prévoir le destin final d'une configuration initiale de l'automate de Conway.

Un autre problème indécidable lié aux automates finis est celui de l'existence d'un automate inverse. La question est la suivante : si un automate fait un certain travail, existe – t - il un autre automate, l'automate inverse, qui permette de revenir en arrière, ceci pour n'importe quelle configuration initiale ? Par exemple, l'automate "Déplacement Est" possède sans aucun doute un automate inverse, que l'on pourrait appeler "Déplacement Ouest". Au contraire, l'automate de Conway n'est pas inversible. En effet, si un automate possède un automate inverse, deux configurations différentes se transforment toujours en deux configurations différentes. Or l'automate de Conway avec la configuration initiale proposée sur la figure 5 ne possède pas cette propriété. Les générations 9 et 11 se transforment toutes deux en une même configuration (la configuration 10). Etant donné un automate fini, est-il facile de savoir si cet automate admet un automate inverse ? La réponse à ce problème été donnée récemment par J. Kari : déterminer si un automate fini possède un automate inverse est un problème indécidable. Personne ne pourra jamais écrire de programme informatique qui, prenant pour donnée un automate quelconque, fait un calcul et sans se tromper indique au bout d'un temps fini si l'automate est inversible ou non.

Un autre problème intéressant sur les réseaux d'automates finis est celui de l'existence de "Jardins d'Eden". Une configuration "Jardin d'Eden" est une configuration qui ne peut être le résultat d'aucune configuration antérieure. Si un automate est inversible, il ne possède pas de jardin d'Eden car bien sûr toute configuration a un prédécesseur obtenu en appliquant l'automate inverse. Il a été prouvé que l'automate de Conway possède des jardins d'Eden, mais il a été plus difficile d'en trouver un (figure 6).

Parmi les conséquences intéressantes du résultat obtenu par J Kari au sujet de l'indécidabilité de l'inversion, il en est une qui concerne le voisinage utile des automates inverses. Le voisinage utile d'un automate est défini comme l'ensemble des cases qui sont consultées par l'automate lorsqu'il change d'état. L'automate "Déplacement Est" ne consulte qu'une seule case : la case Ouest. L'automate de Conway consulte les 8 cases immédiatement voisines. Le résultat précis de J. Kary est en fait plus fort que ce que nous avons énoncé

précédemment : il stipule que savoir si un automate qui utilise seulement ses 8 cases voisines est inversible est un problème indécidable. Sous cette forme ce résultat entraîne que pour tout entier n (par exemple 1000), il existe un automate inversible n'utilisant que ces 8 voisins immédiats et dont l'automate inverse utilise des voisins à plus de 1000 cases de distance. En effet si les voisins à moins de 1000 cases de distance suffisaient toujours pour inverser un automate n'utilisant que ses 8 voisins immédiats, on pourrait écrire un programme qui déterminerait si un automate est inversible en essayant tous les automates dont les voisins utiles sont à moins de 1000 cases de distance. Ce programme contredirait le résultat de J. Kari.

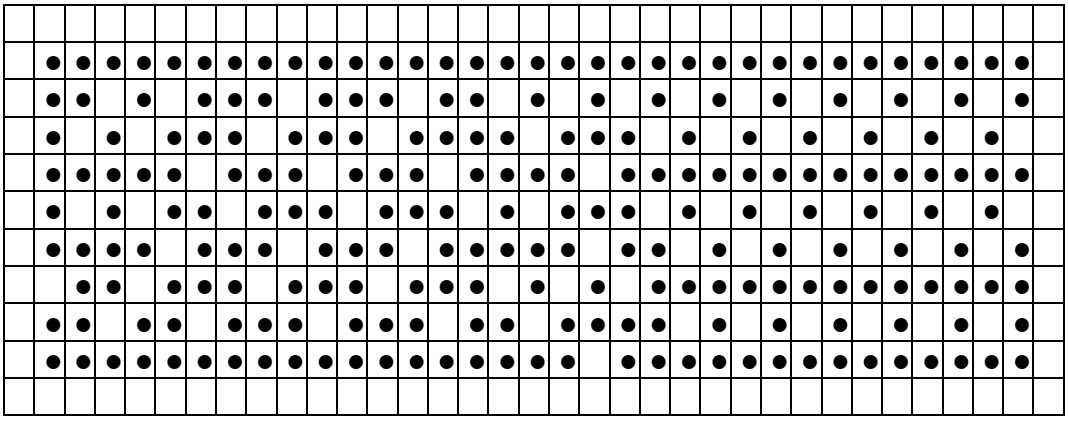

Figure 6 : Exemple de "Jardin d'Eden"

Cette difficulté pour inverser un automate suggère que l'on peut utiliser les automates finis pour concevoir des systèmes cryptographique à clé publique, c'est à dire dont la méthode de codage est connu de tous sans que la méthode de décodage le soit. L'idée proposée par le chercheur finlandais consiste à utiliser un automate inversible que l'on rend public en gardant pour soi l'automate inverse. La personne qui désire vous faire parvenir un message codé devrait procéder de la façon suivante: elle commence par traduire son message en une configuration. Ensuite, elle fait fonctionner le réseau d'automate pendant un grand nombre de générations. La clé de codage étant publique, tout le monde peut envoyer des messages. La personne vous transmet la configuration obtenue, sans avoir à la cacher puisque vous êtes le seul à pouvoir la déchiffrer. Pour décoder le message, vous utiliser l'automate inverse pendant le même nombre de générations. La difficulté prouvée par J. Kari du calcul de l'automate inverse d'un automate donné, vous assure que personne ne pourra facilement déchiffrer les messages qui vous sont destinés.

Les automates finis comme les machines de Turing sont des processus universels de calcul capables de représenter n'importe quel algorithme, et sujets tous deux à des problèmes

d'indécidabilité. On ne peut s'empêcher de constater que notre monde physique est bien plus complexe que celui de l'automate de Conway ou de la machine de Turing. Il ne fait donc pas de doute que, lui aussi, est sujet à une imprévisibilité fondamentale (sans qu'il soit pour autant nécessaire d'invoquer la mécanique quantique). Mais ces résultats concernant l'indécidabilité ne sont pas uniquement négatifs. Contrairement à ce que l'on a longtemps pensé, les mondes discrets, localement finis et homogènes que sont les mondes des automates finis et des machines de Turing sont riches et complexes, et leur étude doit être poursuivie.